Design a Influx db with Apache Superset model for real time smart building operations monitoring

Janhavi Bawaskar *IIT2020263*
Pratyaksh Singh *IIB2020015*
Mohit Kumar *IIT2020220*
Harshkant Bhatnagar *IEC2020113*

***Abstract*—Intelligent monitoring combines real-time sensing, data processing, event detection, predictive analytics, and collaborative decision-making tools. We have We use data Stream Generator which takes data from the dataset and this is passed through the Kafka topic which is named as StreamToSpark. This acts as a messenger and broker between the stream and Spark. This data stream is then ingested to Spark from where it is used to train Machine Learning Models. The models use Gradient based learner and Tree based learner. Then the information which is generated from this is again ingested in Spark. After the data is passed through the models, it is passed from Spark to InfuxDB through a kafka topic which is named as SparkToInfux. This data is then ingested in InfuxDB. After this for visualization purposes we have used Grafana which takes the data from InfuxDB and visualizes dashboard on the screen based on the features we choose. This way we can visualize, query and add alerts. Alerts can be added and a threshold value can be set based on which alerts are generated. This is our entire workflow which we have worked on in our for BDA Project.**

I. Introduction

Real-time sensing, data processing, event detection, predictive analytics, and cooperative decision-making tools are all combined in intelligent monitoring. Interest in intelligent monitoring has grown as a result of the Internet of Things' (IoT) quick growth and advancements in artificial intelligence. Smart device and sensor network proliferation has led to big data characteristics in environmental monitoring. Predictive analytics and datastream processing best practises are required because the environmental monitoring systems in use today are not built for real-time datastreams. Environmental intelligent monitoring systems require complex event processing (CEP) engines in order to manage heterogeneous datastreams with scalable performance.The study introduces a CEP engine that uses monitoring data from geological carbon sequestration to identify anomalies in real time. There is not much coding needed for the CEP engine, and it can be expanded to accommodate more environmental monitoring uses. It facilitates the production of predictive analytics and actionable insights, supporting the processes of environmental management and resource allocation decision-making. In general, real-time datastream processing improves environmental monitoring systems' responsiveness, accuracy, and efficiency, which helps with sustainability and efficient environmental management.

II. Related Work

The paper we had chosen to take inspiration and to further improve was "Building complex event processing capability for intelligent environmental monitoring" by the authors Alexander Y. Suna, Zhi Zhonga, Hoonyoung Jeongb and Qian Yanga. This paper mainly focuses on Intelligent monitoring, Complex event processing, Machine learning and Anomaly detection.

Author name	Year	Paper Title	Application domain	Achieved Performance
Alejandro Buchmann, TU Darmstadt, Boris Koldehofe, Universität Stuttgart	2019	Complex Event Processing	IoT Smart Cities CEP	Studies focusing on performance metrics, scalability, and optimization techniques in CEP systems. Comparative analysis of various CEP systems
Alexander Y. Suna, Zhi Zhonga, Hoonyoung Jeongb, Qian Yanga	2019	Building complex event processing capability for intelligent environmental monitoring	Real-time Event Detection Complex event processing Predictive Analytics for Environmental Patterns	Complex event processing (CEP) engine for detecting anomalies in real time, and demonstrates it using a series of real monitoring data from the geological carbon sequestration domain.
Nithin Krishna Reghunathan	2020	Real-Time Streaming in Big Data: Kafka and Spark with SingleStore	Real-Time Streaming Big Data Technologies Stream Processing Pipelines Real-Time Analytics	Investigates the integration and capabilities of Kafka and Spark in conjunction with SingleStore to facilitate real-time streaming and processing within the realm of big data applications.
Haruna Isah,Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, Shahzad Khan	Jan 2021	A Survey of Distributed Data Stream Processing Frameworks	Distributed Data Stream Processing Framework Analysis and Evaluation	Synthesizing and presenting existing information regarding the strengths, weaknesses, and characteristics of different distributed data stream processing frameworks based on available literature and evaluations performed by others

III. Comparison with existing work or base paper

Originally designed as messaging queue middleware (software that acts as a bridge between applications or systems), Kafka has evolved into a high-performance, distributed streaming platform with three main functions: (a) publish and subscribe to real-time applications or topics, (b) store event streams or data records in a fault-tolerant manner, and (c) process data as it happens (Apache Kafka, 2018). Kafka is now one of the open-source community's most mature CEP engines. confluent-open-source) into a real-time streaming processing ecosystem that includes a variety of infrastructure services such as database connectors and configuration managers.

In this paper, the authors present a CEP engine for anomaly detection in geological carbon sequestration (GCS) projects. Integral monitoring is required to track the injected carbon dioxide (CO2) plume as it moves through a storage formation to ensure the safe and efficient operation of GCS repositories.

This work contributes significantly in two areas: (a) adapting a high-performance CEP engine for GCS monitoring, and (b)

developing a streamlined intelligent monitoring workflow that requires little coding effort from domain users. Our system's modules are loosely coupled, allowing it to be flexible and expandable, and its components can be swapped out when connecting to new monitoring devices or applications.

They have also created producer and consumer templates for our use case, making the workflow easily adaptable to different types of sensor datastreams.

The paper uses Apache Superset for visualization because of its extensive collection of data visualisation tools (including maps), simple interface for uploading and transforming data, and seamless integration with widely used data stores. Whereas in our project we have utilized Graphana for visualization purposes.

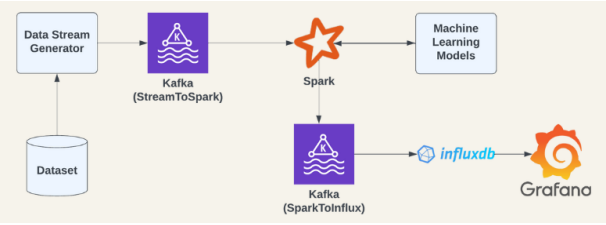
The Isolation Forest Algorithm was used by the authors to detect anomalies. The Isolation Forest (IFO) algorithm detects anomalies by identifying instances with attribute values that differ significantly from the majority of the data. It partitions data samples using an ensemble of random trees, isolating anomalies more efficiently than normal instances. The algorithm computes anomaly scores by averaging path lengths across all trees and requires only two user parameters: the number of trees and the size of the subsample. Because of its linear time complexity, low memory requirements, and scalability for large datasets, IFO is an efficient solution for anomaly detection.

This algorithm uses Principal Component Analysis (PCA) to reduce the dimensionality of training data, assuming that the nominal temperature profile can be effectively represented by a few principal components. This PCA-based method provides a streamlined approach to anomaly detection that is particularly suitable for scenarios where the nominal temperature profile can be adequately captured in a reduced-dimensional space.

Whereas in our work we have employed a gradient based learner as well as a tree based learner. Alongwith this we used The Local Outlier Factor (LOF) which is a machine learning algorithm used for anomaly detection and outlier detection in datasets.

It was introduced by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander in a 2000 research paper. LOF is particularly useful for finding anomalies in high-dimensional datasets and those with complex structures.

IV. Architecture of proposed model.



1) Docker used for running kafka, zookeeper, spark, influxdb and grafana in containers.
2) Docker-Compose used to configure the services
3) Kafka acting as a message broker between data source & spark and spark & influxDB.
4) Spark used for real time processing and prediction.
5) Multiple spark brokers running simultaneously (master-slave architecture) for efficient computation.
6) InfluxDB is a high speed read and write noSQL database used in time series applications
7) Grafana is a data visualization tool used for creating dashboards, alerts etc.

We will be utilising the Occupancy Detection Data Set UCI dataset. Data on occupancy detection in office buildings may be found in the "Occupancy Detection Data Set" from the UCI Machine Learning Repository. The dataset is commonly employed in binary classification problems, where the objective is to ascertain the presence or absence of occupants in a room by analysing many sensor signals.

Date and time, temperature, relative humidity, light, CO2, humidity ratio, and occupancy are some of the properties. The four main ideas of Kafka are broker, topic, producer, and consumer. Messages can be arranged using Kafka topics, which act as intermediate data containers for records that are transferred between systems and applications. For various types of sensors, the user defines the topic data schema. Each topic is divided into multiple internal sections for quicker information retrieval and data redundancy. A Kafka consumer reads from a partition, whereas a producer writes to a topic. A hardware node in a distributed system that manages load balancing, reading, and writing is called a Kafka broker. It is up to the user to define producer(s) and consumer(s).

Application programming interfaces (API) are made available by Kafka to developers so they can create unique producers and consumers. Furthermore, the Kafka connectors enable the configuration of sources and sinks that link Kafka topics to well-known programmes or data systems through industry-standard interfaces like Amazon S3, Hadoop Distributed File System (HDFS), and JDBC (relational SQL databases). The CEP is connected to the data layer and knowledge discovery layer via custom connectors, which automate the information exchange between the CEP and those layers. For our use case, we also created producer and consumer templates that make it simple to modify the workflow to accommodate various sensor datastream types.

Supporting decision-making and knowledge discovery both benefit greatly from visualisation.In recent years, a number of open-source data portals for business intelligence have emerged. Although Apache Superset is mentioned in our problem statement, we have decided to use Grafana as our visualisation tool because of current constraints. It offers a wide range of data visualisation tools, a user-friendly interface for importing and altering data, and smooth integration with widely used data repositories.

The paper mentions use of Apache Superset but due to constraints we have used Grafana for our visualization purpose. After preliminary evaluation of several products, they have chosen the Apache Superset (https://superset.incubator.

apache.org) because of its rich collection of data visualization tools (including maps), easy-to-use interface for uploading and transforming data, and seamless integration with commonly used data stores. Challenges specific to geosciences are (a) data tends to vary both spatially and temporally, representing disparate scales and storage formats (Chen et al., 2014), (b) no single anomaly detection algorithm fits all purposes, (c) the "nominal model" or baseline is elusive in many situations, (d) anomalies may be shadowed by noise, (e) labeled anomaly data is rare, creating imbalance in the training data, and finally (f) establishing the causal mechanism (i.e., event attribution) can be challenging for subsurface processes.

V. Dataset description

Code | Blame | 8144 lines (8144 loc) · 583 KB

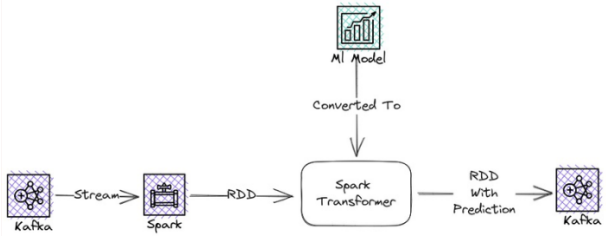
```
"date","Temperature","Humidity","Light","CO2","HumidityRatio","Occupancy"
"1","2015-02-04 17:51:00",23.18,27.272,426,721.25,0.00479298817650529,1
"2","2015-02-04 17:51:59",23.15,27.2675,429.5,714,0.00478344094931065,1
"3","2015-02-04 17:53:00",23.15,27.245,426,713.5,0.00477946352442199,1
"4","2015-02-04 17:54:00",23.15,27.2,426,708.25,0.00477150882608175,1
"5","2015-02-04 17:55:00",23.1,27.2,426,704.5,0.00475699293331518,1
"6","2015-02-04 17:55:59",23.1,27.2,419,701,0.00475699293331518,1
"7","2015-02-04 17:57:00",23.1,27.2,419,701.6666666666667,0.00475699293331518,1
"8","2015-02-04 17:57:59",23.1,27.2,419,699,0.00475699293331518,1
"9","2015-02-04 17:58:59",23.1,27.2,419,689.3333333333333,0.00475699293331518,1
"10","2015-02-04 18:00:00",23.075,27.175,419,688,0.00474535071966655,1
"11","2015-02-04 18:01:00",23.075,27.15,419,690.25,0.00474095189694268,1
"12","2015-02-04 18:02:00",23.1,27.1,419,691,0.00473937073052061,1
"13","2015-02-04 18:03:00",23.1,27.1666666666667,419,683.5,0.00475111875560951,1
"14","2015-02-04 18:04:00",23.05,27.15,419,687.5,0.0047337317970825,1
"15","2015-02-04 18:04:59",23,27.125,419,686,0.00471494214590473,1
"16","2015-02-04 18:06:00",23,27.125,418.5,680.5,0.00471494214590473,1
"17","2015-02-04 18:07:00",23,27.2,0,681.5,0.00472807794966877,0
"18","2015-02-04 18:08:00",22.945,27.29,0,685,0.00472795137178073,0
"19","2015-02-04 18:08:59",22.945,27.39,0,685,0.00474540839700941,0
"20","2015-02-04 18:10:00",22.89,27.39,0,689,0.00472950615591001,0
"21","2015-02-04 18:10:59",22.89,27.39,0,689.5,0.00472950615591001,0
"22","2015-02-04 18:11:59",22.89,27.39,0,689,0.00472950615591001,0
"23","2015-02-04 18:13:00",22.89,27.445,0,691,0.00473907551474663,0
"24","2015-02-04 18:14:00",22.89,27.5,0,688,0.00474864516581148,0
"25","2015-02-04 18:15:00",22.89,27.5,0,689.5,0.00474864516581148,0
```

The dataset which we will be using is Occupancy Detection Data Set UCI. The "Occupancy Detection Data Set" from the UCI Machine Learning Repository is a dataset that contains information related to occupancy detection in an office building. The dataset is typically used for binary classification tasks where the goal is to determine whether a room is occupied or not based on various sensor measurements. The attributes include: Date and Time, Temperature, Relative Humidity, Light, CO2, Humidity Ratio, Occupancy. **Here's a breakdown of the dataset attributes:**

- **Date and Time:**The date and time os recorded when the sensor data is collected. Temporal patterns in occupancy are likely to be established by the attribute.
- **Temperature:** It represents the temperature which is being measured around and in the environment of office space.
- **Relative Humidity:** It showcases the level of humidity relative to that of the maximum humity at the particular given temperature. It is also responsible to measure the moisture content in the air.
- **Light:**Refers to the intensity of light measured in the area. It signifies the brightness level in the office space.
- **CO2:** Represents the carbon dioxide concentration in the air. Elevated CO2 levels might indicate occupancy due to human respiration.
- **Humidity Ratio:** Represents the carbon dioxide concentration in the air. Elevated CO2 levels might indicate occupancy due to human respiration.
- **CO2:** Represents the carbon dioxide concentration in the air. Elevated CO2 levels might indicate occupancy due to human respiration.

VI. Experiment details

A. Prediction Architecture



Data Generator works as a source that continuously generates data. The Data Generator class is used to generate this data from where data points are generated which are further ingested in the Ingestor. This is performed by the ingestor class. This data is ingested to the InfuxDB.

```
version: '3'
services:
  superset:
    image: apache/superset
    container_name: superset
    ports:
      - "8088:8088"
    environment:
      - SUPERSET_SECRET_KEY=q9utf37t3wqfg9wf7bjkwpyqgfqwph08
    volumes:
      - ../superset-volume:/var/lib/superset
    networks:
      - superset-network
  influxdb:
    image: influxdb:latest
    container_name: influxdb
    ports:
      - "8086:8086"
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=admin_password
      - DOCKER_INFLUXDB_INIT_ORG=IIITA
      - DOCKER_INFLUXDB_INIT_BUCKET=bda
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=my-super-secret-auth-token
    volumes:
      - ./influx-volume:/var/lib/influxdb2
networks:
  superset-network:
```

Docker compose to create a network for Apache Superset and Influx DB and initialize container for both of them.

```
→  Project git:(main) docker-compose up
Creating network "project_superset-network" with the defau
Creating network "project_default" with the default driver
Creating superset ... done
Creating influxdb ... done
```

We tried to implement Apache Superset in the first component but due to issues related to paid subscription and other issues which are mentioned in the image below.

1) Data Generator: Data generator: gets data when get function is called.

2) Data Ingestor: Data ingestor: Ingests data when get function is called.

3) Ingestion to InfluxDb: Data ingestion into influxDB successful. The tags are selected to be CO2 and Occupancy. Tags are used for indexing.

```python
class DataGenerator:

    def __init__(self,
                 file_loc: str,
                 measurment: str,
                 tags: list[str],
                 time_delay=1,
                 circular_generator=False) -> None:
        """DataGenerator is used to generate the data from a csv

        Args:
            file_loc (str): Location of the csv file to generate data from
            measurment (str): Name of the table where we want to place the data
            tag (list[str]): Columns used as tag, should not be date
            time_delay (int, optional): Delay for each datapoint. Defaults to 1.
            circular_generator (bool, optional): Whether to generate data from start again. Defaults to False.
        """
        self.measurment = measurment
        self.tags = tags
        self.time_delay = time_delay
        self.circular_generator = circular_generator

        df = pd.read_csv(file_loc)
        if "date" in df.columns:
            df.drop(columns=["date"],inplace=True)

        self.data = df
        self.columns = list(set(df.columns) - set(self.tags))

    def get(self) -> Point:
        idx = 0
```

```python
class Ingestor:
    def __init__(self,
                 token: str,
                 org: str,
                 url: str,
                 bucket: str,
                 generator: DataGenerator) -> None:
        """Ingestor is used to ingest the data in influxdb

        Args:
            token (str): The token being used for authentication
            org (str): The organisation of the user
            url (str): The url to access the influxdb
            bucket (str): The bucket used for storage of the data
            generator (DataGenerator): A generator whose get function can be called to get a datapoint
        """
        self.generator = generator
        self.bucket = bucket
        self.org = org

        self.client = InfluxDBClient(url=url, token=token, org=org)

    def ingest(self) -> None:
        with self.client.write_api(write_options=SYNCHRONOUS) as write_client:
            for point in self.generator.get():
                write_client.write(
                    bucket=self.bucket,
                    org=self.org,
                    record=point
                )
```

B. Methodology

Create a data delivery system, that continuously produces the data. For this the same data frame is iterated again and again. Create a data ingestion system that ingests the data to influx DB. Create a connection from influx DB to Grafana so that data can be queried and graphs can be analyzed.

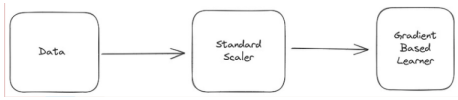
- **Phase 1:** Create a data delivery system, that continuously produces the data. For this the same data frame is iterated again and again.
- **Phase 2:** Create a data ingestion system that delivers the data to spark, then processes the data and ingests it into influx db.
- **Phase 3:** Create a connection from influx DB to Grafana so that data can be queried and graphs can be analyzed.
- **Phase 4:** Create alerts in grafana which will help to analyse any abnormalities in the building data

C. Gradient based learner

A machine learning algorithm known as a gradient-based learner optimises and updates its parameters using gradient descent to minimise a given cost or loss function.

An iterative optimisation technique called gradient descent is used to determine a function's minimum. This function is usually the cost or loss function that calculates the difference between expected and actual values in the context of machine learning. The gradient, which points in the direction of a function's steepest ascent, is used to update the model's parameters in the opposite direction with the goal of reaching the function's minimum.

Our model includes a standard scaler which gets data and then passes it in the Gradient Based Learner.



D. Tree Based Learner

A class of machine learning algorithms known as "tree-based learners" builds decision trees or ensembles of trees in order to generate predictions. These algorithms divide the feature space recursively into regions, then predict the target variable based on the average (for regression) or majority (for classification) of those regions.

Tree-based learners come in a variety of forms:

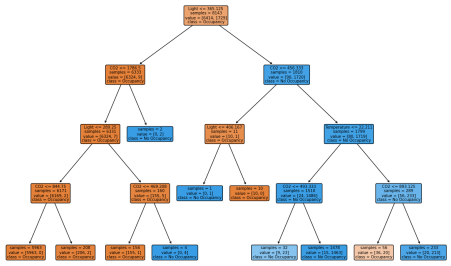
1) Decision trees are hierarchical structures in which decisions are represented by nodes based on features, and branches branch out based on additional features. Until a halting condition is satisfied, like reaching a maximum depth or purity threshold, the process keeps going.
2) The Random Forest ensemble method reduces overfitting and increases accuracy by constructing several decision trees and combining their predictions. A random subset of the data and features are used to train each tree, and the final output is either voted (classification) or averaged (regression) based on the predictions made.
3) Gradient Boosting Machines (GBM): Gradient Boosting, XGBoost, LightGBM, and CatBoost are some of the algorithms that generate a series of trees, each of which aims to fix the mistakes made by the one before it. By gradually fitting new trees to the residuals of the current ensemble, these models lower the prediction error overall.

E. Grafana

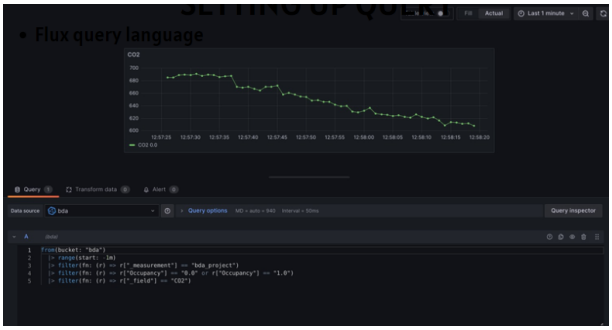
Supporting decision-making and knowledge discovery both benefit greatly from visualisation. Although using Apache Superset is mentioned in our problem statement, we have decided to use Grafana as our visualisation tool instead of Apache Superset because of current constraints. It offers a wide range of data visualisation tools, a user-friendly interface for importing and altering data, and smooth integration with widely used data repositories. Visualized data on the basis of various features. The features include:

1) CO2 level
2) Humidity
3) Temperature
4) Light

Flux query language is used to display the dashboards. Added alerts when CO2 goes above 300. Here, 300 is the threshold value.

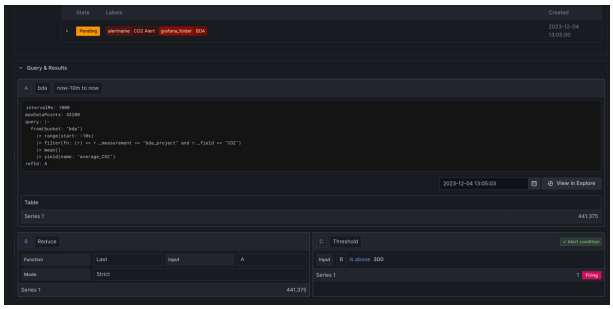


1) Setting up query: The purpose of Influx Query Language (InfluxQL) is to query data kept in the time-series database InfluxDB. Grafana is a widely used visualisation tool that frequently integrates with InfluxDB. This enables users to use InfluxQL queries to create powerful visualisations of time-series data. When creating visualisations with Grafana, users can enter these InfluxQL queries straight into the interface. These queries can be used in conjunction with different visualisation options such as tables, bar charts, and line charts to display time-series data stored in InfluxDB quite effectively.



2) Setting up alert: There are several steps involved in using InfluxDB to set up alerts in Grafana. Here's a broad overview:

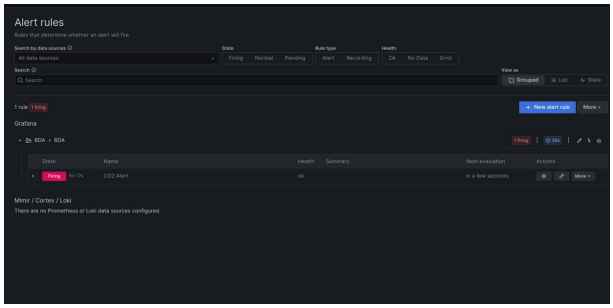
1) Data Source Configuration: First, make sure Grafana has InfluxDB configured as a data source. In Grafana, navigate to Configuration (gear icon) ¿ Data Sources. To add a data source, click. Choose InfluxDB. Input the required information, such as the name, URL, database, and authentication details. To make sure the data source is connected, save and test it.
2) Construct a Dashboard: To set an alert for a particular panel (graph, table, etc.), create or open a dashboard that contains that panel.
3) Construct Alerts: Establish or Modify a Panel:
   You can set alerts for a panel by clicking on its title. Navigate to the Settings panel's Alert tab.
4) Specify the Alert Conditions:
   To create an alert, click. Using the data you have, define conditions. You may, for example, create a condition that says, "when CPU usage exceeds 90Select the evaluation frequency and the amount of time that should pass before an alert is sent.
5) Set Up Notifications:
   In the Notifications section, define the channels for notifications. Grafana is compatible with multiple notification channels, including Slack, PagerDuty, email, and more. Set up the recipients, message type, and notification frequency.
6) Test the Alert: To make sure your alert is functioning properly, it is imperative that you test it. To set off the alert and confirm that notifications are being sent as expected, create test data or mimic a situation that matches your alert criteria.



3) Alert Firing: When an alert sounds in Grafana, it indicates that the prerequisites have been satisfied. This usually occurs in accordance with the parameters specified in the alert rule for a particular dashboard panel.

When an alert goes off, the following happens:

Condition Met: The panel's query and the data inside the chosen time window satisfy the alert rule's requirements. For example, an alert will sound if you have set a threshold for CPU usage to exceed 90 percent for five minutes, and the incoming data meets this condition. Transition to Alerting State: The alert goes from "OK" to "Alerting" when the necessary conditions are satisfied. This alteration in status signifies that the alert's condition has changed.

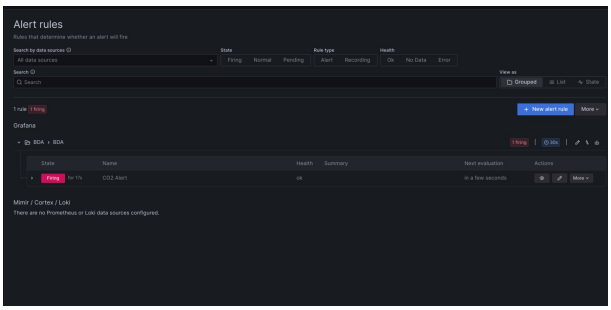


VII. Result

We use data Stream Generator which takes data from the dataset and this is passed through the Kafka topic which is named as StreamToSpark. This acts as a messenger and broker between the stream and Spark. This data stream is then ingested to Spark from where it is used to train Machine Learning Models. The models use Gradient based learner and Tree based learner. Then the information which is generated

from this is again ingested in Spark. After the data is passed through the models, it is passed from Spark to InfuxDB through a kafka topic which is named as SparkToInfux. This data is then ingested in InfuxDB. After this for visualization purposes we have used Grafana which takes the data from InfuxDB and visualizes dashboard on the screen based on the features we choose. This way we can visualize, query and add alerts. Alerts can be added and a threshold value can be set based on which alerts are generated. This is our entire workflow which we have worked on in our for BDA Project.

This shows the Grafana Dashboard which we have visualized based on various features. This shows our alert firing



system, where we can set alerts and threshold values.

VIII. References

[1] Complex Event Processing

[2] Building complex event processing capability for intelligent environmental monitoring

[3] A Survey of Distributed Data Stream Processing Frameworks

[4] Real-Time Streaming in Big Data: Kafka and Spark With Singlestore