



Tom 35/2022, ss. 115-135
ISSN 2719-4175
e-ISSN 2719-5368
DOI: 10.19251/ne/2022.35(7)
www.ne.mazowiecka.edu.pl

Katarzyna Racka

e-mail: k.racka@mazowiecka.edu.pl

Mazowiecka Uczelnia Publiczna w Płocku

ORCID ID: <https://orcid.org/0000-0002-9589-3360>

APACHE NIFI AS A TOOL FOR STREAM PROCESSING OF MEASUREMENT DATA

Summary

In order to analyze data in real time, without wasting time on preliminary aggregation of this data, organizations are increasingly redesigning the way they make decisions by implementing streaming analysis. Such analysis ensures ongoing data monitoring in order to evaluate them and detect possible irregularities. Thanks to this approach, we are able to increase the speed and accuracy of decision-making, which is important wherever we want to quickly respond to data anomalies, e.g. by indicating failures or threats.

In order to present the solutions in a better way, the article describes a project whose task

was to design and implement a system using the Apache NiFi program to stream process air quality measurement data from the API of the Chief Inspectorate for Environmental Protection.

The project allowed to automate the flow of measurement data between the systems. Data flow presentation was presented using a visual interface. The NiFi program enabled early filtering of data by dividing them according to the scale of measurement thresholds, which enabled their monitoring and evaluation, taking into account possible data gaps. As the measurement data was processed in a streaming manner, the

project was able to register the missing measurements, which would be impossible in another solution using batch processing, in which measurements are collected in databases and then analyzed on static data, because such information would be supplemented later time which would lead to overwriting of this data.

Keywords: Apache NiFi, Streaming data, Apache Kafka, Apache ZooKeeper, Apache Spark Structured Streaming.

JEL Classification: O3

INTRODUCTION

Each organization generates and collects a lot of different data. According to the slogan “Business is data and data is business”, the collected data can be helpful in the implementation of business strategies. The improvement of business strategies, based on the analysis of the collected data, is now a necessity in order to catch up with the competition.

Big Data technologies and their application to business processes is growing dynamic. Many companies believe that unstructured data analysis will be the key to a deeper understanding of customer behavior and risk assessment. They believe that analytics is absolutely essential or very important to running the company’s overall business strategy and improving operational performance.

We are on the threshold of a very fast pace of technological change. Not only the Internet and mobile communication have changed the world. Accelerating technologies such as artificial intelligence and big data began another big change.

The problem of collecting large amounts of data, forces organizations to pay attention to the cost of mass memory, which causes, among other things, interest in streaming data processing. In addition, given the way COVID-19 has changed the business landscape, the historical data that companies have so readily collected may sometimes no longer be relevant and the models used – no longer apply.

Transitioning from big data to small and wide data is one of the Gartner top data and analytics trends for 2021. These trends represent business, market and technology dynamics that data and analytics leaders cannot afford to ignore¹.

To take advantage of the increasing availability of real-time streaming data without wasting time pre-aggregating that data, organizations are redesigning

¹ <https://www.gartner.com/smarterwithgartner/gartner-top-10-data-and-analytics-trends-for-2021>

their way of making decisions by implementing streaming analytics and streaming data integration.

By using event stream processing platforms, we are able to increase the speed and accuracy of making decisions, which is important wherever we want to quickly respond to data anomalies, for example indicating failures or threats. By resigning from batch processing, which is usually performed on static data from a specific time period and collected in sets containing huge amounts of data, we avoid the problem of a large loss of time needed to process this data.

Streaming data often appears in the Big Data context, where data is generated by many different sources at high speed.

The aim of the article is to present an alternative to traditional batch data processing by presenting a solution such as data streaming. The article presents Apache NiFi as a data stream processing tool.

1. APACHE NIFI

Apache NiFi is a software project from the Apache Software Foundation. Apache NiFi is an efficient data processing and distribution system. Apache NiFi is a reliable and secure data transfer between systems because it is scalable, i.e. you can use many NiFi servers (server cluster). It allows for data buffering between processors using mini-queues called connectors. It also has a back pressure mechanism that controls the amount of data in the buffer.

Apache NiFi allows you to create and maintain directed graphs that allow you to design information flow.

Apache NiFi provides:

- graphic web-based user interface,
- managing and modifying data flow,
- data flow monitoring,
- feedback management,
- low latency,
- high throughput,
- security through data encryption (SSL, SSH, HTTPS, etc.)
- creating your own processors.
- Data enrichment and preparation allows you to:
 - conversion between formats,
 - extraction (extracting data from internet resources),
 - parsing,

- flow control.

The basic processing unit in Apache NiFi is FlowFile, which contains data content and attributes (metadata). FlowFile attributes are used by NiFi processors for data processing. FlowFile attributes consist of eg UUID - identifier, FlowFile name, FlowFile size, type, path.

A FlowFile represents each object moving through the system and for each one, NiFi keeps track of a map of key/value pair attribute strings and its associated content of zero or more bytes.

The FlowFile object differs from a regular file in that FlowFile attributes can be modified and new attributes can be created based on the content.

The second part of FlowFile is the content of the FlowFile, which usually contains data taken from source systems.

What distinguishes NiFi is its graphical web-based user interface, in which you can design data flow graphs using ready-made components in the form of processes.

The processor performs the designated transformations of the data that flow through the processor. Each processor can process any FlowFile and the processors pass references to the FlowFile so that the data is not copied to individual processors. Moreover, each processor runs in a separate thread (in parallel), so it is a multi-threaded, distributed system.

Processors are used for:

- adding, modifying, removing attributes,
- content modification (eg by conversion, extraction).

The connection between the processors is the connector, which is the queue that can buffer the FlowFile between the processors. Within the queue, you can prioritize the attributes (which attribute is more important) to determine the order in which the attribute will be handled. You can also set upper bounds on load which enable back pressure.

2. STREAMING DATA

Streaming data is data that is generated continuously. Such data should be processed incrementally using stream processing techniques without reverting to historical data. Stream data analysis can be performed in real time without the need to store the data in advance, which is distinguished by the speed and continuous nature of such analyzes.

The source of the streaming data can be:

- computer systems,

- database transactions,
- signals from devices (e.g. GPS, IoT, mobile devices),
- messages from websites (e.g. tracking user activity),
- social media,
- sensors of measuring devices,
- machine sensors.

In summary, any system that sends data can generate streaming data. The key aspect is whether we want and are able to process this data as streaming data.

Streaming processing enables data processing in real time or in micro-batch. Thanks to this, the time from the appearance of the record to its processing is counted in seconds. After such pre-processing, such data can of course be stored both in a relational database and in NoSQL databases or in Hadoop.

Acquisition, transmission, storage and analysis of measurement data using Big Data tools ensures detailed analysis of measurement data in real time in a durable and fail-safe manner. Such analysis ensures ongoing data monitoring in order to evaluate them and detect possible irregularities.

The next part of the article presents an example of the implementation of a Big Data system with the use of Apache NiFi for stream processing of measurement data on air quality from the API of the Chief Inspectorate for Environmental Protection.

3. AN EXAMPLE OF STREAM DATA PROCESSING USING APACHE NIFI

The aim of the project was to design and implement a Big Data system for data processing enabling automatic collection, processing and evaluation of air quality measurement data streams.

Project assumptions:

- automation of the flow of measurement data related to air quality between systems,
- presentation of data flow in the form of a visual interface,
- the possibility of early filtering of data by dividing the data according to the scale of measurement thresholds in order to monitor and evaluate them, taking into account possible data gaps.

In the project presented the automation of air quality data flow. These data come from the API of the Chief Inspectorate for Environmental Protection

available on powietrze.gios.gov.pl. The data is transferred in the current time. The API interface of the “Air Quality” portal of the Chief Inspectorate of Environmental Protection allows access to data on air quality in Poland, produced under the State Environmental Monitoring and collected in the JPOAT2.0 database. According to the information from the Chief Inspectorate of Environmental Protection:

The data provided is not verified on an ongoing basis, so it may be changed at a later time. (...) due to occurrences of measuring equipment failures, data transmission failures or other random events, there may be occasional interruptions in the transmission and presentation of measurement data on the “Air Quality” portal and in applications using the API interface, which was observed in during the implementation of this project and taken into account in the project objectives.



JSON	Nieprzetworzone dane	Nagłówki
Zapisz	Kopuj	Zwiń wszystkie Rozwiń wszystkie
Filtruj JSON		
key:	"NO2"	
▼ values:		
▼ 0:	date: "2021-06-09 15:00:00"	value: 42.7024
▼ 1:	date: "2021-06-09 14:00:00"	value: 49.2977
▼ 2:	date: "2021-06-09 13:00:00"	value: 36.2779
▼ 3:	date: "2021-06-09 12:00:00"	value: null
▼ 4:	date: "2021-06-09 11:00:00"	value: 44.9352
▼ 5:	date: "2021-06-09 10:00:00"	value: 45.7441
▼ 6:	date: "2021-06-09 09:00:00"	value: 53.706
▼ 7:	date: "2021-06-09 08:00:00"	value: 57.9809
▼ 8:	date: "2021-06-09 07:00:00"	value: 68.1782
▼ 9:	date: "2021-06-09 06:00:00"	value: 52.0999
▼ 10:	date: "2021-06-09 05:00:00"	value: 46.8336

Figure 1 An example of the result of the network service of the Chief Inspectorate for Environmental Protection providing measurement data on the basis of the NO2 measurement station ID provided.

Source: Own elaboration.

Due to the diversity of measurement indicators and the averaging time of measurements at measurement stations, the research in the project was carried out on a randomly selected indicator number 740, which returns the measurements of nitrogen dioxide. The measurements are made available in JSON format, which consists of the measurement time and the measurement value.

Nitrogen dioxide is one of the most dangerous compounds polluting the atmosphere. One of the main sources of their emissions is road transport.

The air quality index is presented on the website of the Chief Inspectorate for Environmental Protection.

Table 1. Own elaboration based on the Chief Inspectorate for Environmental Protection air quality index.

Air quality index	NO2 [$\mu\text{g}/\text{m}^3$]
Very good	0-40
Good	40,1-100
Moderate	100,1-200
Sufficient	200,1-350
Bad	350,1-500
Very bad	>500
Index not available	

Own elaboration. Source: Chief Inspectorate of Environmental Protection

Taking into account the above index, a seven-level scale of measurement thresholds has been adopted in the project.

In addition, the website of the Chief Inspectorate for Environmental Protection also presents health information on air quality explaining the individual categories presented in the index.

Table 2. Health information on air quality measurements prepared by the Chief Inspectorate of Environmental Protection.

Air quality index	Description
Very good	The air quality is very good, air pollution does not pose a health risk, the conditions are very favorable for any outdoor activities, without restrictions.
Good	Air quality is satisfactory, with air pollution causing little or no health risk. You can stay in the open air and perform any activity, without restrictions.
Moderate	The air quality is acceptable. Air pollution can pose a health risk in special cases (for the sick, the elderly, pregnant women and young children). Moderate conditions to outdoor activities.

Sufficient	Air quality is sufficient, air pollution is a health risk (especially for the sick, the elderly, pregnant women and young children) and can have negative health effects. Reduction (shortening or staggering) of outdoor activities should be considered, especially if the activity involves prolonged or increased physical exertion.
Bad	The air quality is bad and the sick, the elderly, pregnant women and young children should avoid being outdoors. The rest of the population should minimize any physical activity in the open air - especially those requiring prolonged or increased physical effort.
Very bad	The air quality is very bad and has a negative impact on health. Sick people, the elderly, pregnant women and young children should absolutely avoid being outdoors. Remaining population should limit outdoor activities to the necessary minimum. Any physical activity outside is discouraged. Long-term exposure to airborne substances increases the risk of changes, e.g. in the respiratory, cardiovascular and immune systems.
Index not available	„Index not available” corresponds to a situation where no particulate matter or ozone measurements are currently carried out at a given measuring station, and one of them is at the moment the decisive air pollution in the voivodeship. The Air Quality Index is then not determined and the color of the points on the map of the current measurement data changes to gray. The station, despite the lack of a specific Index, is still visible and it is possible to check all other measurement results.

Source: Own elaboration. Chief Inspectorate of Environmental Protection

4. TECHNOLOGIES USED IN THE PROJECT

The data from the air quality measurement sensors presented in the project is continuously generated in an incremental manner. As, according to the assumptions of the project, they are to be processed on an ongoing basis, therefore they are treated as streaming data.

In order to solve the problem of scalability resulting from the constantly growing amounts of data and to quickly process this data, the following Big Data technologies were used in the project.

DOCKER COMPOSE

In order to ensure system isolation, including independence from the operating system and separation of processes, the project was based on a solution using containers.

Initially, Docker Desktop was installed on the local machine. Docker is an open platform for developing, shipping, and running applications².

Docker allows you to run a single application in a container separate from the operating system. In addition, unlike virtual machines, it is characterized

² Docker Documentation, <https://docs.docker.com/get-docker/>

by low resource consumption because it does not require additional operating systems for individual applications.

Docker Compose was used in the project to automate the creation and running of multiple dependent containers.

Docker Compose is a tool for defining and running multi-container Docker applications. Docker Compose uses the YAML file to configure the services of the application that will be able to use:

- Apache ZooKeeper
- Apache Kafka
- Apache NiFi,
- Jupyter Notebook Python,
- Apache Spark (pyspark-notebook)

As part of the YAML file, among others:

- determined which containers and in what order are to be launched from which images,
- the ports and names that can be used to connect to the containers have been mapped.

APACHE KAFKA

Apache Kafka was used as a protection against data loss in the event of a possible failure in the project, which acts as a record stream buffer, i.e. protects against engine flooding caused by data overload. Kafka also allows you to read the data that was sent to it and was not read by the target system, after resuming its operation after a possible failure (the retention time can be determined, but it does not have an infinite value).

Kafka publishes and subscribes records similar to message queues, with the difference, however, that Kafka replaces traditional message queues. In a classic queue, if the first consumer downloads messages from the queue, this element is no longer available to other consumers, which means that one message could reach only one recipient. Kafka allows you to transfer message streams between applications in distributed systems. From Kafka, many reading applications can download the same data. Kafka also serves as integration between multiple systems.

Kafka is run as a cluster of one or more servers. If you put Kafka in the cluster on one machine, there is a high risk that if the machine fails, you will lose the entire Kafka cluster. Therefore, Kafka should be installed on many machines to protect against data loss and this is what this project has designed.

The events Kafka keeps are stored in topics, which are message queues. Events that are written for topics are of the form key value with an associated timestamp. The topic ID is its name.

The project developed 7 themes according to a seven-point scale of thresholds for air quality measurements.

Topics are partitioned. If the topic has at least two partitions, it means that part of the message goes to one part of the topic - the partition, and the rest goes to the other partition. Each of these partitions has its own offset, which informs how many events in this partition are stored. In addition, offset allows you to arrange events because each event in the topic is assigned an increasing offset. The events in the partitions are in order. Additionally, it is worth remembering that each partition has its own offset and it has no relation to the offsets of other partitions, which means that the order of events is only guaranteed inside the partition. Data written to the partition cannot be changed, ie an events written to the event partition cannot be changed.

A Kafka cluster consists of many servers called Brokers. Each broker has its own ID and each broker supports a set of topic partitions.

When connecting to Kafka, you can specify the address of one or all brokers, although it is better to specify the addresses of all brokers, because if one of the brokers is not working then the publisher or subscriber can connect to another broker on the list.

The topic can also be replicated. The topic should have a replication rate greater than 1, usually set to 2 or 3. A replication factor of 3 means that each partition has two backups written to the other brokers. Only one broker at a time can be the leader of a given partition on a topic to write to and read from.

In order to determine the reliability of the system in the project, the number of partitions is 3 and the number of replications is 3.

Producer writes events for a given topic. The producer knows which partition to write to, if there are more. In the event of a broker failure, the producer automatically switches to a new leader. The producer may attach a key to the event. Keyless events will be spread across all partitions. However, events with the same key will go to the same partition.

Moreover, in distributed systems, such prepared events with a key, sent from Kafka to Spark, facilitate the operations of grouping events and make it unnecessary to exchange data from a given group for a given process. Each of Spark's processes reads from a specific broker, in which the data has already

been sorted by key (within the same partition, an event is sent to a specific process in Spark). This issue was also included in the projects and the events were enriched with keys before being sent to Spark.

From the topic, the data is read by the consumer who knows from which broker to read it. In the event of a broker failure, the consumer will automatically switch to a new leader. Data is read in the same order as it was written within the partition (based on the event offset in the partition).

The Apache Kafka consumer group is characterized by consumers reading the data as a group. Each partition is assigned to only one consumer within the group. An application written in SparkStreaming connects to Kafka in consumer group mode. Each Kafka reading process has a specific partition assigned to it. Consumer groups cause that a given process to be assigned one or more partitions. This way, the given Spark process will have records with the same key as it is reading from partition data. If the number of consumers is greater than the number of partitions, some of the consumers will be inactive, which should be remembered for efficient processing. Moreover, if the application is connected as a consumer group (by specifying the name of the group), Kafka will remember up to which point we have read events from a given partition (based on the partition's offset). Thanks to this, in the event of a failure, we are able to read unread data from Kafka.

APACHE ZOOKEEPER

A required component of any Apache Kafka cluster is the Zookeeper. ZooKeeper is a high-performance distributed application management service³.

Apache Zookeeper is used to manage the cluster. Zookeeper allows you to:

- naming service,
- configuration management,
- synchronization,
- group services,
- choosing a leader,
- management of the entire cluster.

Zookeeper makes it easy to add new servers. Zookeeper will automatically include the server in the cluster and ensure that the configuration is the same on all servers.

³ *Apache ZooKeeper*, <https://zookeeper.apache.org/doc/r3.7.0/index.html>

Zookeeper allows Kafka to go into high availability mode. At the start of Kafka, we give the addresses of the zookeepers. Kafka uses Apache Zookeeper to manage the cluster metadata. Zookeeper stores information about brokers, who are the leader, on which brokers are partitions.

To ensure high availability and fault tolerance in the project, when using Docker Compose, the following were called: zookeeper1, zookeeper2, zookeeper3.

APACHE NIFI

In the project prepared a flow made in Apache NiFi with the use of the following processors.

GetHTTP - This operator gets data from an HTTP or HTTPS URL and writes the data to the FlowFile content.⁴ The processor settings include, among others frequency of loading new data and API address from which measurement data related to air quality are collected. Data from the API is data in JSON format. The processor forwards this downloaded data to the next processors.

The second flow processor is SplitJson, which splits the JSON file into multiple separate FlowFiles for the array element specified by the JsonPath Expression.

The third processor used in the project is EvaluateJsonPath. This processor evaluates one or more JsonPath expressions against the contents of the FlowFile. The results of these expressions are either assigned to the FlowFile attributes or saved to the contents of the FlowFile itself, depending on the CPU configuration.


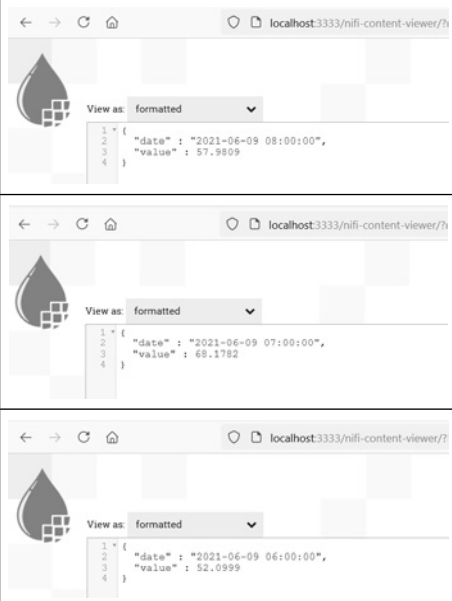
In the project, the EvaluateJsonPath processor creates the date and value attributes from the contents of the FlowFile. The project parses the contents of the JSON object.

UpdateAttribute was used as the fourth processor in the project. It updates the attributes for the FlowFile using the attribute expression language, and it can remove attributes based on a regular expression. A new attribute named key is created in the project, the value of which is taken from the attribute date.

The fifth processor in the project is RouteOnAttribute, which redirects the FlowFile to the output based on attribute values using an attribute expression language.

⁴ Apache NiFi Documentation, <https://nifi.apache.org/docs.html>

Table 3. Separation of an array of JSON objects into individual objects and each placed in a separate FlowFile

Data collected from 1 measurement in JSON format	JSON file split into multiple separate FlowFiles
	

Source: Own elaboration.

In the project in the RouteOnAttribute processor, based on the value of the value attribute, according to the Air Quality Index of the Chief Inspectorate of Environmental Protection for nitrogen dioxide, FlowFile redirects are performed according to the following seven-step scale:

1. value between 0 and 40 will be redirected to the bdb output,
2. value between 40.1 and 100 will be redirected to db output,
3. value between 100.1 and 150 will be redirected to um output,
4. value between 150.1 and 200 will be redirect to the dst output,
5. value between 200.1 and 400 will redirect to the zle output,
6. value greater than 400 will redirect to bzle output,
7. if there is no value in the given measurement - no index will be redirected to the bind output.

In addition, for FlowFiles that do not match any user-defined expression, the unmatched option was used, which automatically terminates the relationship

The following Apache NiFi functions were used to define the redirection of FlowFile to the output based on the value of the value attribute in the project:

- `gt`, which is used for numeric comparison and returns true if the subject is greater than its argument.
- `ge`, which is used for numeric comparison and returns true if the subject is greater than or equal to its argument. If either the subject or the argument cannot be converted into a number, this function returns false.
- `lt`, which is used for numeric comparison and returns true if the subject is less than its argument. If either the subject or the argument cannot be converted into a number, this function returns false.
- `and` - this function takes a boolean value as a single argument and returns true if both the subject and the argument are true. If either the subject or the argument is false or cannot be converted into Boolean, the function returns false. Typically this is used with an embedded expression as the argument.
- `isEmpty` - this function returns true if the subject is null, contains no characters or contains only white-space (new line, carriage return, space, tab), otherwise false.

The last processors used in the project are `PublishKafka_2_6`. They send the FlowFile content as a message to Apache Kafka using the Kafka 2.5 Producer API. The messages to send may be individual FlowFiles or they may be delimited, using a user-specified delimiter such as a new-line.

In the project, Kafka Brokers and Kafka key were defined in the `PublishKafka_2_6` processors.

The figure 2 shows the flow of measurement data related to air quality between the systems prepared in the project. This figure presents the processes discussed earlier and connectors that act as connectors between processors.

The process begins with reading the data from the API of the Chief Inspectorate of Environmental Protection, the data is downloaded in JSON format. Then an array of JSON objects is split into individual objects, i.e. separate measurements, and then each is placed in a separate FlowFile. In the next step, the contents of the JSON object are parsed for the presence of the key. Attributes are created and then FlowFile is redirected to the output to Apache Kafka, based on the value of the value attribute, according to the Air Quality Index of the Chief Inspectorate of Environmental Protection for nitrogen dioxide (Table 1).

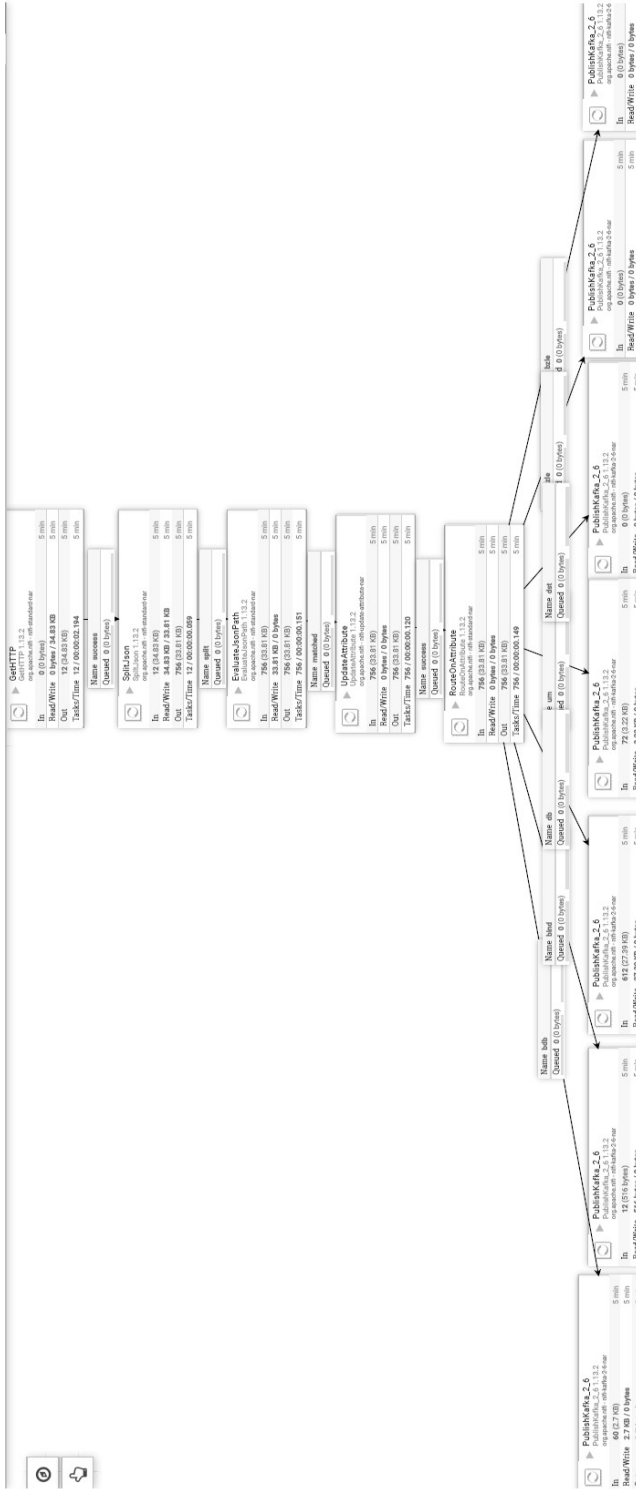


Figure 2 NiFi flow of the project taking into account the seven-step measurement scale

Source: Own elaboration.

Apache NiFi allows you to check what is happening with events, where there are so-called “bottlenecks” extending the time of the transformation process. You can use diagrams to check the event path, how long it lasted, and use graphs to check the amount of data transferred.

In line with the aim of the project, thanks to NiFi, data flow can be monitored.

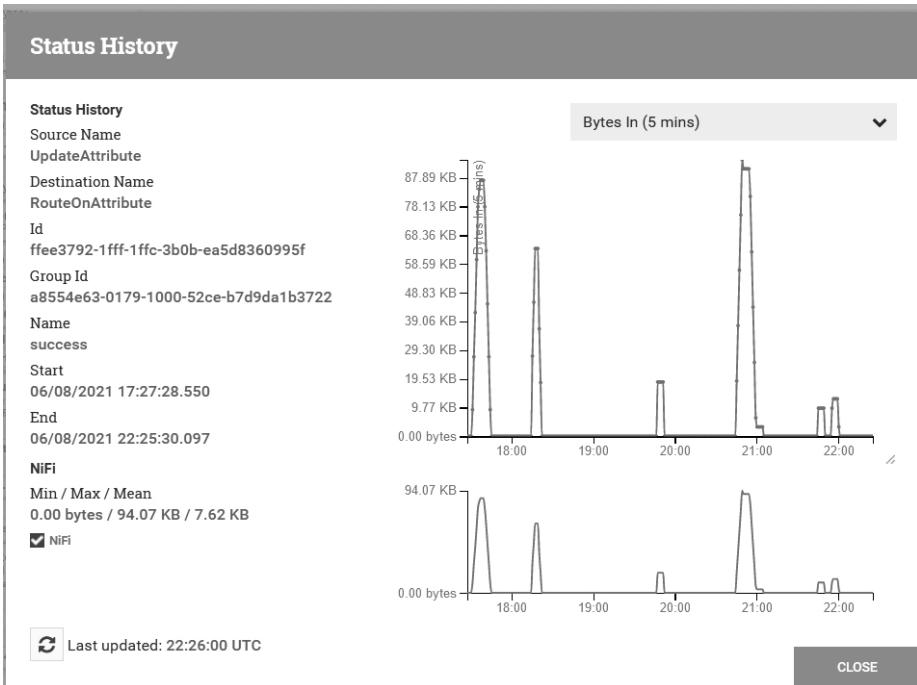


Figure 3. Status History related to the project

Source: Own elaboration.

NiFi Data Provenance - shows what operations were performed on a given FlowFile.

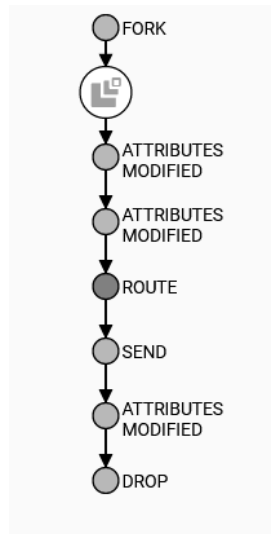


Figure 4. NiFi Data Provenance related to the project

Source: Own elaboration.

Apache NiFi processes each event in a separate thread. One event cannot be processed across multiple threads. Therefore, NiFi is not suitable for distributed computing, computing sequences, and combining and aggregating events.

Structured Streaming Apache Spark was proposed for the project for the next processing steps.

APACHE SPARK STRUCTURED STREAMING

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. You can express your streaming computation the same way you would express as batch computation on static data. The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as the streaming data continues to arrive.⁵

The result of using structured-streaming in the project is presented below. It allowed to download data from the indicated Kafka topic and then convert them from JSON to a table in the key-value format. This example includes also

⁵ *Structured Streaming Programming Guide*, <https://spark.apache.org/docs/2.2.2/structured-streaming-programming-guide.html>

the problem of late events. The result is presented in 2 hour windows, updated every second.

	key	value
950cb70fc953_dane_pyspark-notebook_1		
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 13:00:00	{"date": "2021-06-10 13:00:00", "value": 82.4885}
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 10:00:00	{"date": "2021-06-10 10:00:00", "value": 57.6994}
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 07:00:00	{"date": "2021-06-10 07:00:00", "value": 61.5255}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 18:00:00	{"date": "2021-06-09 18:00:00", "value": 65.8934}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 15:00:00	{"date": "2021-06-09 15:00:00", "value": 42.7024}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 10:00:00	{"date": "2021-06-09 10:00:00", "value": 45.7441}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 08:00:00	{"date": "2021-06-09 08:00:00", "value": 57.9809}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 06:00:00	{"date": "2021-06-09 06:00:00", "value": 52.0999}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 03:00:00	{"date": "2021-06-09 03:00:00", "value": 53.1005}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 02:00:00	{"date": "2021-06-09 02:00:00", "value": 85.4813}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 01:00:00	{"date": "2021-06-09 01:00:00", "value": 99.3374}
950cb70fc953_dane_pyspark-notebook_1	2021-06-08 21:00:00	{"date": "2021-06-08 21:00:00", "value": 75.6462}
950cb70fc953_dane_pyspark-notebook_1	2021-06-08 16:00:00	{"date": "2021-06-08 16:00:00", "value": 61.2313}
950cb70fc953_dane_pyspark-notebook_1	2021-06-08 15:00:00	{"date": "2021-06-08 15:00:00", "value": 56.0413}
950cb70fc953_dane_pyspark-notebook_1	2021-06-08 14:00:00	{"date": "2021-06-08 14:00:00", "value": 64.7085}
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 09:00:00	{"date": "2021-06-10 09:00:00", "value": 56.8278}
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 06:00:00	{"date": "2021-06-10 06:00:00", "value": 62.9943}
950cb70fc953_dane_pyspark-notebook_1	2021-06-10 03:00:00	{"date": "2021-06-10 03:00:00", "value": 42.0316}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 21:00:00	{"date": "2021-06-09 21:00:00", "value": 71.0457}
950cb70fc953_dane_pyspark-notebook_1	2021-06-09 19:00:00	{"date": "2021-06-09 19:00:00", "value": 49.2021}
950cb70fc953_dane_pyspark-notebook_1		

Figure 5. The result of converting both the key and the value to a string type

Source: Own elaboration.

In order to run the query for several days, it is necessary for the system to associate the amount of accumulated intermediate state in memory. This means that the system needs to know when an old aggregate can be wiped out of memory as the application will no longer receive lagged data for that aggregate. To make this possible, Spark 2.1 introduces a watermark that allows the engine to automatically track the current event time in the data and erase the old condition accordingly. You can define a question watermark by specifying the event time column and the expected data delay threshold in terms of the event time. For a specific window starting at time T, the motor will hold the state and allow the late data to update the state (maximum event time as seen by the motor - delay threshold > T). In other words, data delayed within the threshold will be aggregated, but data beyond the threshold will be discarded.

	window	count
950cb70fc953_dane_pyspark-notebook_1		
950cb70fc953_dane_pyspark-notebook_1	[2021-06-08 00:00:00, 2021-06-08 00:00:05]	4
950cb70fc953_dane_pyspark-notebook_1	[2021-06-09 00:00:00, 2021-06-09 00:00:05]	8
950cb70fc953_dane_pyspark-notebook_1		

Figure 6. Result

Source: Own elaboration.

MONGODB DATABASE

In order to save the results, the project was expanded with an additional MongoDB cluster.

MongoDB is a NoSQL database whose basic data storage unit is a document that can contain key - value records. Documents are handled in Json / BSON format. They are stored in collections. Mongo has extensive scaling and great indexing capabilities. It allows you to index nested documents and arrays.

Apache Nifi was used to transfer data from Apache Kafka to the MongoDB database with the following processors.

ConsumeKafka_2_6, which consumes messages from Apache Kafka specifically built against the Kafka 2.6 Consumer API. In this process defined, inter alia, Kafka Broker data.

Another processor used in the data flow was the PutMongo processor, which writes the content of a FlowFile to MongoDB.

As a result, the following flow was obtained.

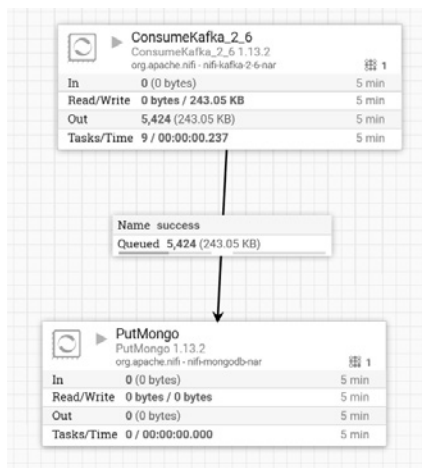


Figure 7. Data flow from Kafka to MongoDB made in the NiFi program

Source: Own elaboration.

DESCRIPTION OF THE RESULTS

The assumption of the project was to automate the flow of measurement data between systems, present the data flow in the form of a visual interface and the possibility of early filtering of data by dividing the data according to the scale of measurement thresholds in order to monitor and evaluate them, taking into account possible data gaps. Moreover, taking into account the fact

that the data sent from the measuring sensors are constantly generated in an incremental manner, an additional assumption of the project was to ensure the scalability and reliability of the system.

Thanks to the use of tools such as Docker and Docker Compose, Apache Kafka, Apache ZooKeeper, Apache NiFi, Apache Spark Structured Streaming, it was possible to implement the assumptions adopted at the beginning.

When analyzing the results obtained from the above project, it was noticed that most air quality measurements in the analyzed period were characterized by good, very good and moderate results.

As the measurement data was processed in a streaming manner, the project was able to register missing measurements that resulted from transmission interruptions and measurement data presentation on the Air Quality portal and in applications using the API. According to the information provided by the Chief Inspector of Environmental Protection, the disclosed data is not verified on an ongoing basis, so they may be changed at a later time.⁶ Therefore, the detection of data gaps in another solution using batch processing and not stream processing, in which the measurements were collected in databases and then analyzed on static data, would sometimes be impossible, because such information was often supplemented at a later time, which would lead to overwriting of this data.

SUMMARY

Legal regulations and company policy force companies to constantly collect data, encounter problems resulting from hardware limitations and the time needed to analyze this data.

On the other hand, following the recent experiences of the COVID-19 pandemic, many business analysts have noticed that historical data collected before the pandemic period has become useless as the situation in many business sectors has changed radically.

The assumption of the article was to present trends in the development of IT technologies resulting from the collection of huge amounts of data and at the same time the need to instantly detect anomalies in this data. Streaming data processing allows for the analysis of this data in real time and ensures ongoing data monitoring in order to evaluate it and detect possible irregularities. The Big Data tools described in the article ensure the resistance of such systems to failures while collecting and processing huge amounts of data.

⁶ API portal "Air Quality", <https://powietrze.gios.gov.pl/pjp/content/api>

In order to present the solutions in a better way, the article describes a project whose task was to design and implement a system using the Apache NiFi program to stream process air quality measurement data from the API of the Chief Inspectorate for Environmental Protection.

References:

Chief Inspectorate of Environmental Protection, <https://powietrze.gios.gov.pl>

API portal "Air Quality", <https://powietrze.gios.gov.pl/pjp/content/api>

Documentation:

Apache Kafka documentation, <https://kafka.apache.org/documentation>

Apache NiFi Documentation, <https://nifi.apache.org/docs.html>

Apache ZooKeeper, <https://zookeeper.apache.org/doc/r3.7.0/index.html>

Docker Documentation, <https://docs.docker.com/get-docker/>

Structured Streaming Programming Guide, <https://spark.apache.org/docs/2.2.2/structured-streaming-programming-guide.html>