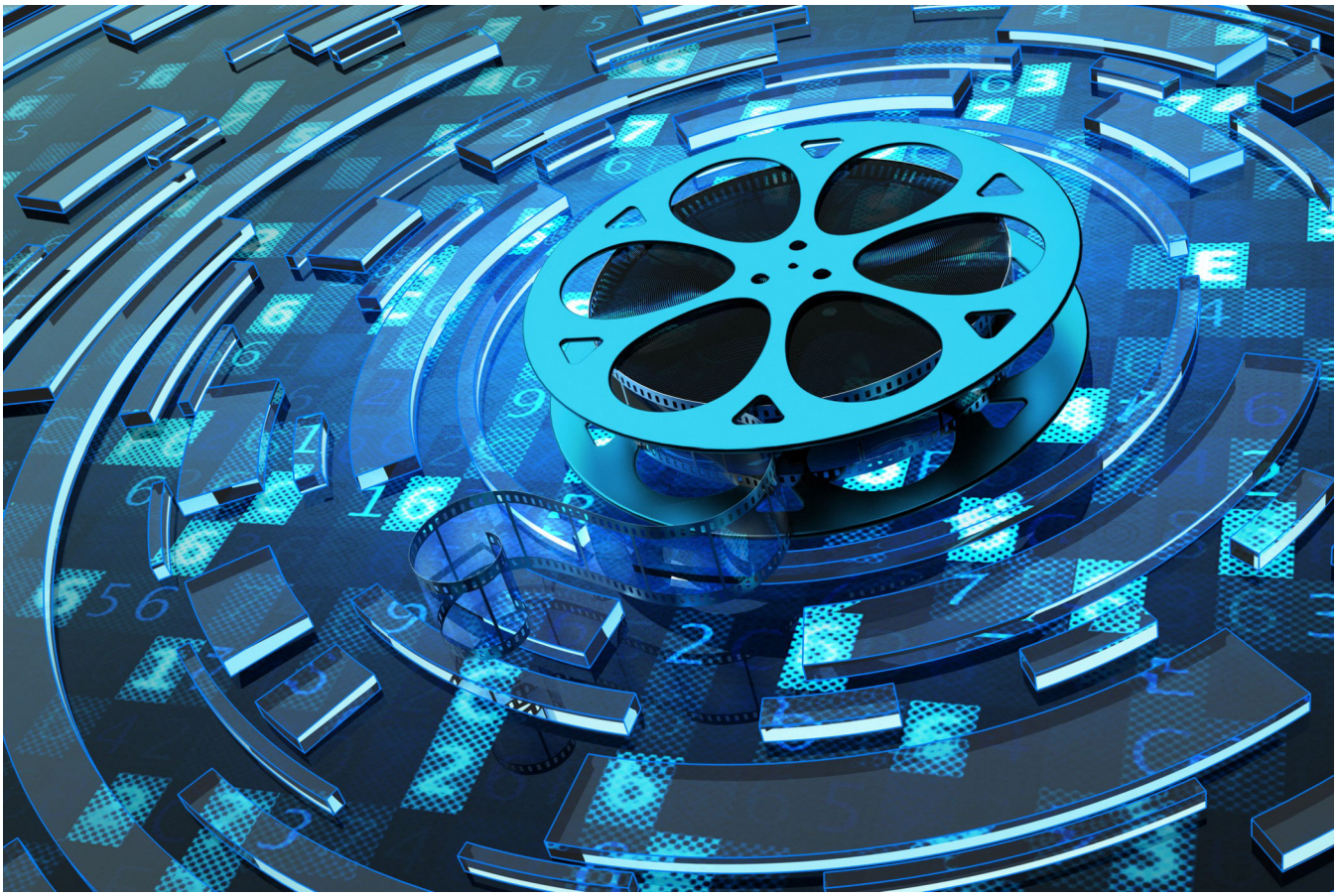
National and Kapodistrian
UNIVERSITY OF ATHENS

Video Stream Analytics Using OpenCV, Kafka and Spark Technologies Tutorial



University of Athens
Department of Informatics and Telecommunications
Pervasive Computer Group
RAWFIE Project
Author: Chalvatzaras Athanasios
Email: achalv@di.uoa.gr

Contents

Table of Contents

1. Introduction

Technology has brought an unprecedented explosion in unstructured data. Sources like mobile devices, websites, social media, scientific apparatus, satellites, IoT devices, and surveillance cameras are generating a vast number of images and videos every second.

Point of this tutorial is to build a system, which consists of three parts:

- a video stream collector
- a stream data buffer
- a video stream processor

Video stream collector : Receives the video stream data from a cluster of IP cameras, video files or raw http streams. Reads the input frame by frame, converts the frame to a Mat, and then serializes all the data of the Mat saving it in a 64-base encoded string, in order to pass it through the stream data buffer (Kafka-based).

Stream data buffer : Kafka distributed streaming platform. A fault tolerant data queue for streaming data (in this example video data).

Video Stream Processor : consumes the stream data from buffer and processes it. This component will apply video-processing algorithms to detect motion in the video-stream data.

Required tools

There will be brief explanation about installing and setting the required tools

1. Java Development Kit (JDK)
2. Maven
3. Zookeeper
4. Kafka
5. Spark
6. OpenCV
7. Eclipse

2. Installation of required tools

(The installation and execution guide refers to Linux users, and more specifically to Ubuntu users.)

Java development Kit (JDK)

- sudo apt-get install openjdk-8-jdk

Check if the installation is successful by using the command

- java -version

Maven

- sudo apt-get install maven

Check if the installation is successful by using the command

- mvn -version

Kafka and Zookeeper

Kafka and zookeeper are components of the Confluent package, so it is enough to just download and set only this package.
Installation steps:

- wget -qO - https://packages.confluent.io/deb/4.1/archive.key | sudo apt-key add -

- sudo add-apt-repository "deb [arch=amd64] https://packages.confluent.io/deb/4.1 stable main"

- sudo apt-get update && sudo apt-get install confluent-platform-oss-2.11

Kafka and zookeeper are now set. To start the services type in CLI:

- confluent start

Stop the services by executing:

- confluent stop

Spark

The requirement to run Spark is JDK and Scala. JDK was installed earlier in this tutorial, so only Scala needs to be installed.

In order to install Scala execute:

- sudo apt-get install scala

Now Spark:

- Download the latest release of Spark build with hadoop from the spark download page: https://spark.apache.org/downloads.html
- Unpack the archive
- Move the folder: sudo mv spark-2.1.1-bin-hadoop2.7 /usr/local/ (example)
- Create symbolic link: sudo ln -s /usr/local/spark-2.1.1-bin-hadoop2.7/ /usr/local/spark
  (example)
- Add SPARK_HOME to your environment: export SPARK_HOME=/usr/local/spark

That is all for Spark.

Eclipse

It is not completely necessary to install eclipse. The CLI can be also used to execute the example project.

- Download the latest version of eclipse for Ubuntu from here
- Unpack the archive
- Open terminal inside the folder and execute: sudo ./eclipse-inst
- Pick Eclipse IDE for Java EE Developers
- Install under your home directory

- Go into eclipse folder under your home folder and execute: sudo ./eclipse in order to run eclipse

OpenCV

Now the hard part. OpenCV has to be installed on the system. The project uses native libraries in order to run, so OpenCV should be built on the system.

At first before the OpenCV installation, some libraries should be installed on the system:

Refresh the repositories:

- sudo apt-get update
- sudo apt-get upgrade

Developer tools:

- sudo apt-get install build-essential cmake pkg-config git g++ gcc

Libraries that facilitate the I/O for various file formats:

- sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev

Libraries for video stream processing and camera frame access:

- sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
- sudo apt-get install libxvidcore-dev libx264-dev

GUI libraries:

- sudo apt-get install libgtk-3-dev

Optimization libraries:

- sudo apt-get install libatlas-base-dev gfortran

Python Development Headers:

- sudo apt-get install python2.7-dev python3.5-dev

ApacheAnt and CMake GUI

- sudo apt-get install ant
- sudo apt-get install cmake-qt-gui

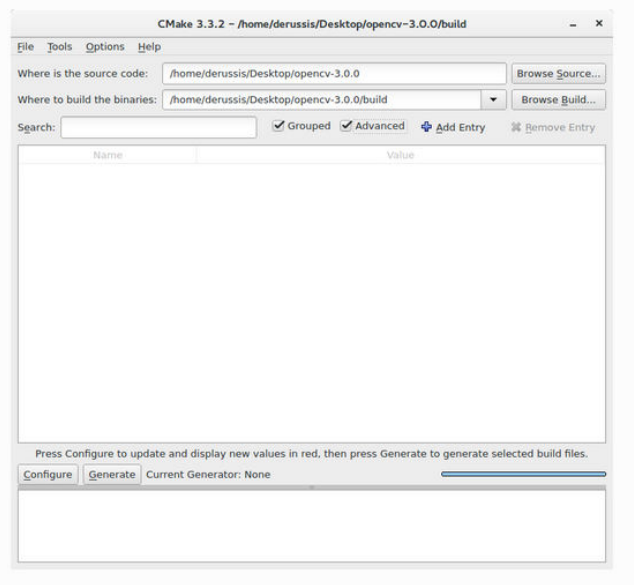
Everything is set to start the OpenCV installation!

Open the CLI, go to home folder and download the source code of the OpenCV 3.2.0 version :

- cd ~
- wget -O opencv.zip https://github.com/opencv/opencv/archive/3.2.0.zip
- unzip opencv.zip

Build OpenCV:

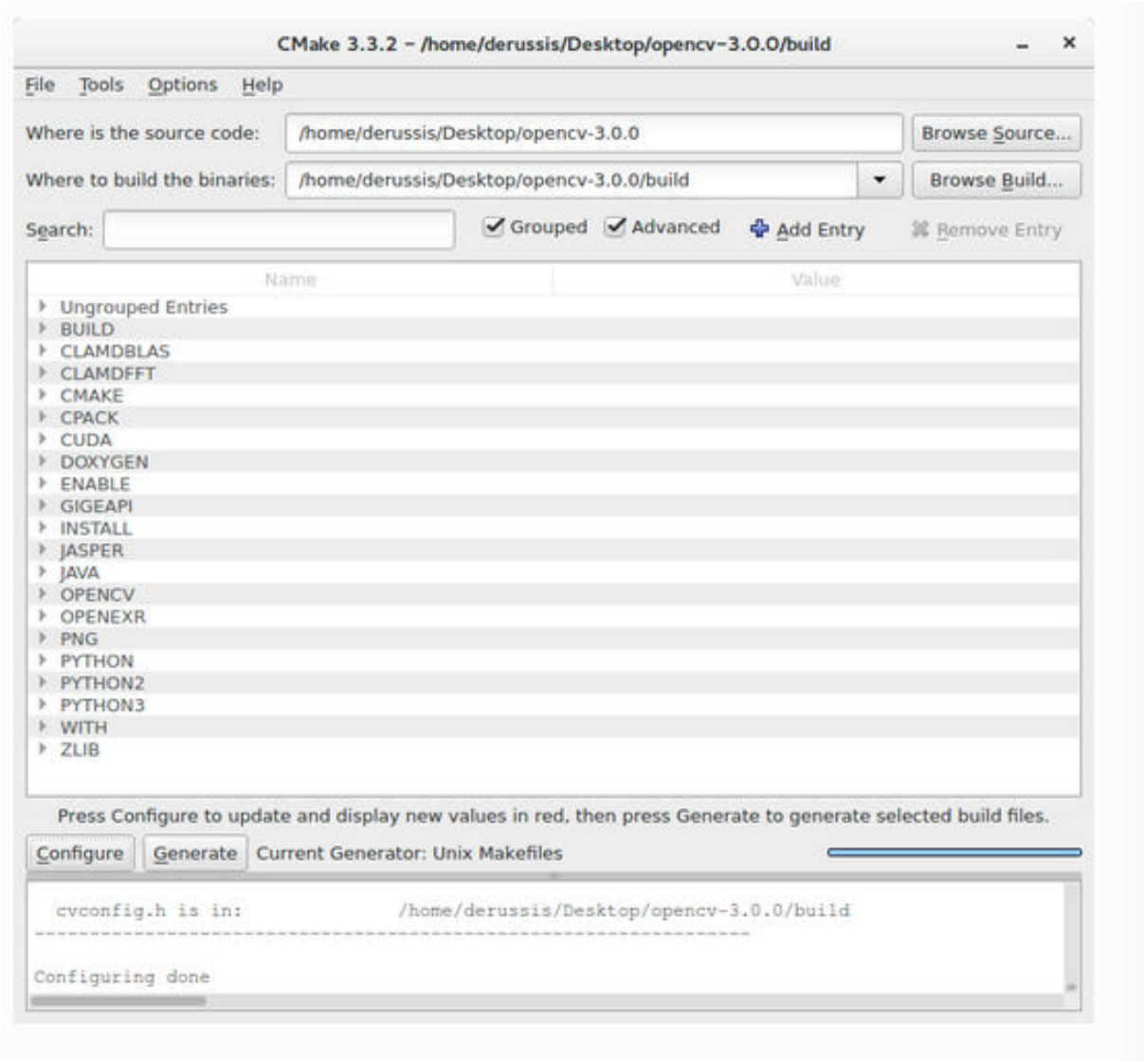
- Open CMake
- Where is the source code: <Your OpenCV folder path>
- Where to build the binaries: <Your OpenCV folder path>/build



- Check Grouped and Advanced checkboxes
- Press Configure and use the default compilers for Unix Makefiles

In the Ungrouped Entries group:

- insert the path to the Apache Ant executable (e.g., /apache-ant-1.9.6/bin/ant).



In the BUILD group, unselect:

- BUILD_PERF_TESTS
- BUILD_SHARED_LIBRARY (to make the Java bindings dynamic library all-sufficient)
- BUILD_TESTS
- BUILD_opencv_python

In the CMAKE group:

- set to Debug (or Release) the CMAKE_BUILD_TYPE

In the JAVA group:

- insert the Java AWT include path (e.g., /usr/lib/jvm/java-1.8.0/include/)
- insert the Java AWT library path (e.g., /usr/lib/jvm/java-1.8.0/include/jawt.h)
- insert the Java include path (e.g., /usr/lib/jvm/java-1.8.0/include/)
- insert the alternative Java include path (e.g., /usr/lib/jvm/java-1.8.0/include/linux)
- insert the JVM library path (e.g., /usr/lib/jvm/java-1.8.0/include/jni.h)

Press Configure, then press Generate and close CMake

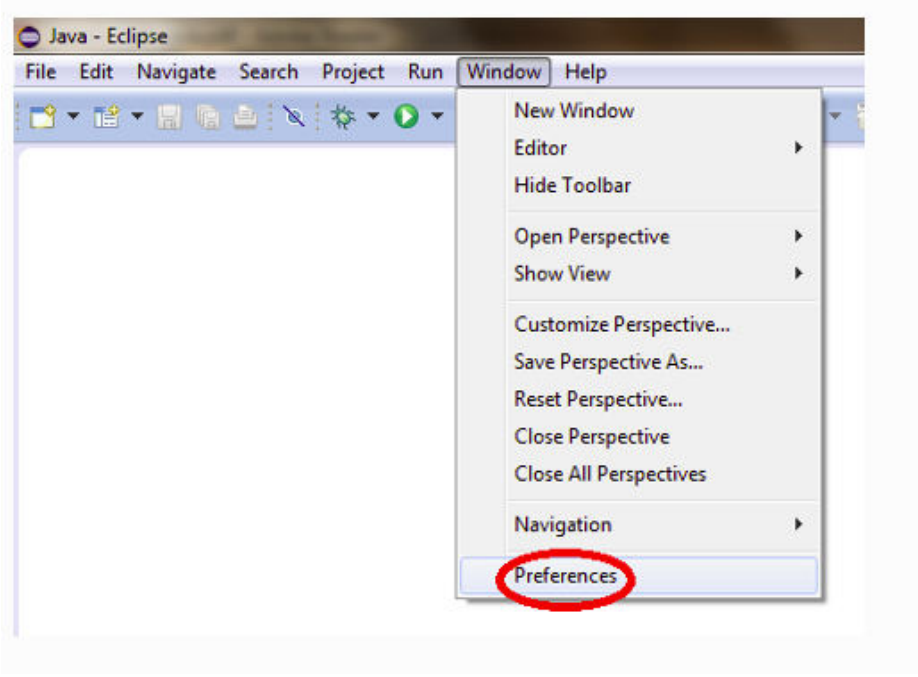
Build OpenCV

- Open terminal and go to the build folder of OpenCV
- make -j

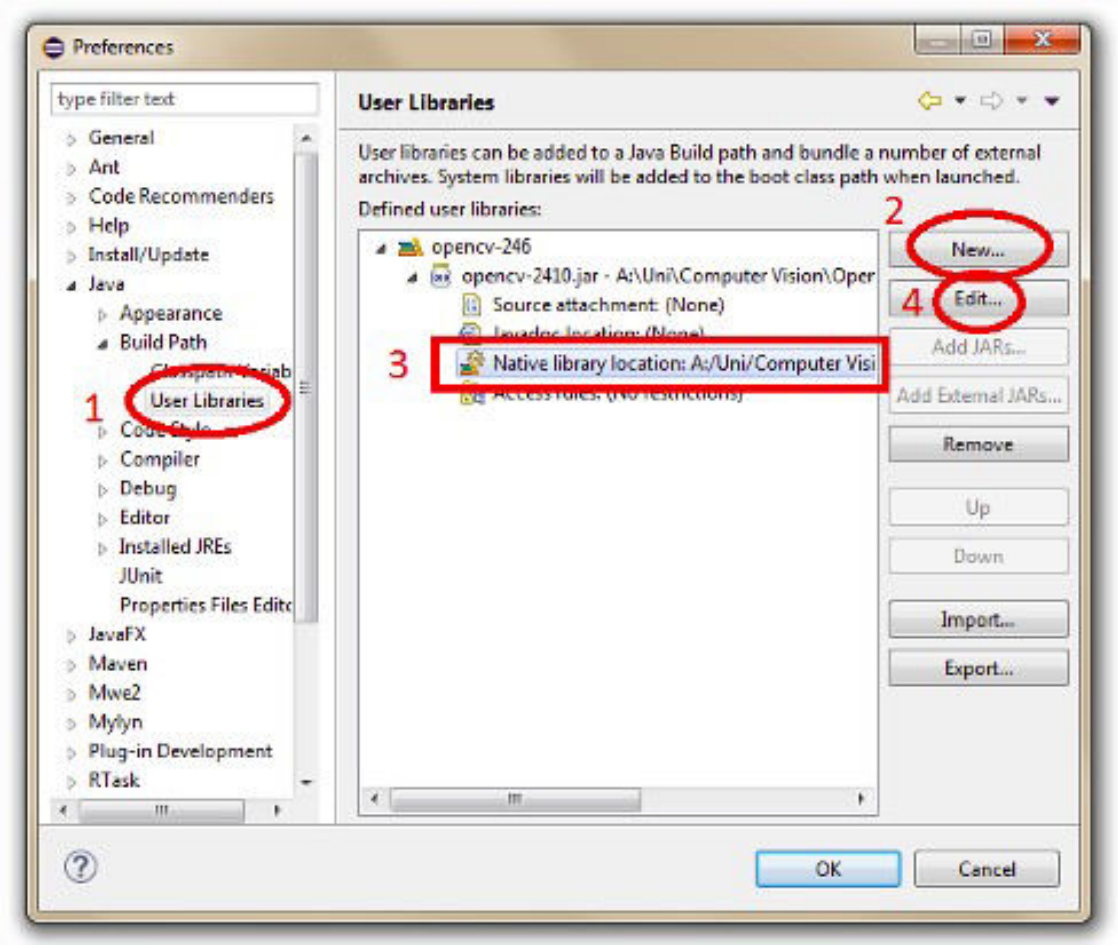
If everything went well you should have opencv-3xx.jar in the /opencv/build/bin directory and libopencv_java3xx.so in the /opencv/build/lib directory. The 3xx suffix of each file is a shortcut for the current OpenCV version, e.g., it will be 300 for OpenCV 3.0 and 330 for OpenCV 3.3. This is everything you need.

Setting up OpenCV for Java in Eclipse

- Create a User Library, ready to be used on all your next projects: go to Window > Preferences…



- From the menu navigate under Java > Build Path > User Libraries
- Choose New....
- Enter a name for the library (e.g., opencv)
- Select the newly created user library
- Choose Add External JARs...
- Browse to select opencv-3xx.jar from your computer
- After adding the jar, extend it, select Native library location and press Edit…
- Select External Folder...
- Browse to select the folder containing the OpenCV libraries



3. Example project explanation

System Architecture

The architecture diagram of video stream analytics system is illustrated in Figure 1 below.

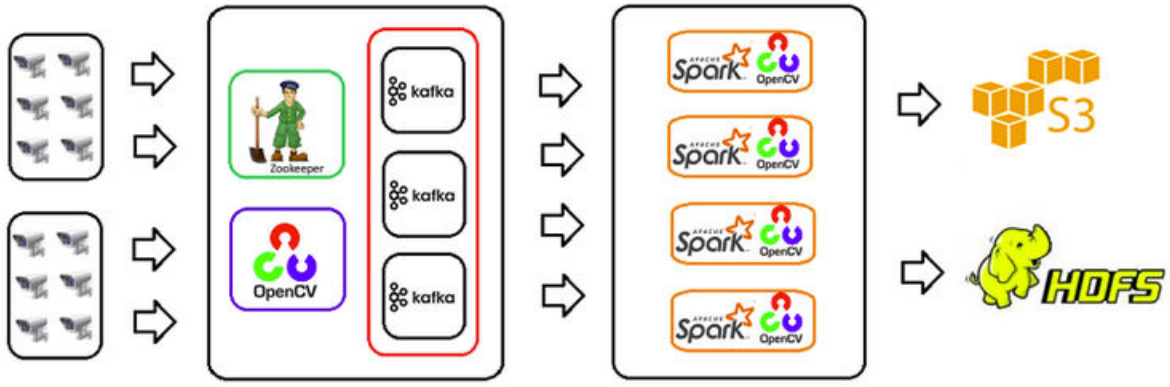


Figure 1. Video Stream Analytics System Architecture Diagram

Video Stream Collector

The video stream collector works with a cluster of IP cameras that provide live video feeds. The component must read the feed from each camera and convert the video stream into a series of video frames. To distinguish each IP camera, the collector maintains the mapping of camera ID and URL with camera.url and camera.id properties in a stream-collector.properties file. These properties can have comma-separated lists of camera URLs and IDs. Different cameras may provide data with different specifications such as the codec, resolution, or frames per second. The collector must retain these details while creating frames from the video stream.

The video stream collector uses the OpenCV video-processing library to convert a video stream into frames. Each frame is resized to the required processing resolution (e.g. 640x480). OpenCV stores each frame or image as a Mat object. Mat needs to be converted in serialise-able (byte-array) form by keeping intact the details of frame —

i.e. rows, columns, and type. The video stream collector uses the following JSON message structure to store these details.

{"cameraId":"cam-01","timestamp":1488627991133,"rows":12,"cols":15,"type":16,"data":"asdfh"}

cameraId is the unique ID of the camera. timestamp is the time at which the frame was generated. rows, cols, and type are OpenCV Mat-specific details. data is a base-64 encoded string for the byte array of the frame.

The video stream collector uses the Gson library to convert the data to JSON messages, which are published in the video-stream-event topic. It sends the JSON messages to the Kafka broker using the KafkaProducer client. KafkaProducer sends data into the same partition for each key and order of these messages is guaranteed.

```
JsonObject obj = new JsonObject();
obj.addProperty("cameraId",cameraId);
obj.addProperty("timestamp", timestamp);
obj.addProperty("rows", rows);
obj.addProperty("cols", cols);
obj.addProperty("type", type);
obj.addProperty("data", Base64.getEncoder().encodeToString(data));
String json = gson.toJson(obj);
producer.send(new ProducerRecord<String, String>(topic,cameraId,json),new EventGenerator
```

Kafka is primarily designed for text messages of small sizes but a JSON message comprising the byte array of a video frame will be large (e.g. 1.5 MB), so Kafka will require configuration changes before it can process these larger messages. The following KafkaProducer properties need to be adjusted:

- batch.size
- max.request.size
- compression.type

(The code automatically does that)

Stream Data Buffer

To process a huge amount of video stream data without loss, it is necessary to store the stream data in temporary storage. The Kafka broker works as a buffer queue for the data that the video stream collector produces. Kafka uses the file system to store the messages, and the length of time it retains these messages is configurable.

Keeping the data in storage before processing ensures its durability and improves the overall performance of the system as processors can process data at different times and at different speeds depending on the load. This improves the reliability of the system when the rate of data production exceeds the rate of data processing.

Kafka guarantees the order of messages in a single partition for a given topic. This is extremely helpful for processing data when the order of the data is important. To store large messages, the following configurations might need to be adjusted in the server.properties file of the Kafka server:

- message.max.bytes
- replica.fetch.max.bytes

(The code automatically does that)

Video Stream Processor

The video stream processor performs three steps:

Read the JSON messages from the Kafka broker in the form of a VideoEventData dataset.
Group the VideoEventData dataset by camera ID and pass it to the video stream processor.
Create a Mat object from the JSON data and process the video stream data.

The video stream processor is built on Apache Spark. Spark provides a Spark Streaming API, which uses a discretized stream or DStream, and a new Structured Streaming API based on a dataset. This application’s video stream processor uses the Structured Streaming API to consume and process JSON messages from Kafka. Please note this application processes structured data in the form of JSON messages and the unstructured video data is an attribute of these JSON messages that the video stream processor will process. The Spark documentation states "Structured Streaming provides fast, scalable, fault-tolerant, end-to-end exactly-once stream processing

without the user having to reason about streaming." This is why the video stream processor is designed around Spark's Structured Streaming. The Structured Streaming engine provides built-in support for structured text data and state management for aggregation queries. This engine also provides features like processing of non-aggregate queries and external state management of datasets (a new feature in Spark 2.2.0).

To process large messages, the following Kafka consumer configurations must be passed to the Spark engine:

- max.partition.fetch.bytes
- max.poll.records

The main class for this component is VideoStreamProcessor. This class first creates a SparkSession object that is the entry point for working with the Spark SQL engine. The next step is to define a schema for incoming JSON messages so that Spark can use this schema to parse the string format of a message into JSON format. Spark's bean encoder can transform this into Dataset<VideoEventData>. VideoEventData is a Java bean class that holds the data of JSON message.

```
Dataset<VideoEventData> ds = spark.readStream().format("kafka")
.option("kafka.bootstrap.servers",prop.getProperty("kafka.bootstrap.servers"))
.option("subscribe",prop.getProperty("kafka.topic"))
.option("kafka.max.partition.fetch.bytes",prop.getProperty("kafka.max.partition.fetch.by
.option("kafka.max.poll.records", prop.getProperty("kafka.max.poll.records"))
.load().selectExpr("CAST(value AS STRING) as message")
.select(functions.from_json(functions.col("message"),schema).as("json"))
.select("json.*").as(Encoders.bean(VideoEventData.class));
```

Next, groupByKey groups the dataset by camera ID to get KeyValueGroupedDataset<String, VideoEventData>. It uses a mapGroupsWithState transformation to work on a group of VideoEventData (Iterator<VideoEventData>) for the current batch of video frames that are grouped by camera ID. This transformation first checks that the last processed VideoEventData (video frame) is present and passes that to the video processor for next step of processing. After video processing, the last processed VideoEventData (video frame) is returned from the video processor and the state updates. To start the streaming application, the writeStream method is called on the dataset with console sink and update output mode.

The video stream processor uses the OpenCV library to process video stream data. Our application is meant to detect motion; VideoMotionDetector is the class with the

logic for detecting motion in a series of frames. The first step in this process is to sort the list of VideoEventData (Iterator<VideoEventData>) by timestamp for a given camera ID to compare video frames in order. The next step is to iterate the sorted list of VideoEventData objects and convert them to an OpenCV Mat object. If the last processed video frame is available, then it uses that as the first video frame for processing the current series of frames. VideoMotionDetector compares two consecutive frames and detects the differences using an API provided by the OpenCV library. If it finds differences that exceed defined criteria, these are considered to be motion. VideoMotionDetector will save this detected motion in form of image file to a preconfigured S3 bucket or HDFS directory. This image file can undergo further processing by another application or VideoMotionDetector can trigger an event to notify a user or application it has detected motion.

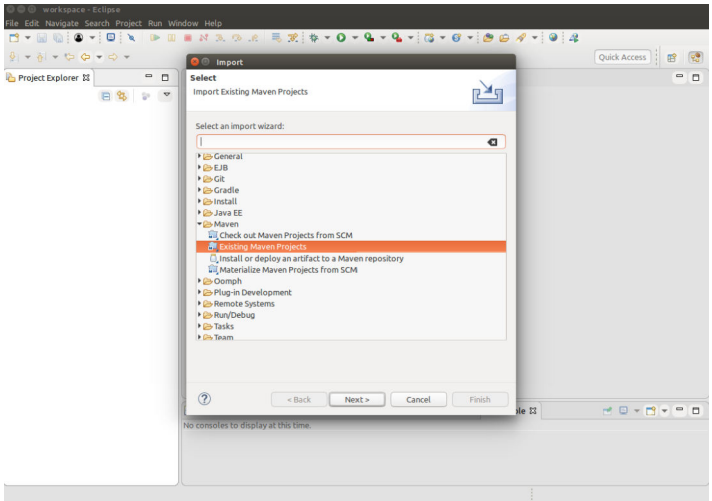
4. Download Project Source Code and Execute

- Download the source poject code from: https://github.com/baghelamit/video-stream-analytics
- The stream-collector.properties file has the Kafka topic as video-stream-event. Create this topic and partitions in Kafka. Use the kafka-topic command to create the topic and partitions.

```
kafka-topics --create --zookeeper localhost:2181 --topic
video-stream-event --replication-factor 1 --partitions 3
```

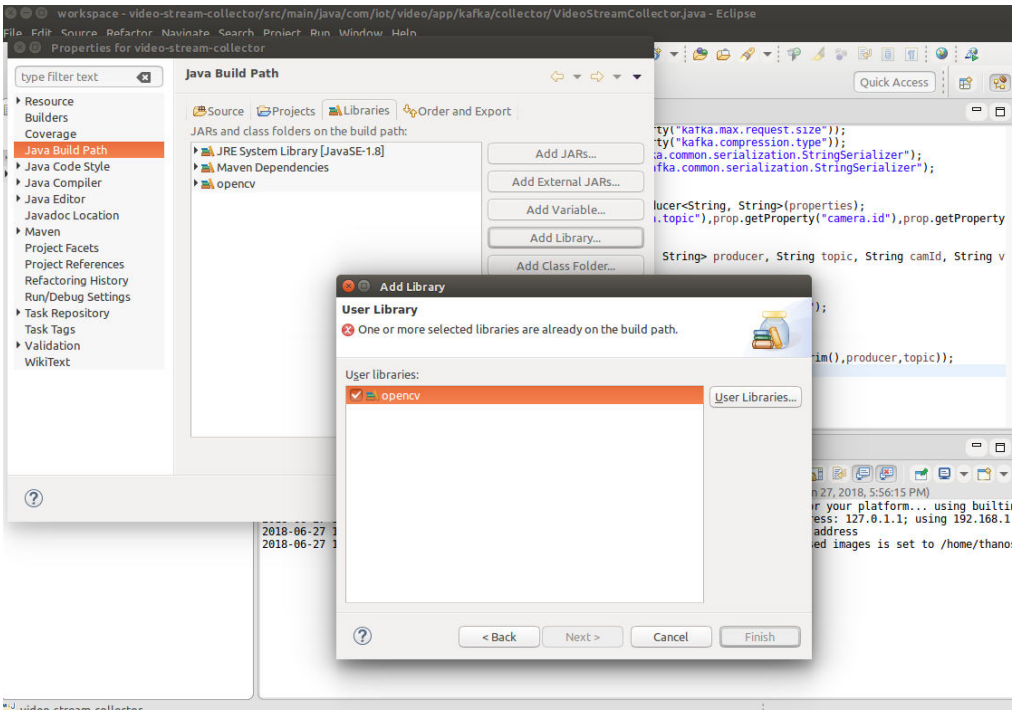
```
(!!!Warning every time you restart the pc, the topic is gone,
because by default the folder is the tmp folder!!! You can
change the folder that topics are saved in order to avoid
this!!!)
```

- make sure confluent is up and running (confluent start on CLI)
- Open eclipse
- Import both maven projects in the workspace

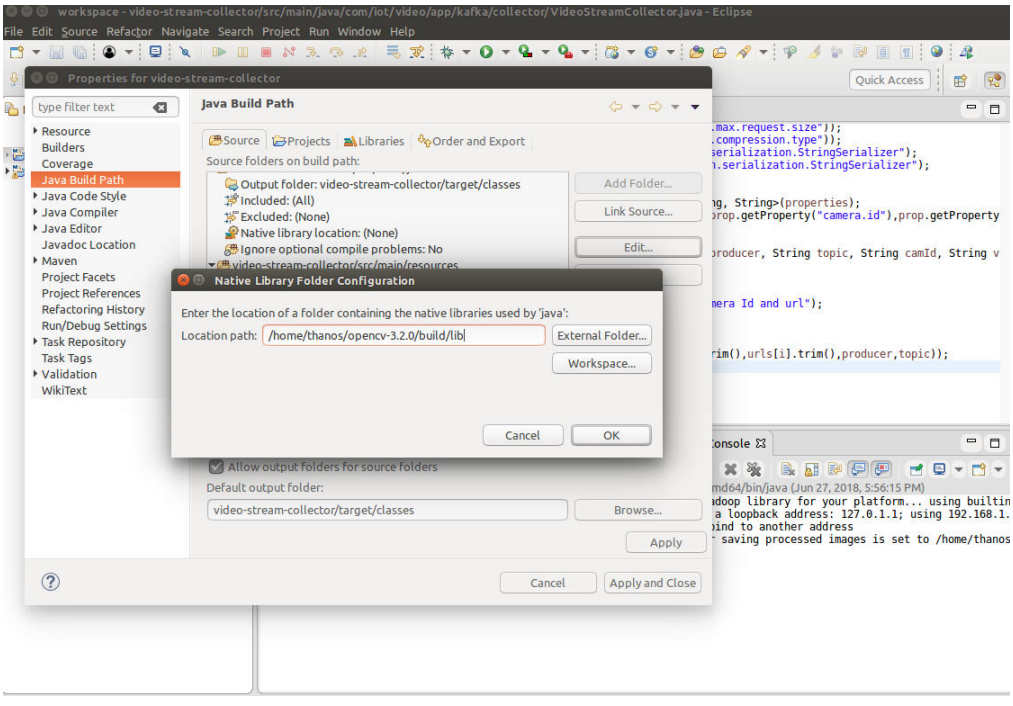


Add the OpenCV library jar and path to both projects

- Right Click on the project > Build Path > Configure Build Path > Java Build Path > Add library > User Library and check opencv that you created earlier, apply and close



- Right Click on the project > Build Path > Configure Build Path > Java Build Path > Source Tab , for every package edit Native library location to point on OpenCV libraries in home folder



Build Both Projects

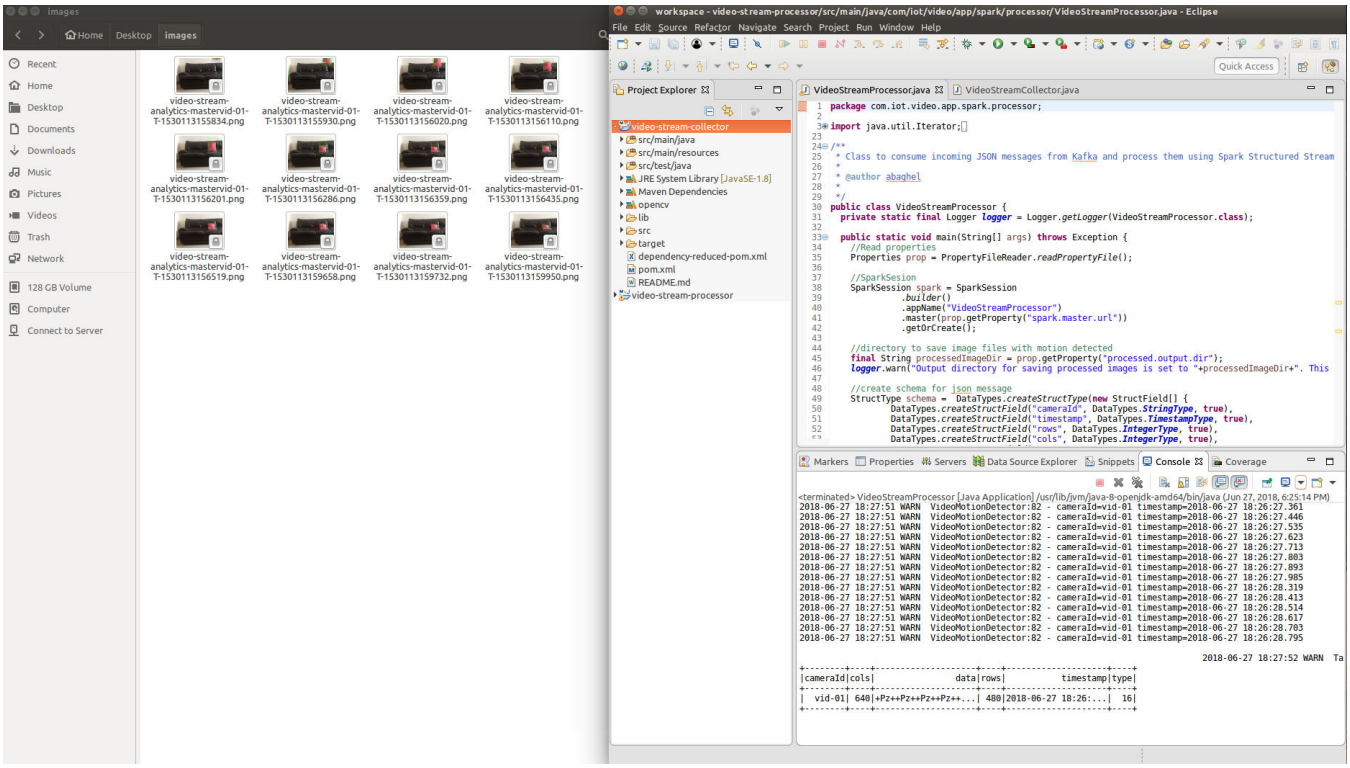
- Right Click > Run As > Maven Build , on the goal write “clean install” and click apply and Run.

Executing

- To keep the application logic simple, `VideoStreamProcessor` processes only new messages. The `VideoStreamProcessor` component should be up and running before starting the `VideoStreamCollector` component. So execute first VideoStreamCollector

- Then execute VideoStreamProcessor.

5. Example Execution And Results



6. Conclusion

Large-scale video analytics of video streams requires a robust system backed by big-data technologies. Open-source technologies like OpenCV, Kafka, and Spark can be used to build a fault-tolerant and distributed system for video stream analytics. We used OpenCV and Kafka to build a video stream collector component that receives video streams from different sources and sends them to a stream data buffer component. Kafka serves as the stream data buffer component that provides durable storage of streaming data. The video stream processor component is developed using OpenCV and Spark's Structured Streaming. This component receives streaming data from the stream data buffer and analyses that data. The processed files are stored in a preconfigured HDFS or S3 bucket.

7. References and Sources

- https://www.infoq.com/articles/video-stream-analytics-opencv
- https://docs.confluent.io/current/installation/installing_cp.html
- https://datawookie.netlify.com/blog/2017/07/installing-spark-on-ubuntu/
- https://medium.com/@josemarcialportilla/installing-scala-and-spark-on-ubuntu-5665ee4b62b1
- http://opencv-java-tutorials.readthedocs.io/en/latest/01-installing-opencv-for-java.html
- https://milq.github.io/install-opencv-ubuntu-debian/
- https://medium.com/@debugvn/installing-opencv-3-3-0-on-ubuntu-16-04-lts-7db376f93961
- https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/