AUGUST 2018

Develop streaming pipelines and analytics solutions for CERN's IoT Platform

AUTHOR:

Natacha Quintero
Oracle/IT-DB-SAS

SUPERVISORS:

Manuel Martin
Marquez
Jose Carlos Luna
Duran

CERN openlab

ABSTRACT

There are two very popular concepts that we hear in the world of technology, Big Data and Internet of Things. Big data is referring to a data which size, complexity and velocity is really high and is difficult to capture, pre-process and analyze it with conventional technologies. As a response to this new demands, a set of new technologies were developed to manage these new data flow problems. The second concept used is Internet of things,the network of physical devices, vehicles, and other items embedded with electronics, software, sensors and connectivity which enables these things to connect and exchange data.

The purpose of the project is to create data pipelines using Big Data technologies for Iot systems and asses how each one this technologies works and compare them, and made decisions of how those works in the context of the iot system.

TABLE OF CONTENTS

INTRODUCTION

As John Naisbitt stated “We are drowning in information but starved for knowledge”, we are living in a world where Data is the new gold and the volume of information that we can get from Data is getting bigger every day. The needs in the world of technology are constantly changing and the use of data becomes increasingly important in decision making. While the needs grow, the data also grow too and for this reason the concept of Big Data was born.

Big Data is a term that describes the large volume of data, both structured and unstructured, that is part of businesses every day. The collection of large amounts of data and the search for trends and hidden patterns within the data, allow companies to move quickly, smoothly and efficiently in decision making. It also allows them to avoid problem before they are really happening. Big Data analysis helps organizations take advantage of their data and use it to identify new opportunities.Doing this, it is possible to make smarter movements, more efficient operations and higher profits.

Due to the existence of Big Data, it was necessary to create new technologies that made it possible to manage this data and its exploitation. In the project that will be discussed below, technologies such as Kafka, Flume, Telegraf, InfluxDB and Grafana were used. Another important concept related to the project is Internet of Things, that basically is network of physical devices, vehicles, and other items embedded with electronics, software, sensors and connectivity which enables these things to connect and exchange data. The idea of the project was the development of pipelines for the resolution of a use case with the Iot platform of CERN. Data from sensors must travel through these pipelines to be monitored and visualized. The aforementioned tools were chosen in order to assess them, learn how they work and make a decision toward the implementation of a final solution.

BACKGROUND INFORMATION

During the development of this project we used the following technologies:

- ***Telegraf*** is a plugin-driven server agent for collecting & reporting metrics, and is the first piece of the TICK stack. Telegraf has plugins to source a variety of metrics directly from the system it's running on, pull metrics from third party APIs, or even listen for metrics via a statsd and Kafka consumer services. It also has output plugins to send metrics to a variety of other datastores, services, and message queues, including InfluxDB, Graphite, OpenTSDB, Datadog, Librato, Kafka, MQTT, NSQ, and many others.

  Key features

  Here are some of the features that Telegraf currently supports that make it a great choice for metrics collection.

  - Written entirely in Go. It compiles into a single binary with no external dependencies.
  - Minimal memory footprint.
  - Plugin system allows new inputs and outputs to be easily added.
  - A wide number of plugins for many popular services already exist for well known services and APIs.

  Telegraf is able to parse the following input data formats into metrics:

  1. InfluxDB Line Protocol
  2. JSON
  3. Graphite
  4. Value, ie: 45 or “booyah”
  5. Nagios
  6. Collectd
  7. Dropwizard

  Telegraf metrics, like influxDB points, are a combination of four basic parts:

  1. Measurement name
  2. Tags

3. Fields
4. Timestamp

- ***Flume*** is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.

- ***Apache Kafka*** is a distributed streaming platform, that it means three key capabilities:

  - Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
  - Store streams of records in a fault-tolerant durable way.
  - Process streams of records as they occur.

  Kafka is generally used for two broad classes of applications:

  - Building real-time streaming data pipelines that reliably get data between systems or applications.
  - Building real-time streaming applications that transform or react to the streams of data

  To understand how Kafka does these things, let's dive in and explore Kafka's capabilities from the bottom up. **First a few concepts:**

  - A Kafka cluster consists of one or more servers (Kafka brokers), which are running Kafka.
  - Kafka is run as a cluster on one or more servers that can span multiple datacenters.
  - The Kafka cluster stores streams of *records* in categories called *topics*.
  - Topic: A Topic is a category/feed name to which messages are stored and published.
  - Topic partition: Kafka topics are divided into a number of partitions, which allows you to split data across multiple brokers.

- Replicas A replica of a partition is a "backup" of a partition. Replicas never read or write data. They are used to prevent data loss.
- Producer: Application that sends the messages.
- Consumer: Application that receives the messages.
- Message: Information that is sent from the producer to a consumer through Apache Kafka.
- Connection: A connection is a TCP connection between your application and the Kafka broker.
- Consumer Group: A consumer group includes the set of consumer processes that are subscribing to a specific topic.
- Offset: The offset is a unique identifier of a record within a partition. It denotes the position of the consumer in the partition.

Kafka has four core APIs:

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

- ***InfluxDB*** is an open-source time series database developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. It also has support for processing data from Graphite.

- ***Grafana*** is a beautiful dashboard for displaying various Graphite metrics through a web browser. Grafana is nice because it is simple to set up and maintain and is easy to use and displays metrics in a very nice Kibana like display style.

PROJECT DETAILS

SOLUTION ARCHITECTURE

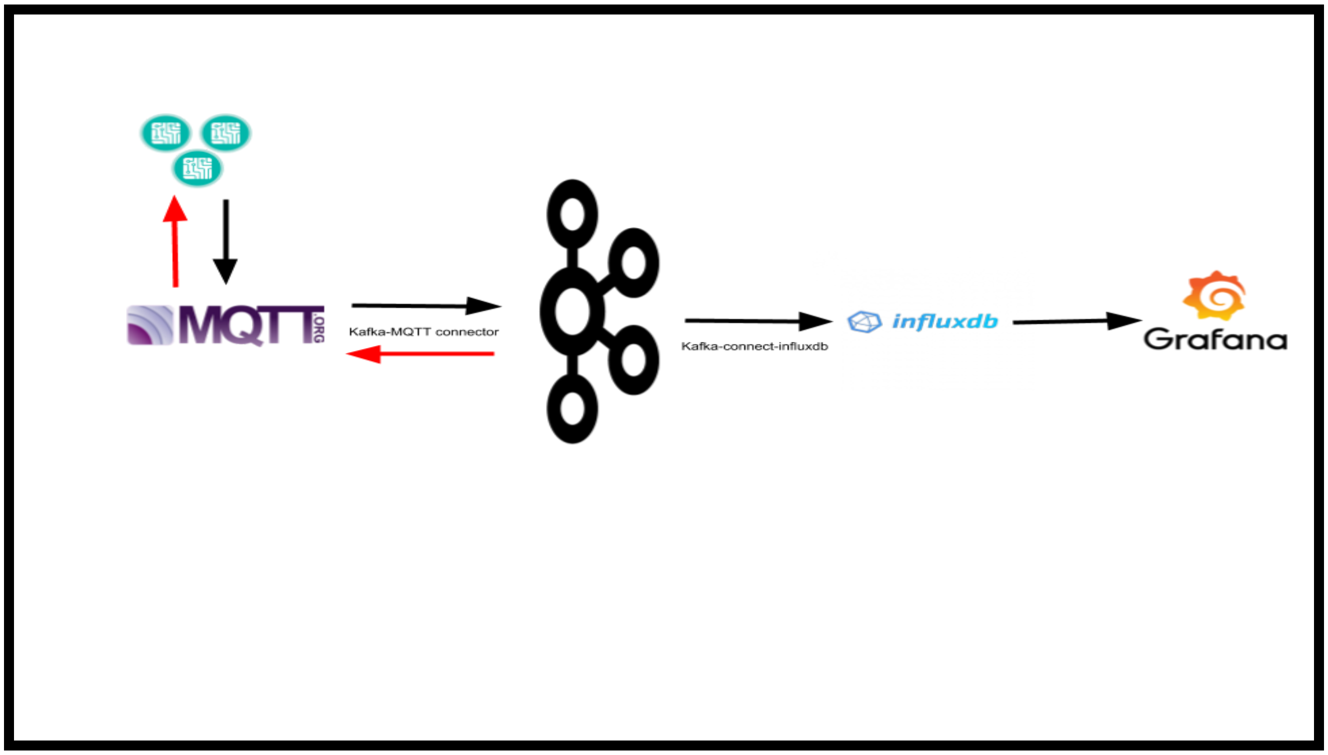


Figure 1: Solution Architecture.

The figure 1 describes a draft solution. First we can see some sensors with IoT data and send it to MQTT. After this, the data will flow to a kafka cluster with brokers. The data will be store in influxDB using kafka-connect-influxDB. Finally the data will be show in Grafana. The final idea is create some alerts and alarms for monitoring.

PHASE #1 AND RESULTS

The project was developed in two phases. The phase number one consisted in using Telegraf and flume as a generator. With this task, i managed to get experience of how these technologies works.

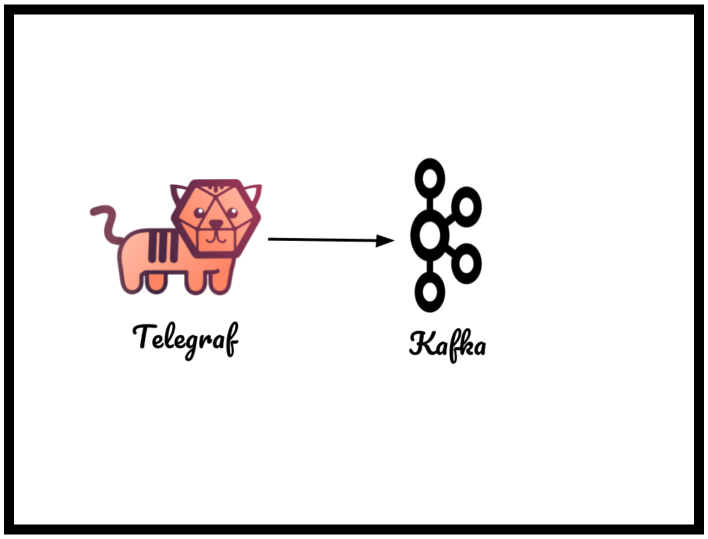


Figure 2: Telegraf to Kafka

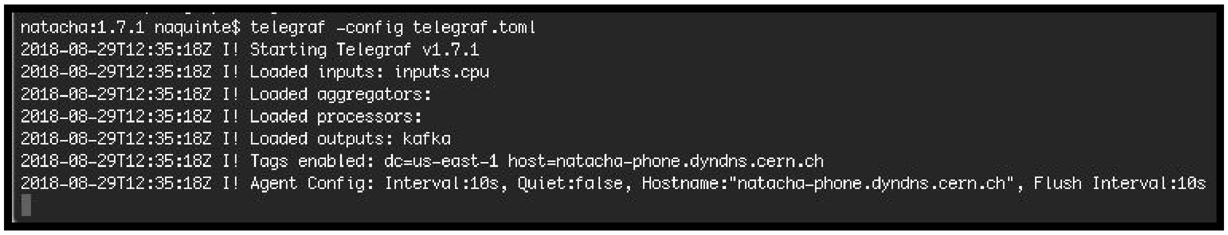
1. The first task consisted in using **Telegraf to send Data to Kafka**. It is important to say that Telegraf use SSL certificates as authentication. I worked with version of Kafka 2.11-1.1.1.

   We can find the configuration file as:

```
[tags]
   dc = "us-east-1"
# OUTPUTS
[outputs]
#[outputs.influxdb]
#   urls = ["https://XXXXX:port"]
#   database = "XXXX"
#   username="user"
#   password="XXXXXXXXXXXXXX"
```

```
[[outputs.kafka]]
    brokers = ["broker1:port","broker2:port","broker3:port"]
    topic = "topic name"
    required_acks = -1
    max_retry = 3

#
# Optional SSL Config
ssl_ca = "/System/Library/OpenSSL/certs/ca.pem"

ssl_cert ="/System/Library/OpenSSL/certs/c.crt.pem"

ssl_key = "/System/Library/OpenSSL/private/c.key.pem"

# Use SSL but skip chain & host verification*
insecure_skip_verify = true
#data_format = "influx"

# PLUGINS
# Read metrics about cpu usage
[cpu]
   percpu = false
   totalcpu = true
```

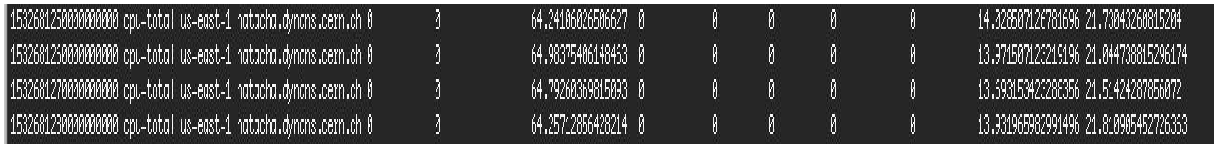
~



```
natacha:1.7.1 naquinte$ telegraf -config telegraf.toml
2018-08-29T12:35:18Z I! Starting Telegraf v1.7.1
2018-08-29T12:35:18Z I! Loaded inputs: inputs.cpu
2018-08-29T12:35:18Z I! Loaded aggregators:
2018-08-29T12:35:18Z I! Loaded processors:
2018-08-29T12:35:18Z I! Loaded outputs: kafka
2018-08-29T12:35:18Z I! Tags enabled: dc=us-east-1 host=natacha-phone.dyndns.cern.ch
2018-08-29T12:35:18Z I! Agent Config: Interval:10s, Quiet:false, Hostname:"natacha-phone.dyndns.cern.ch", Flush Interval:10s
```

Figure 3: Telegraf

In this configuration file, we are sending data to Kafka and InfluxDB too. The code to send data to influxdb are commented.



```
1532681250000000000 cpu-total us-east-1 natacha.dyndns.cern.ch 0      0        64.2410602650627  0      0     0      0        14.02850712678169 6 21.7304326081520 4
1532681260000000000 cpu-total us-east-1 natacha.dyndns.cern.ch 0      0        64.9837540614846 3  0      0     0      0        13.97150712321919 6 21.0447388152961 74
1532681270000000000 cpu-total us-east-1 natacha.dyndns.cern.ch 0      0        64.7926036981509 3  0      0     0      0        13.69315342328835 6 21.5142428785607 2
1532681280000000000 cpu-total us-east-1 natacha.dyndns.cern.ch 0      0        64.2571285642821 4  0      0     0      0        13.93196598299149 6 21.8109054527263 63
```

Figure 4: Data in InfluxDB

In the figure number 3 we can see that telegraf was working correctly and the CPU data was sending to kafka and in figure number 4, we can see some data inserted in InfluxDB. Telegraf was configure to authenticate with SSL certificates.

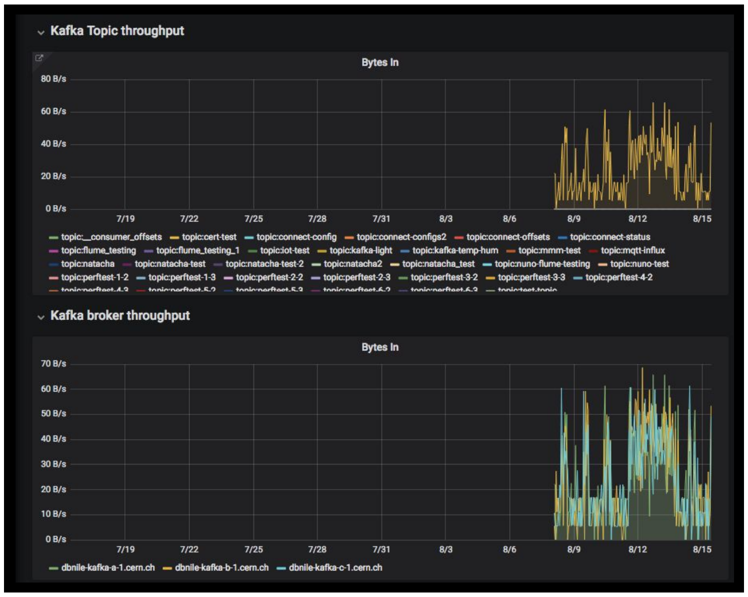


Figure 5: Data received from Telegraf to Kafka

2. The second task consisted in using Flume to send data to Kafka. We can see the two configuration files. The first one represent the sink and the second one represent the source.

```
#Agent Configuration

flume_agent.channels = memory_channel
flume_agent.sources = stresssource-1
flume_agent.sinks = kafka_sink

#Channel
flume_agent.channels = memory_channel
flume_agent.channels.memory_channel.type = memory

#Flume Source
flume_agent.sources.stresssource-1.type =
org.apache.flume.source.StressSource
flume_agent.sources.stresssource-1.channels = memory_channel
flume_agent.sources.stresssource-1.size = 100

# Flume Sink
flume_agent.sinks.kafka_sink.type =
org.apache.flume.sink.kafka.KafkaSink
flume_agent.sinks.kafka_sink.channel = memory_channel
flume_agent.sinks.kafka_sink.kafka.bootstrap.servers
server1,server2,server3
flume_agent.sinks.kafka_sink.kafka.topic = topic_name
flume_agent.sinks.kafka_sink.kafka.producer.security.protocol =
SASL_SSL
flume_agent.sinks.kafka_sink.kafka.producer.sasl.mechanism =
GSSAPI
flume_agent.sinks.kafka_sink.kafka.producer.sasl.kerberos.service.na
me = kafka
flume_agent.sinks.kafka_sink.kafka.producer.ssl.truststore.location =
/path//truststore.jks
flume_agent.sinks.kafka_sink.kafka.producer.ssl.truststore.password =
XXXXXXXX
```

Source Agent:

```
#Agent Configuration
flume_agent.channels = memory_channel
flume_agent.sources = kafka_source
flume_agent.sinks = null_sink

#Channel
flume_agent.channels = memory_channel
flume_agent.channels.memory_channel.type = memory
flume_agent.channels.memory_channel.capacity = 1000
flume_agent.channels.memory_channel.transactionCapacity = 1000

#Flume Source
flume_agent.sinks.null_sink.type = null
flume_agent.sinks.null_sink.channel = memory_channel

# Flume Source
flume_agent.sources.kafka_source.type =
org.apache.flume.source.kafka.KafkaSource
flume_agent.sources.kafka_source.channels = memory_channel
flume_agent.sources.kafka_source.kafka.bootstrap.servers
=server1,server2,server3
flume_agent.sources.kafka_source.kafka.topics = topic_name
flume_agent.sources.kafka_source.kafka.consumer.security.protocol =
SASL_SSL
flume_agent.sources.kafka_source.kafka.consumer.sasl.mechanism =
GSSAPI
flume_agent.sources.kafka_source.kafka.consumer.sasl.kerberos.servi
ce.name = kafka
flume_agent.sources.kafka_source.kafka.consumer.ssl.truststore.locati
on = /path/t.jks
flume_agent.sources.kafka_source.kafka.consumer.ssl.truststore.pass
word = XXXXXXXXXX
```

	Authentication	**Works with Graphite format(influxDB format)**
Flume	Kerberos	NO
Telegraf	SSL Certificates	YES

The table describes the comparison between the two technologies used. I decided to continue using Telegraf because it worked with Graphite format, the one is the format that ***InfluxDB*** works with. Was interesting to learn how to configure each technologies.

PHASE #2 AND RESULTS

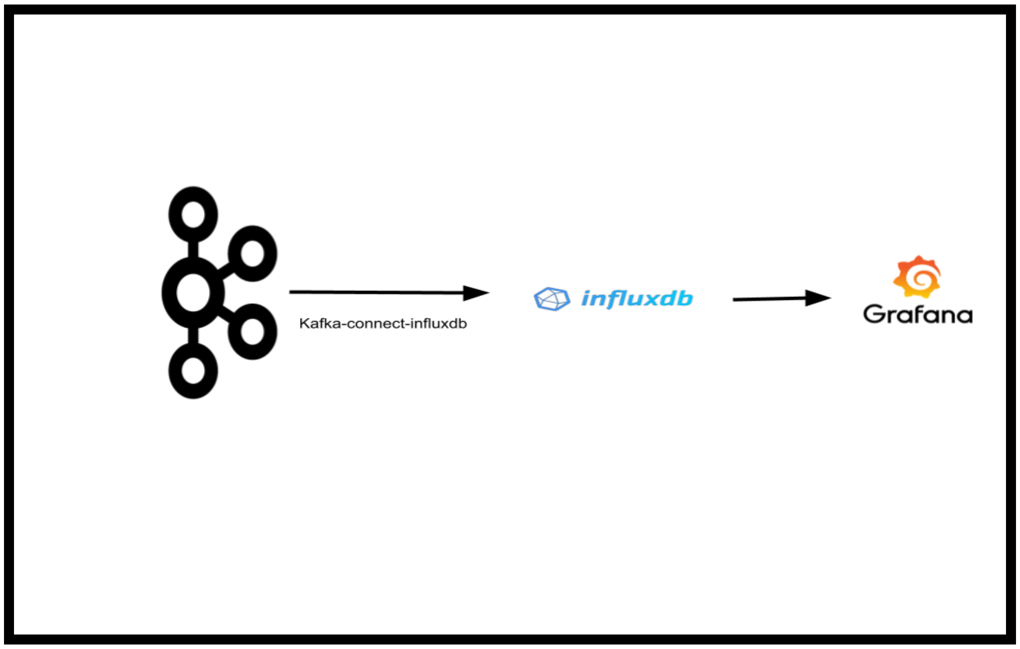


Figure 6: Second phase of the architecture

In this part of the project, the challenge consists in connect Kafka with influxDB using Kafka-connect-InfluxDB, which is a connector.

I followed this steps:

1. Downloaded https://github.com/Landoop/stream-reactor/releases/download/1.1.0/kafka-connect-influxdb-1.1.0-1.1.0-all.tar.gz
2. Create the database in InfluxDB: `CREATE DATABASE mydb`
3. Configure the connector with kerberos:

PHASE #2 AND RESULTS

influx-sink properties

```
# connector name
name=influxdb-sink
# connector class
connector.class=com.datamountaineer.streamreactor.connect.influx.InfluxSink
Connector
# maximum number of Kafka Connect tasks
tasks.max=1
# Kafka topic to read from (example: influx-topic)
topics=topic_name
# KCQL query - should include InfluxDB measurement name fo write to and
Kafka topic to read from
connect.influx.kcql=INSERT INTO cpu SELECT * cert-test WITHTIMESTAMP
sys_time()
# InfluxDB instance parameters
connect.influx.url=https://XXXX:XX
connect.influx.db=database_name
connect.influx.username=user
connect.influx.password=XXXXXXXXX
```

connect-standalone.properties

```
# These are defaults. This file just demonstrates how to override some
settings.
```

```
bootstrap.servers=server1:port,server2:port,server3:port


# if you run more than one standalone connector on the same host, each of
them must have a unique rest.port (default: 8083)

# in that case you would also need to prepare two separate
connect-standalone.properties files

rest.port = XXX


# set key.converter and value.converter

key.converter=org.apache.kafka.connect.json.JsonConverter

value.converter=org.apache.kafka.connect.json.JsonConverter


# if you want to send JSON payload only (without a schema) set the variables
below to false

key.converter.schemas.enable=false

value.converter.schemas.enable=false

# if your cluster is using SSL security, set values as below (use consumer.
prefix as it is a sink connector):

consumer.sasl.kerberos.service.name = kafka

consumer.sasl.mechanism = GSSAPI

consumer.security.protocol = SASL_SSL

plugin.path=path
```

But we encountered an error with kerberos. This error was cause for the way that kerberos authentication works with the key in the cluster. This error is reported for find a future solution. When this problem was founded, an alternaty was using the connector with SSL certificates with the following configuration file.

connect-standalone.properties

```
        bootstrap.servers=server1:port,server2:port,server3:port
```

```
# if you run more than one standalone connector on the same host, each of them must have a unique rest.port (default: 8083)
# in that case you would also need to prepare two separate connect-standalone.properties files
rest.port = XXX
# set key.converter and value.converter
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# if you want to send JSON payload only (without a schema) set the variables below to false
key.converter.schemas.enable=false
value.converter.schemas.enable=false


# if your cluster is using SSL security, set values as below (use consumer. prefix as it is a sink connector):
sasl.kerberos.service.name = kafka
sasl.mechanism = GSSAPI
security.protocol = SASL_SSL
sasl.jaas.config = com.sun.security.auth.module.Krb5LoginModule required \
useKeyTab=true \
storeKey=true \
keyTab="/etc/krb5.keytab" \
principal="principal";
consumer.sasl.kerberos.service.name = kafka
consumer.sasl.mechanism = GSSAPI
consumer.security.protocol = SASL_SSL
```

```
        consumer.sasl.jaas.config =
    com.sun.security.auth.module.Krb5LoginModule required \
        useKeyTab=true \
        storeKey=true \
        keyTab="/etc/krb5.keytab" \
        principal="principal";
        plugin.path=path..
```

But we encountered another problem. This problem was reported for a future solution.

IMPACT & FUTURE WORK

Impact

The data coming from sensors will be in the future data for monitoring radioactivity level of containers from other Iot system at CERN. This case of use is really interesting, because CERN The radioactivity level of more than 100 metallic containers for ordinary waste is routinely monitored.In a technical way, the user of Kafka 2.11-1.1.1 version, give a lot of problems in first time, and one of this problems with ACLS permission. While doing the configuration a ACLS error was encountered. The DB-IT group started working with Kafka 2.11-1.1.1 and with the before kafka version too. During the use of the new Kafka version, the problems with ACLS was useful to check what was happening.

For the future work:

1. The first task for the future work consists in encountered a solution for the kafka-connect-influxDB with the Kafka 2.11-1.1.1 version, so the data flow in this case will be finished.
2. Stop using telegraf for generating random data and replace for the sensors data using MQTT source and MQTT-kafka-connector.
3. Integrating kafka with Spark.

CONCLUSION

When we talk about Big Data and its impact, we also talk about the use of technologies that flood the market today. Each technology has different characteristics and different configurations that are interesting to assess to make a final decision on a architecture's solution.The development of this project consisted in the Development of streaming pipelines and analytics solutions for CERN's IoT Platform.

The draft solution consisted of sending sensor data using MQTT source, then sending data to Kafka, store in influxDB and then being monitored by Grafana. The realization of this solution was divided into two phases. In the first phase, data from sensors was not used, but two tools were tested: Telegraf and Flume. The main difference in its configuration is the way in which it authenticates and that Telegraf for example already accepts the Graphite data format, which is the one used by InfluxDB.

The second phase of the project consisted, once the data was in kafka, was sent to influxDB for monitoring in Grafana. For sending the data, we used the kafka-connect-nfluxdb connector.During the accomplishment of this task, some errors arose. First the connector was configured using Kerberos and then with SSL certificates, but in both cases errors were found. Having tested each of these tools was very useful to learn its operation and make better decisions for the final solution.Even mistakes found during the learning were useful for the team to solve other conflicts. For the future it would be necessary to solve the pending errors, use the sensors instead of Telegraf and Flume, and make an integration with Kafka and Spark.

REFERENCES

[1] Power Data. https://www.powerdata.es/big-data.

[2] Apache kafka. https://kafka.apache.org/intro.

[3] Telegraf. https://www.influxdata.com/time-series-platform/telegraf/.

[4] Telegraf. input data formats
https://docs.influxdata.com/telegraf/v1.7/concepts/data_formats_input/.

[5] InfluxDB. https://en.wikipedia.org/wiki/InfluxDB.

[6] Apache Flume. https://flume.apache.org/.

[7] Josh Reichardt(04-06-2014) Introduction to Grafana
https://thepracticalsysadmin.com/introduction-to-grafana/.

[8] internet of Things. https://en.wikipedia.org/wiki/Internet_of_things.

[9] Lovisa Johansson(2016-12-13). Part 1: Apache Kafka for beginners - What is Apache Kafka? .https://www.cloudkarafka.com/.

Develop streaming pipelines and analytics solutions for CERN's IoT Platform