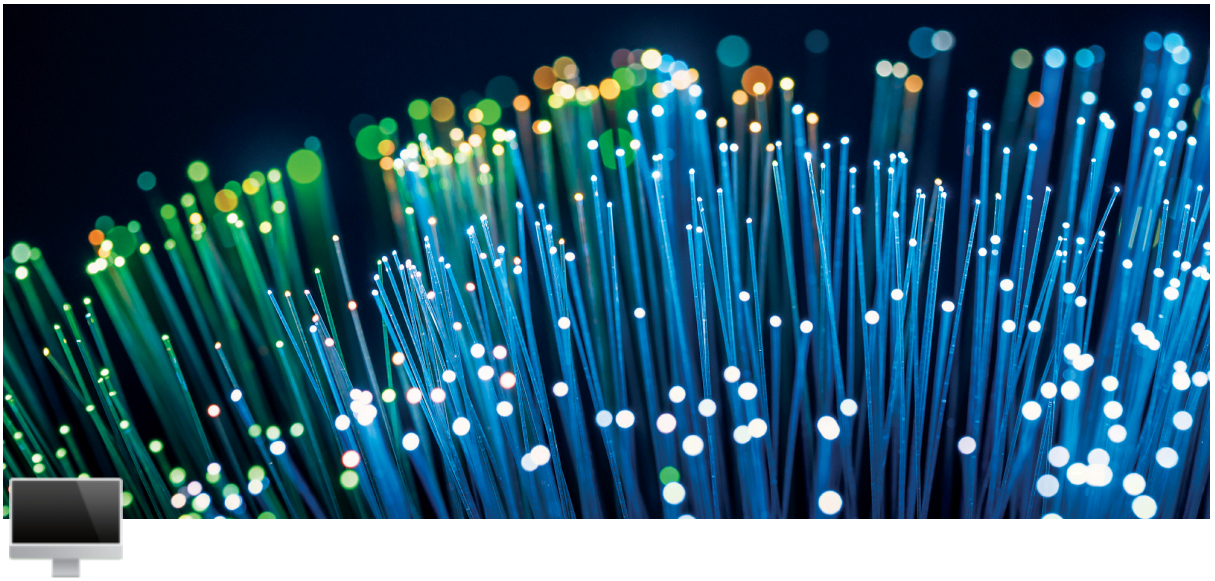


Unified Telemetry and Monitoring Platform User Guide

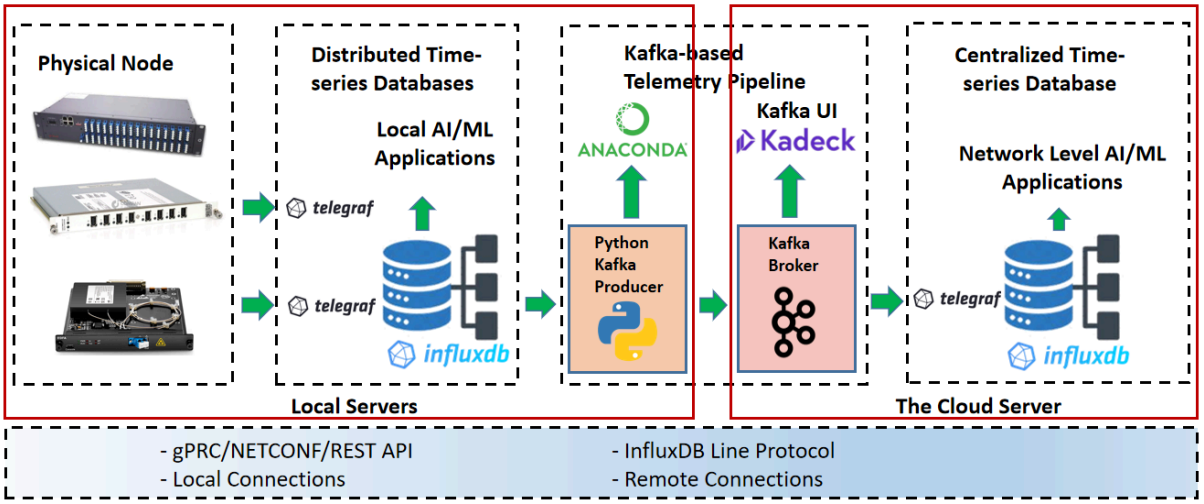
1. Introduction

Our unified telemetry and monitoring platform, based on Apache Kafka and InfluxDB with Kafka Streams and InfluxDB plugins, is deployed over the UK

National Dark Fibre Facility (NDFF). This platform aims to achieve real-time data collection, streaming, and support for advanced local and network-level ML applications.

Within this proposed framework, both distributed and centralized databases are implemented using InfluxDB, an open-source framework designed for high-performance handling of time-series data. In the local connections from the physical layer to distributed databases, distributed databases deployed on each node are responsible for fetching, sampling and aggregating massive amounts of performance data. Telegraf, a plugin-driven server agent, is developed by InfluxData to collect and process data. The Telegraf agent enables data to be collected from various sources and sent to different storage systems efficiently.

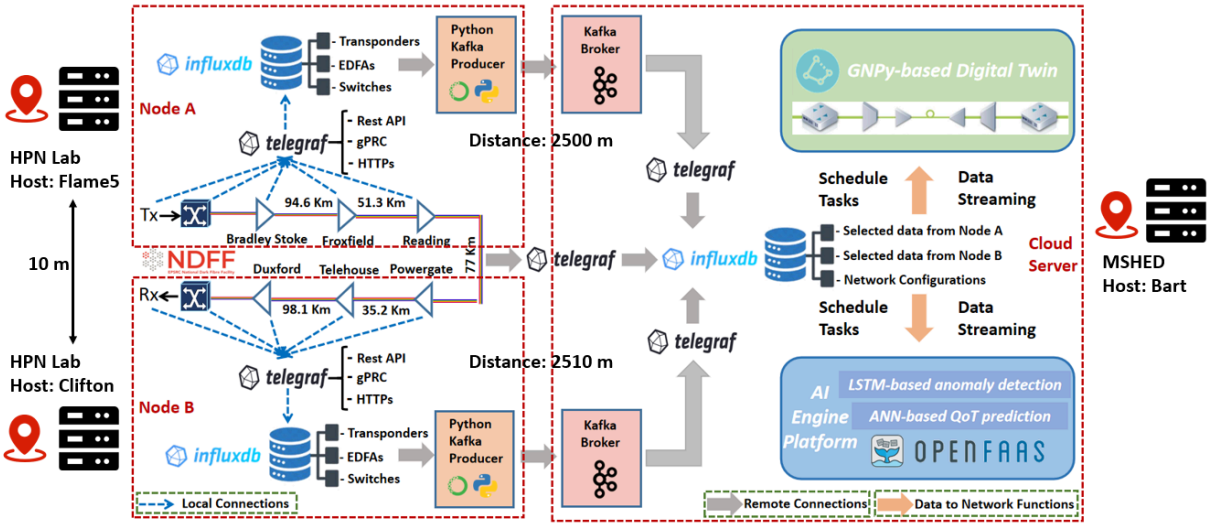
On the other hand, a centralized database is designed to store network configuration information from the optical network and filtered telemetry data from distributed databases via remote telemetry service to support network-level ML applications. For remote connections, Kafka-based telemetry pipelines are deployed between distributed and centralized databases to ensure highly efficient and reliable telemetry data transmission. Each distributed database hosts a Python Kafka Producer using Anaconda to send data to Kafka clusters for processing. Kafka brokers, which are part of the cluster, act as server instances to receive, manage and store telemetry data sent by producers. Each broker hosts multiple Kafka topics to partition the workload among the available brokers, thus ensuring data reliability and increasing the overall speed of data retrieval. In order to provide a user-friendly interface to help users monitor and manage Kafka clusters, Kadeck, a data visualization and management tool designed specifically for Apache Kafka, is adopted in the telemetry pipelines aiming to enhance data transparency and mitigate potential failures that may arise in brokers. Besides, Telegraf agents are deployed to facilitate remote connections and automatically forward data managed by Kafka brokers to the centralized database.



Telemetry service design

2. Get Started

The proposed platform has been fully deployed on three virtual machines (VMs) running Ubuntu 22.04 (Graphical Interface: Mate) in our testbed over the NDFF, each with 4 CPUs, 8 GB of RAM, and 50 GB of storage. Two distributed databases are deployed on two different servers (Host Flame5 and Host Clifton) in the HPN lab, whereas the centralized database is deployed on the server located on Mshed (Host Bart) which has a certain physical distance from the HPN lab.



Field-trial Implementation over NDFF

2.1 System Environment and Version

The systems on three VMs are Ubuntu 22.04.4 LTS (Kernel version: 5.15.0-113-generic) with the graphical interfaces (Mate version: 1.26.0).

2.2 Software Installation and Configuration on Local Servers

On the local servers, the required software or packages are InfluxDB, Telegraf, Grafana, Anaconda and GitHub packages of device developments.

To provide operators with comprehensive insights into the proposed framework, Grafana, an open observability and data visualization platform, offers extensive functionalities in querying, visualisation and alerting to monitor network performance by reading raw data from various data sources, establishing a seamless connection to InfluxDB through its HTTP API.

The following steps show the process of downloading and installing these software tools, as well as problems that may arise during installation.

2.2.1 Anaconda: 🔗Anaconda Installation on Linux

1. Prerequisites: Debian

```
sudo apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1    libxcomposite1 libasound2 libxi6 libxtst6
```

2. Download the installer: Linux x86

```
sudo curl -O https://repo.anaconda.com/archive/Anaconda3-<INSTALLER_VERSION>-Linux-x86_64.sh
```

- ***<Installer version>*** can be replaced by any Anaconda version you want to download.

3. Install

```
bash ~/Downloads /Anaconda3-<INSTALLER_VERSION>-Linux-x86_64.sh
```

- replace ***~/Downloads*** with your actual download path.

Then, we can lauch Anaconda Navigator from the command line.

```
anaconda-navigator
```

Problems you may encounter:

1. **QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-ubuntu'**

This warning indicates that the environment variable `XDG_RUNTIME_DIR` is not set, so the system returns to a default location (`/tmp/runtime-ubuntu`). `XDG_RUNTIME_DIR` is typically set automatically by the system and is used for storing runtime files unique to the user.

```
echo $XDG_RUNTIME_DIR
```

- If the output is empty or undefined, it means `XDG_RUNTIME_DIR` is not currently set.

```
echo $UID
```

- Check user ID

```
nano ~/.bashrc
export XDG_RUNTIME_DIR=/run/user/<user ID>
source ~/.bashrc
```

- Add it to the end of the configuration file.

2. **libGL error: MESA-LOADER: failed to open swrast: /usr/lib/dri/swrast_dri.so: cannot open shared object file: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)**
   libGL error: failed to load driver: swrast

```
sudo apt update
sudo apt install --reinstall libgl1-mesa-dri
```

- Ubuntu may flash back and re-log with a black screen. Then, it may show that they *could not acquire name in session bus*.

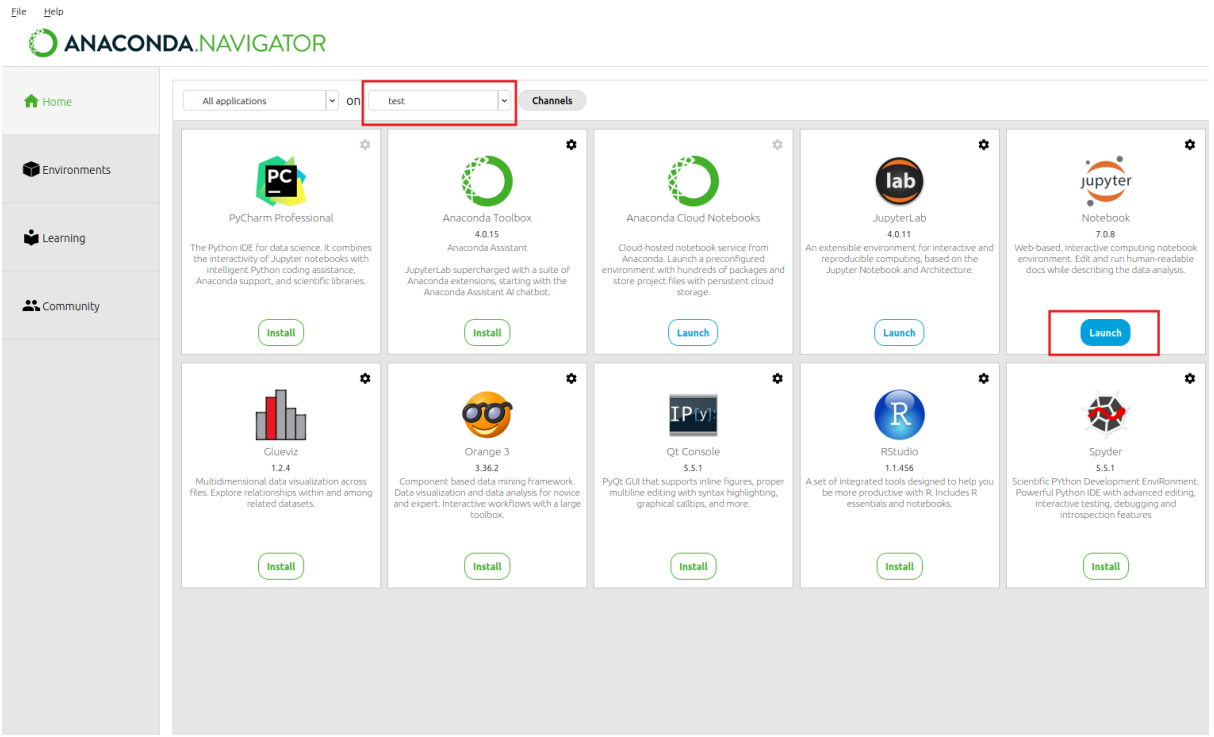
```
sudo systemctl restart dbus
sudo reboot
```

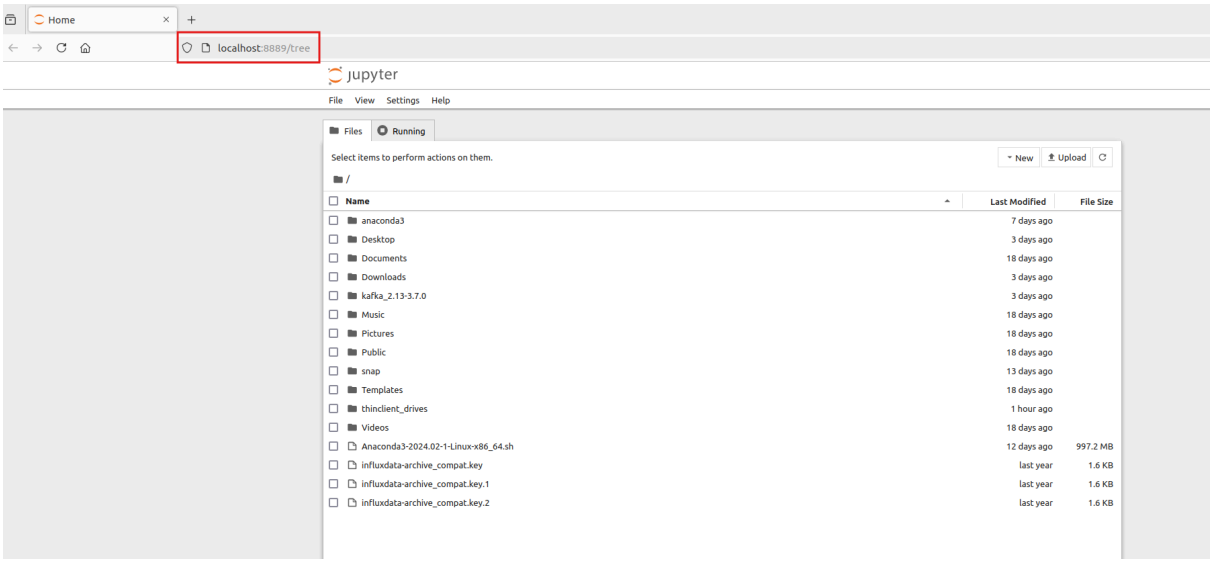
Before launching Jupyter Notebook in Anaconda, we can set up a new environment. The default environment of Anaconda is "base", but we can create a new one called "test".

```
#Create a new Anaconda environemnt
conda create --name test python=3.10
```

- **test**: you can change the name of the new environment to whatever you prefer.
- python=**3.10**: you can adjust the Python version of this new environment to match your needs, such as 3.8, 3.9, etc.

Next, launch Jupyter Notebook in Anaconda by clicking on the "Install" and "Launch" option.





2.2.2 InfluxDB: 🔗InfluxData Downloads

Select: Version: v2.7.6 Platform: Ubuntu & Debian

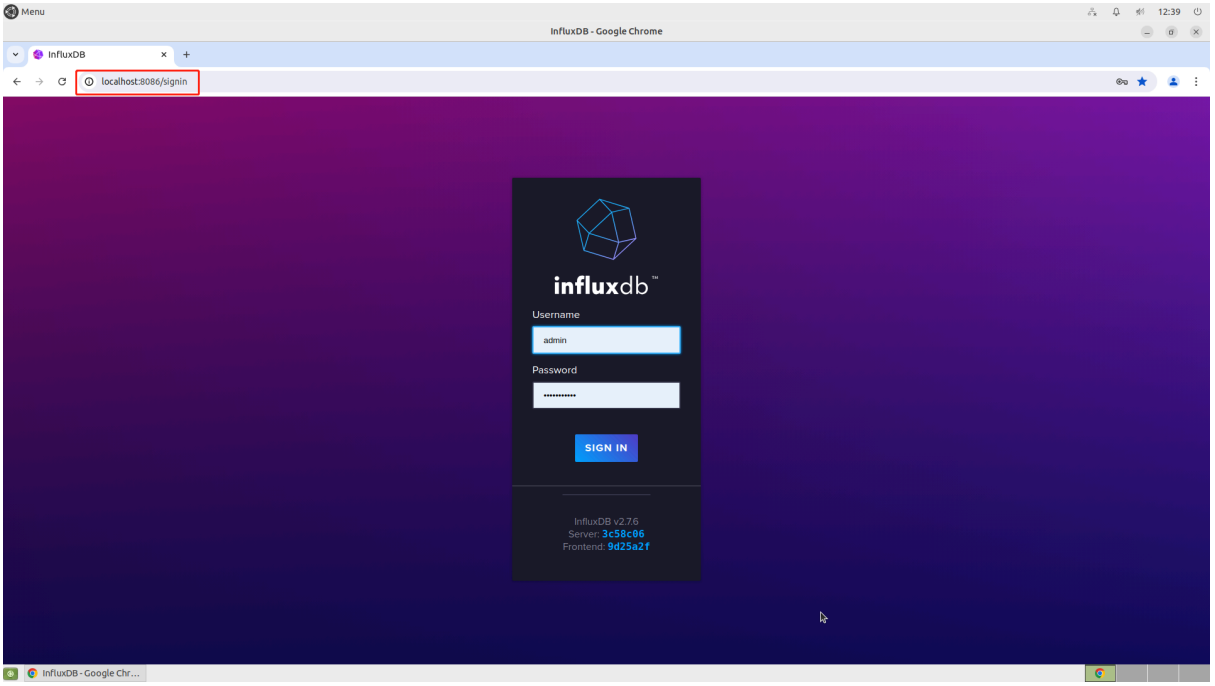
```
# To download InfluxDB
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6a
c9ce4c influxdata-archive_compat.key' | sha256sum -c && cat influxdata-a
rchive_compat.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influx
data-archive_compat.gpg > /dev/null
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gp
g] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sour
ces.list.d/influxdata.list

sudo apt-get update && sudo apt-get install influxdb2
```

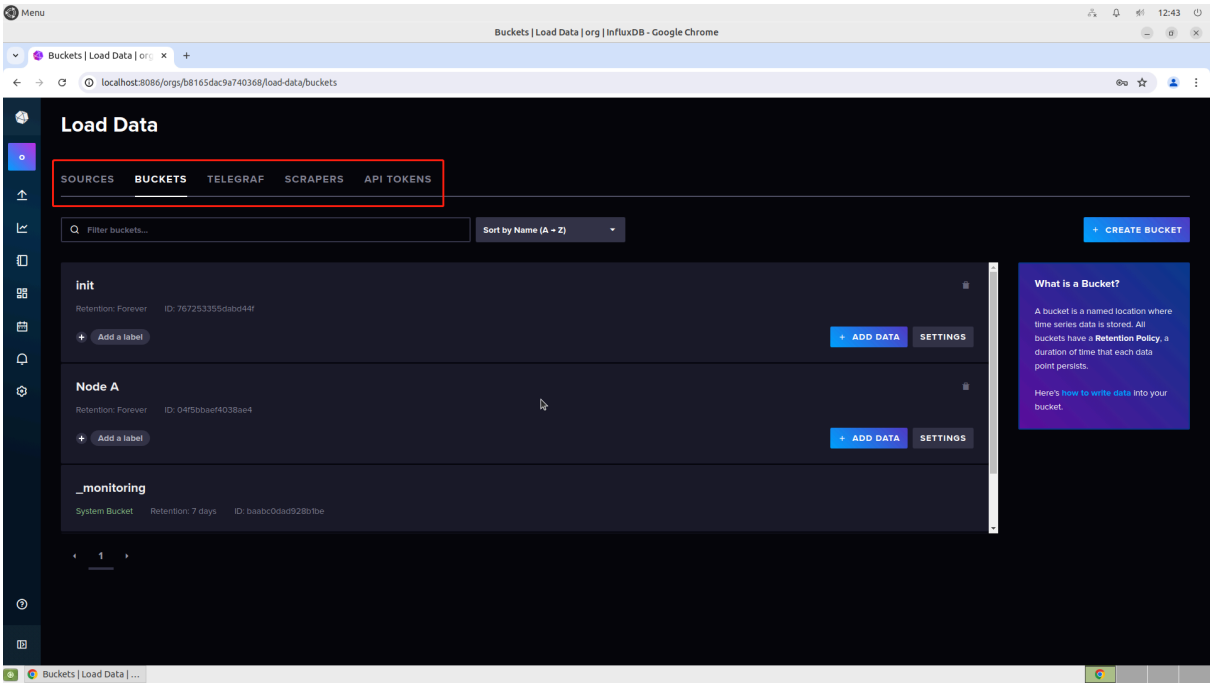
```
# Start the InfluxDB service and set it to startup
sudo systemctl start influxdb
sudo systemctl enable influxdb
```

InfluxDB 2.0 provides a built-in web interface that you can access in your browser: http://localhost:8086 or http://10.68.100.115:8086 (For example, 10.68.100.115 is your server IP)

Then, you can create the username & password, and set the organization name. After that, you can create different buckets (databases) in InfluxDB and

start to write data into it.





InfluxDB provides flexible methods for data writing. You can upload CSV or line protocol files into InfluxDB. Also, you can use InfluxDB Client Libraries supporting many languages such as Python, R, C#, Java and GO. The codes below show the data writing and query through Python InfluxDB Client.

```
# data writing
from influxdb_client import InfluxDBClient, Point, WritePrecision

# configuration
token = "your_token"
org = "your_org"
bucket = "your_bucket"
url = "http://your-server-ip:8086"

# create the client
client = InfluxDBClient(url=url, token=token)

# get the API
write_api = client.write_api(write_options=SYNCHRONOUS)

# write the data
point = Point("temperature").tag("location", "office").field("value", 23.5)
write_api.write(bucket=bucket, org=org, record=point)

print("Data written successfully.")

-------------------------------------------------------------------------
-

# data query
from influxdb_client import InfluxDBClient
from influxdb_client.client.query_api import QueryApi

token = "your_token"
org = "your_org"
bucket = "your_bucket"
url = "http://localhost:8086"

client = InfluxDBClient(url=url, token=token)

query_api = client.query_api()

# data query
```

```
query = f'from(bucket: "{bucket}") |> range(start: -1h) |> filter(fn: (r) => r._
measurement == "temperature")'
tables = query_api.query(query, org=org)

# print the results
for table in tables:
    for record in table.records:
        print(f'Time: {record.get_time()}, Location: {record.values["location"]},
Value: {record.get_value()}')
```

2.2.3 Grafana: 🔗Install Grafana on Debian or Ubuntu

1. Install the prerequisite packages:

```
sudo apt-get install -y apt-transport-https software-properties-common w
get
```

2. Import the GPG key:

```
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /e
tc/apt/keyrings/grafana.gpg > /dev/null
```

3. To add a repository for stable releases, run the following command:

```
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.c
om stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
```

4. Run the following command to update the list of available packages:

```
# Updates the list of available packages
sudo apt-get update
```

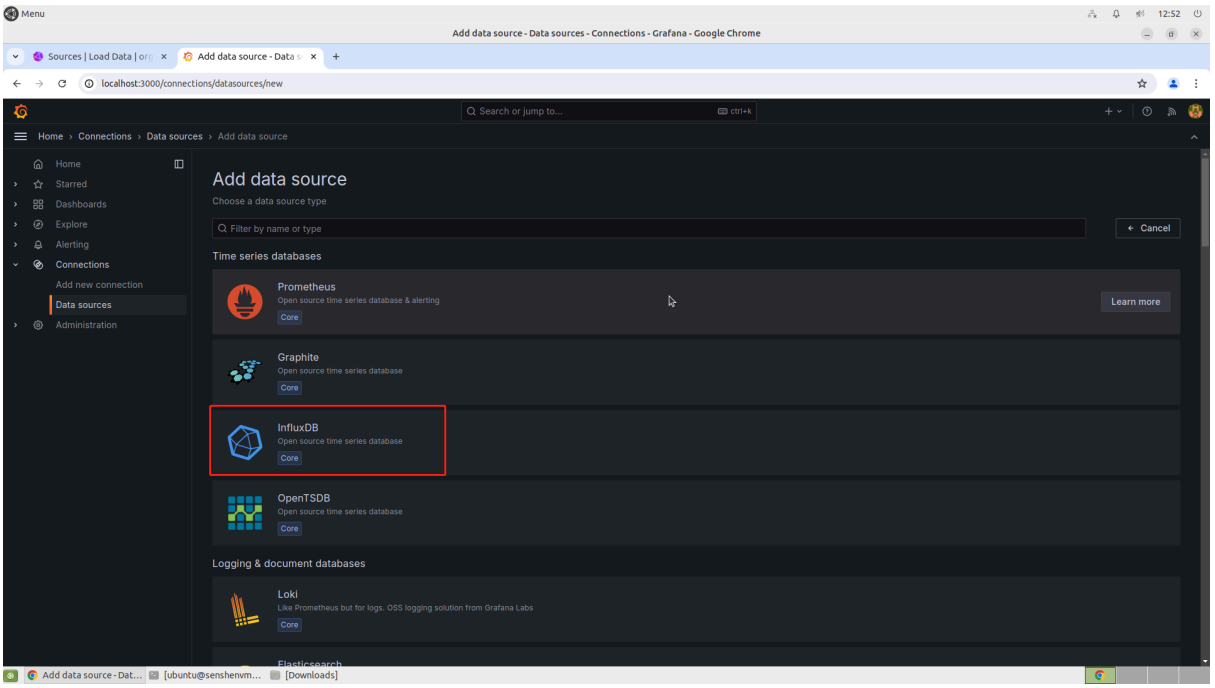
5. To install Grafana OSS, run the following command:

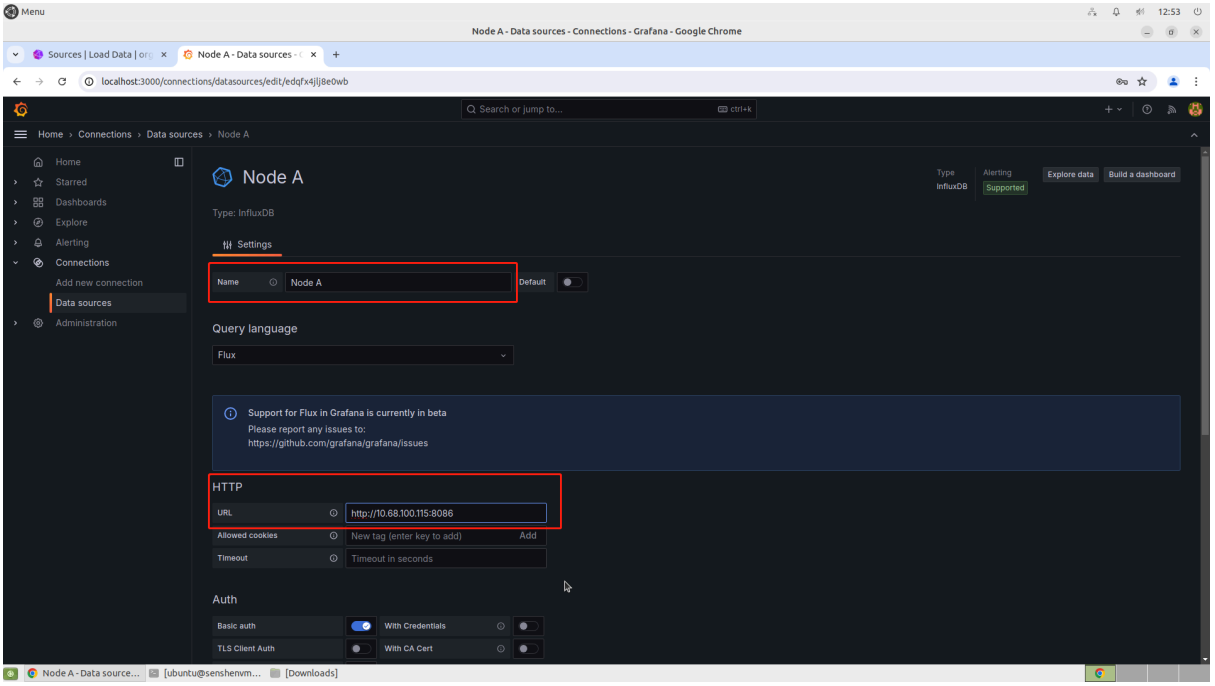
```
# Installs the latest OSS release:
sudo apt-get install grafana
```

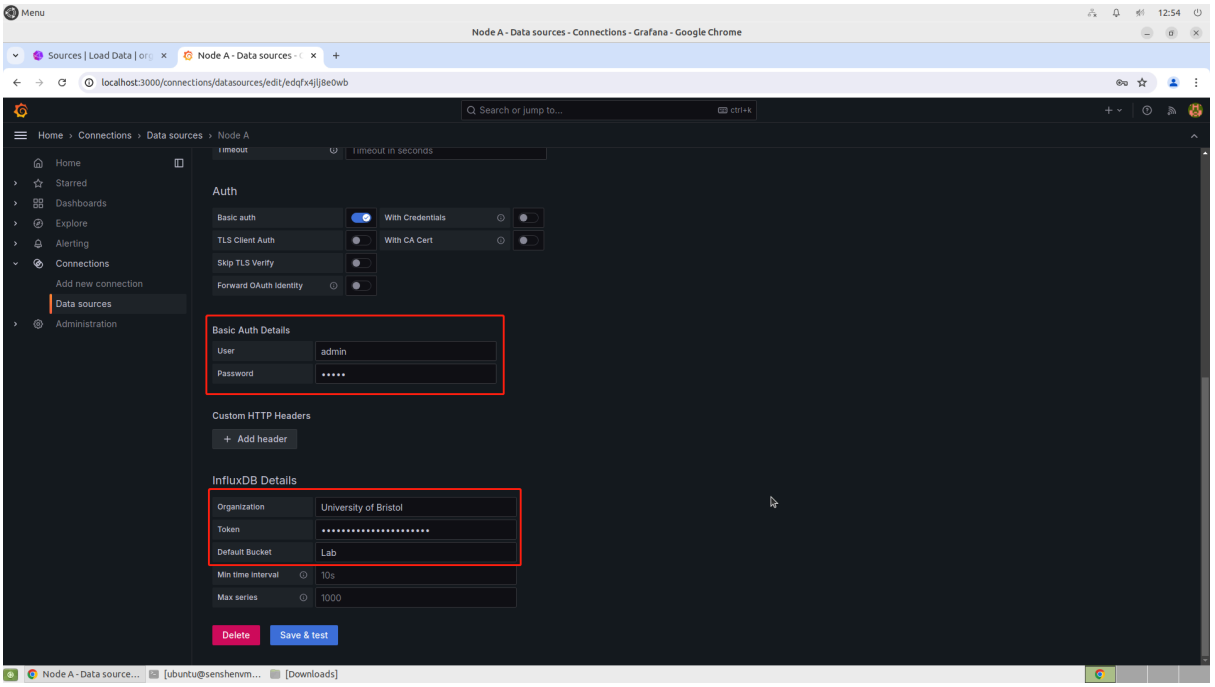
```
# Start the InfluxDB service and set it to startup
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

Then, you can access Grafana interface in your browser: http://localhost:3000 or http://10.68.100.115:3000 (For example, 10.68.100.115 is your server IP)

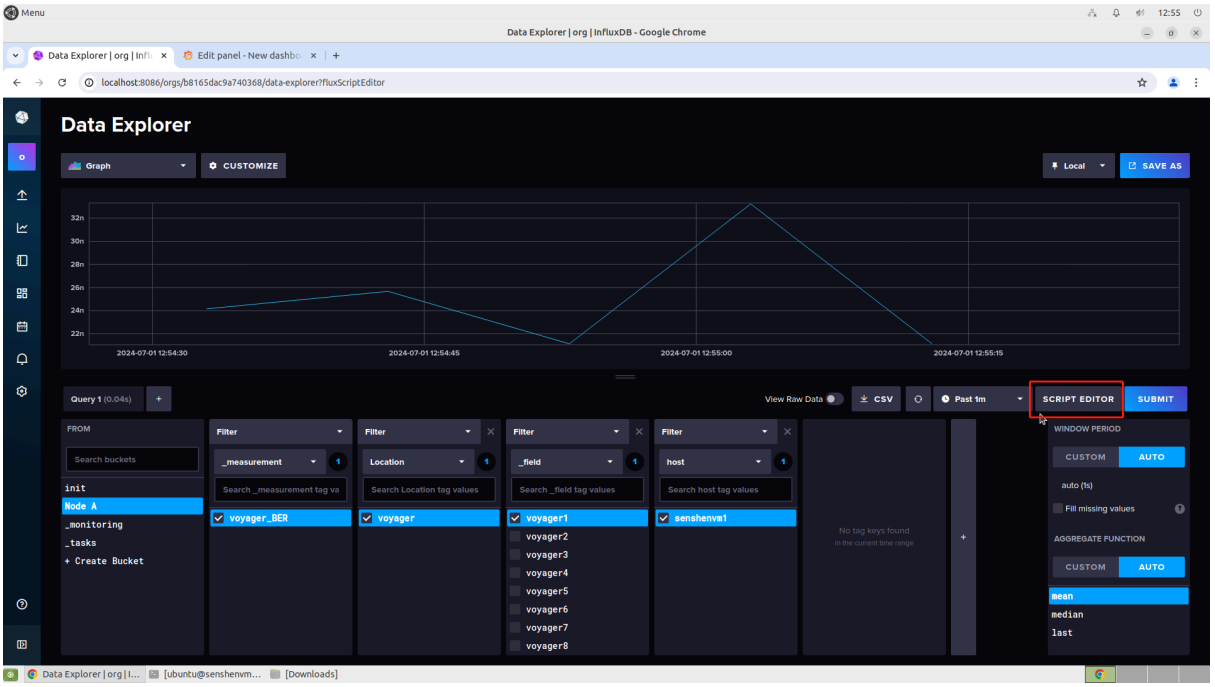
When creating the dashboard in Grafana, you can add the data source from InfluxDB shown in the figures below.

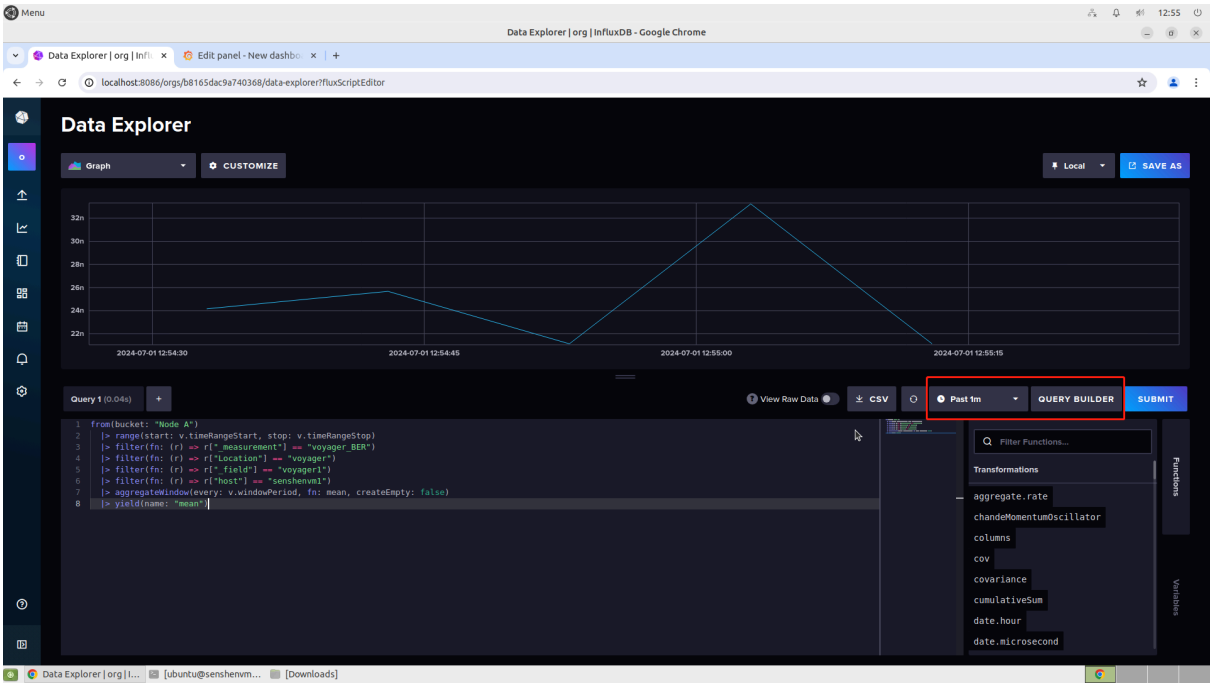


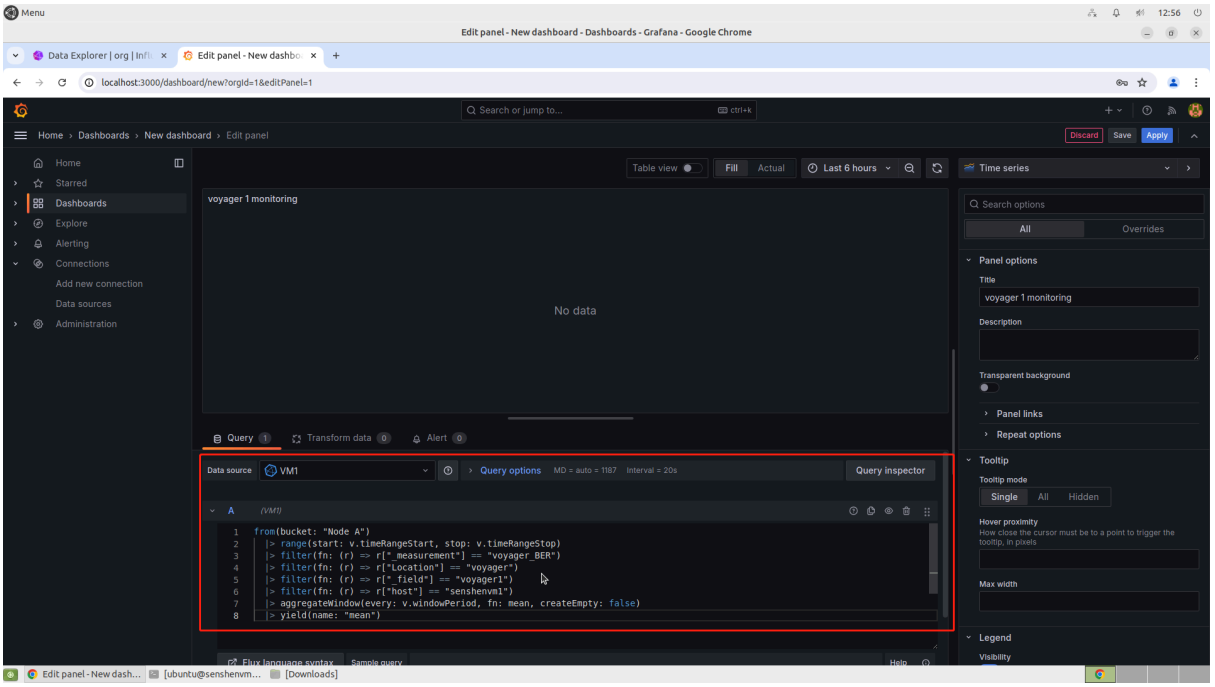


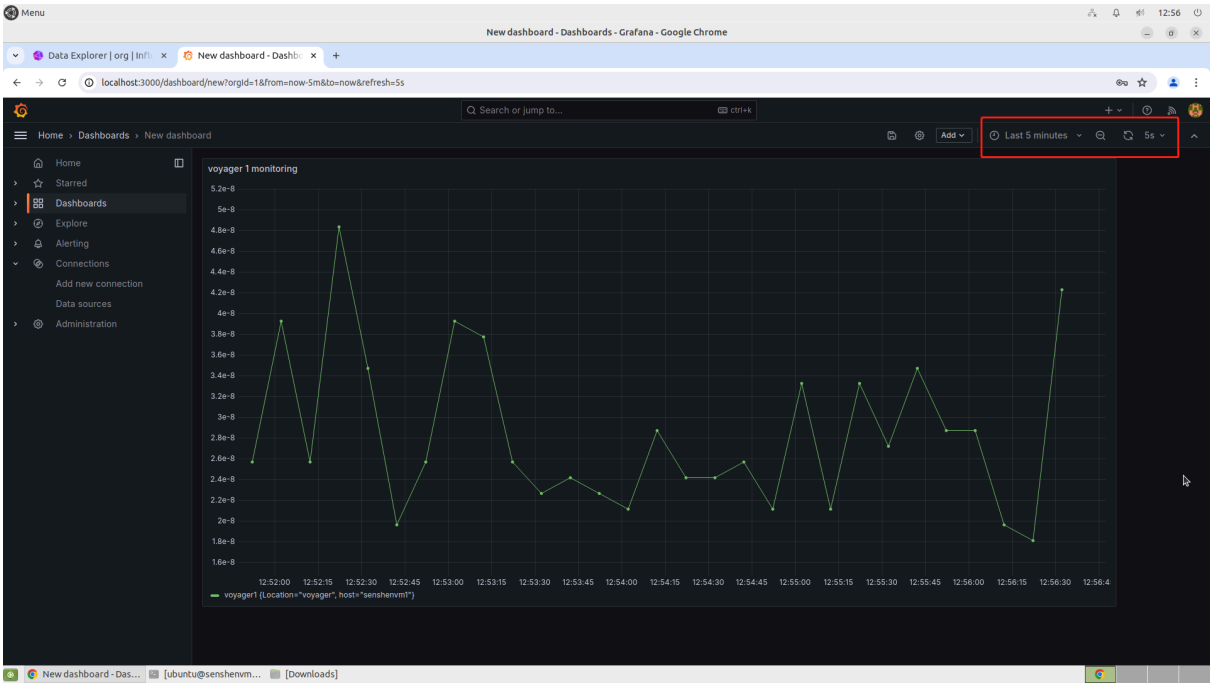


In the InfluxDB interface, select the bucket, the measurement and the field you want, and click SCRIPT EDITOR. Then, copy these codes into the Grafana dashboard, apply them and you will see the results. You can set refresh frequency so that the monitoring interface will be refreshed automatically.




Data Explorer | org | InfluxDB - Google Chrome
localhost:8086/orgs/b8165dac9a740368/data-explorer?fluxScriptEditor
Data Explorer
Graph
CUSTOMIZE
Local
SAVE AS
Query 1 (0.04s)
View Raw Data
CSV
Past 1m
QUERY BUILDER
SUBMIT
from(bucket: "Node A")
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "voyager_BER")
|> filter(fn: (r) => r["Location"] == "voyager")
|> filter(fn: (r) => r["_field"] == "voyager1")
|> filter(fn: (r) => r["host"] == "senshenvm1")
|> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
|> yield(name: "mean")
Transformations
aggregate.rate
chandeMomentumOscillator
columns
cov
covariance
cumulativeSum
date.hour
date.microsecond


Edit panel - New dashboard - Dashboards - Grafana - Google Chrome
localhost:3000/dashboard/new?orgId=1&editPanel=1
Home › Dashboards › New dashboard › Edit panel
Discard
Save
Apply
voyager 1 monitoring
No data
Query
Transform data
Alert
Data source
VM1
Query options
Query inspector
Time series
Panel options
Title
voyager 1 monitoring
Description
Tooltip
Legend



2.2.4 Telegraf: 🔗InfluxData Downloads

Select: Version: v1.31.0 Platform: Ubuntu & Debian

Telegraf is responsible for collecting data from various sources and sending the data to InfluxDB automatically.

```
# To download Telegraf
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6a
c9ce4c influxdata-archive_compat.key' | sha256sum -c && cat influxdata-a
rchive_compat.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influx
data-archive_compat.gpg > /dev/null
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gp
g] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sour
ces.list.d/influxdata.list

sudo apt-get update && sudo apt-get install telegraf
```

Generally, Telegraf will be installed in the folder **/etc**. Go to **/etc/telegraf**, you will find a file named **telegraf.conf**. Then, you can modify it to add different input resources and output targets. The two examples are as follow:

1. Collect data from devices and send it to InfluxDB 2.0

Here, we have a python script using REST API to collect data from voyager transponders. So for the input resource, we set it as the python script.

```
[[inputs.exec]]
  commands = ["python3 /home/ubuntu/Downloads/voyager.py"]
  timeout = "30s"
  data_format = "influx"
  interval = "10s"

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "your All-access API token generated from InfluxDB"
  organization = "your organisation, i.e, University of Bristol"
  bucket = "your bucket, i.e, Lab"
```

2. Read metrics from Kafka topics and send them to InfluxDB 2.0

```
[[inputs.kafka_consumer]]
  ## Kafka brokers
  brokers = ["localhost:9092"]

  ## Topics to consume
  topics = ["EDFA, Voyager"]

  ## Data format to consume
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMA
TS_INPUT.md
  data_format = "influx

  [[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["http://localhost:8086"]
```

```
## Token for authentication.
token = "your All-access API token generated from InfluxDB"

## Organization is the name of the organization you wish to write to.
organization = "your organisation, i.e, University of Bristol"

## Destination bucket to write into.
bucket = "your bucket, i.e, Lab"
```

- **EDFA, Voyager:** you can modify topics according to your needs. In our implementation, we use two topics, EDFA and Voyager, to store the respective data retrieved from EDFAs and Voyagers on various nodes.

Once configured the **telegraf.conf**, open the terminal and run the telegraf service.

```
# Start the telegraf service and set it to startup
sudo systemctl enable telegraf
sudo systemctl start telegraf

# Telegraf will be running according to your configuration
cd /etc/telegraf
telegraf --config telegraf.conf
```

2.3 Software Installation and Configuration on the Cloud Server

On the cloud server, it also needs to install Anaconda, InfluxDB, Telegraf and Grafana. The download and installation processes for Anaconda, InfluxDB, Telegraf and Grafana are identical to those we did on the local servers.

However, different from the local setup on the local servers, we specifically need **Zookeeper** and **Apache Kafka** to establish Kafka-based telemetry pipelines on the cloud server. Additionally, configuration files for Telegraf, Zookeeper, and Kafka will require adjustments. As mentioned in **2.2.4 Example 2: Read metrics from Kafka topics and send them to InfluxDB 2.0**, the modifications to the **telegraf.conf** involve configuring inputs for Kafka consumers and outputs for InfluxDB.

The following sections outline the installation and configuration processes for Zookeeper and Apache Kafka on the cloud server.

2.3.1 Zookeeper: 🔗Apache Zookeeper

1. Both Zookeeper and Apache Kafka require a JDK environment. Therefore, it is essential to verify whether Java is installed before starting them.

```
sudo apt install openjdk-11-jdk -y
```

- Verify after installation is complete:

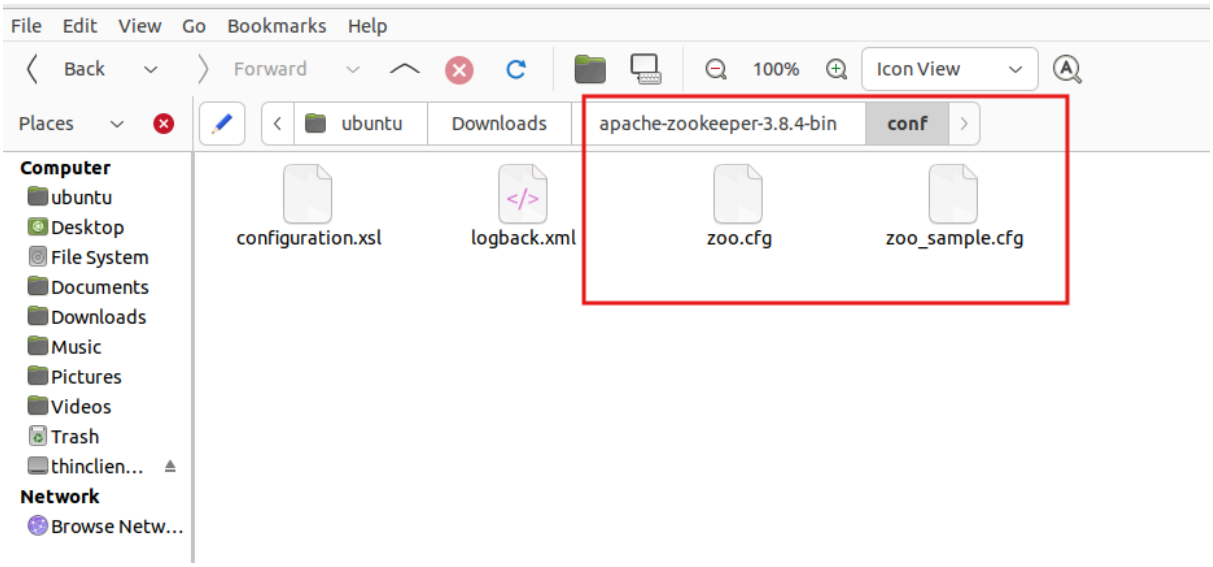
```
Java --version
```

2. Once Java installation is confirmed on the cloud server, we can proceed to download Zookeeper. For our implementation, we use the latest stable version 3.8.4 as an example:

```
wget https://downloads.apache.org/zookeeper/zookeeper-3.8.4/apache-zookeeper-3.8.4-bin.tar.gz
```

```
tar -xzf apache-zookeeper-3.8.4-bin.tar.gz
```

3. Then go to the conf directory under the zookeeper installation directory and make a copy of **zoo_example.cfg**. It is recommended that you name the copied file **zoo.cfg** as the Zookeeper server is started by default through the zoo.cfg configuration file, eliminating the need to explicitly specify the configuration file name when starting the server.



4. Create a new folder named **Data** in Zookeeper directory and modify the configuration file **zoo.cfg** as:

```
#Create the dataDir directory where the snapshot is stored.

dataDir=/home/ubuntu/Downloads/zookeeper3.8.4/Data
```

- **Data:** Create a new folder named **Data** under kafka directory to store snapshot. You can adjust the path to this folder based on the location of your Kafka installation.

5. Once configured the **zoo.cfg**, open the terminal and run the Zookeeper instances.

```
#To launch Zookeeper
bin/zkServer.sh start
```

2.3.2 Kafka: 🔗Apache Kafka

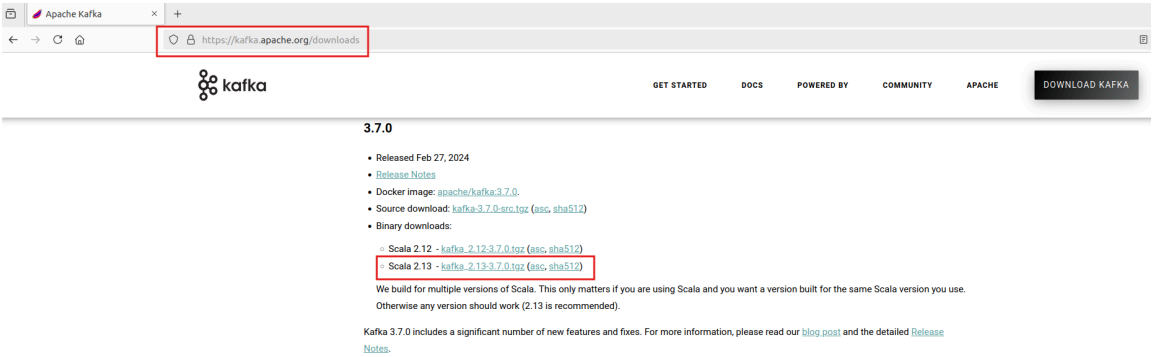
To download Apache Kafka, follow these steps:

1. Visit the Apache Kafka Download Page.
2. Select a Kafka Version you want to download.

- In this implementation, we are using **kafka_2.13-3.7.0,** which is the latest stable version available.

3. Choose the Binary downloads.

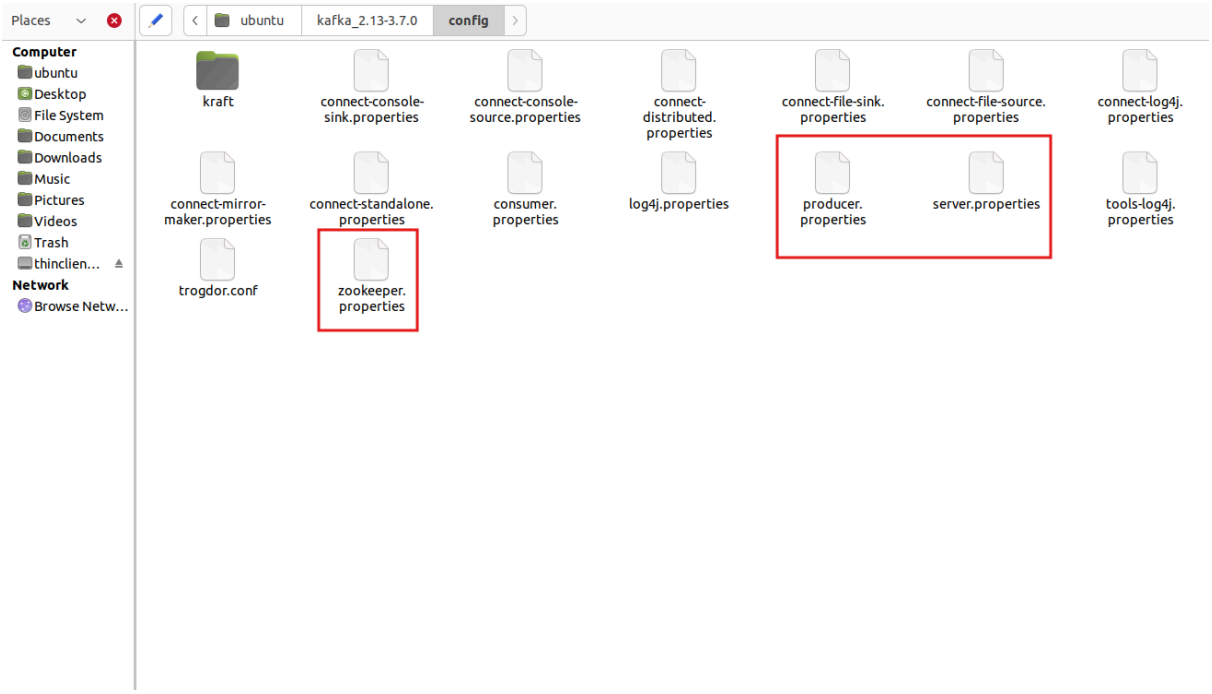
4. Click and download the package.



5. Extract the downloaded **kafka_2.13-3.7.0.tgz** file.

```
tar -xzf kafka_2.13-2.7.0.tgz
```

For Kafka Brokers on the cloud server to receive connections and messages from both the Kafka Producers on the local servers, you need to make the following settings in configuration files **producer.properties, server.properties** and **zookeeper.properties** on the cloud server:



Producer.properties:

```
#Modifications in producer.properties

bootstrap.servers=localhost:9092
```

Server.properties:

```
#Modifications in server.properties

listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://server-ip:9092

#Create the folder by yourself under the Kafka directory.
log.dirs=/home/ubuntu/kafka_2.13-3.7.0/kafka-logs

#Zookeeper is crucial for managing and coordinating Kafka topics, partition
s,
#and broker configurations across the Kafka cluster.
zookeeper.connect=localhost:2181
```

- **server-ip:** can be changed to your own ip address of server.
- Create a new folder named **kafka-logs** under kafka directory to store log files. You can adjust the path to this folder based on the location of your Kafka installation.

Zookeeper.properties:

```
dataDir=/home/ubuntu/kafka_2.13-3.7.0/zookeeper_data
```

- Create a new folder named **zookeeper_data** under kafka directory to store snapshot. You can adjust the path to this folder based on the location of your Kafka installation.

Once configured all the properties files, open the terminal and start the Kafka service.

```
#To launch kafka
bin/kafka-server-start.sh config/server.properties
```

Problems you may encounter:

1. Zookeeper must be started before Kafka, otherwise errors will occur.
2. Since Kafka is downloaded via its official website, it will be saved under **/home/ubuntu/Downloads** by default. It is better to reduce the length of Kafka's path such as moving the **kafka_2.13-2.7.0.tgz** file under the home directory. In my case, I put it under **/home/ubuntu**.

3. Local Telemetry Service

Here, let's make an example of the local telemetry service between the voyager transponders in our lab and the local servers. For the voyager transponders, we developed a REST API-based python script to collect data from them and use the script as the telegraf input.

```
[[inputs.exec]]
  commands = ["python3 /home/ubuntu/Downloads/voyager.py"]
  timeout = "30s"
  data_format = "influx"
  interval = "10s"

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "your All-access API token generated from InfluxDB"
  organization = "your organisation, i.e, University of Bristol"
  bucket = "your bucket, i.e, Lab"
```

Then, start the telegraf service. The data will be collected and written into InfluxDB automatically.

```
cd /etc/telegraf
telegraf --config telegraf.conf
```

4. Kafka-based Telemetry Pipeline

1. To stream data from local servers to the cloud server through Kafka-based telemetry pipeline, we need to send data to Kafka clusters on the cloud server using Python Kafka Producer, which is a Python script.

```
import influxdb_client
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS
from kafka import KafkaProducer
import json
import time

# Connect to InfluxDB
bucket = "your bucket, i.e, Lab"
org = "your organisation, i.e, University of Bristol"
token = "your All-access API token generated from InfluxDB"
url = "http://local-server-ip:8086"


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

# Construct the query
query = '''
from(bucket: "your bucket, i.e, Lab")
  |> range(start: -15s)
  |> filter(fn: (r) => r._measurement == "voyager_BER")
  |> filter(fn: (r) => r._field == "voyager1" or r._field == "voyager2" or r._field
== "voyager3" or r._field == "voyager4" or r._field == "voyager5" or r._field
== "voyager6" or r._field == "voyager7" or r._field == "voyager8")
'''

query_api = client.query_api()

tables = query_api.query(query, org=org)

data_list = []
for table in tables:
    for record in table.records:
        data_list.append(record.values['_value'])  # Append directly to data_lis
```

```
t


def extract_data():
    line_protocol = (
        "voyager_BER,Location=voyager "
        "voyager1={},voyager2={},voyager3={},voyager4={},voyager5={},voy
ager6={},voyager7={},voyager8={}"
    ).format(data_list[0], data_list[1], data_list[2], data_list[3], data_list[4], dat
a_list[5], data_list[6], data_list[7])

    return line_protocol


def main():
    producer = KafkaProducer(bootstrap_servers=['cloud-server-ip:9092'])

    for _ in range(20000):
        data = extract_data()
        producer.send('voyager', bytes(f'{data}','UTF-8'))
        print(f"voyager data is sent: {data}")
        time.sleep(15)


if __name__ == "__main__":
    main()
```

2. Then, start the Zookeeper and Apache Kafka on the cloud server.

```
cd /home/ubuntu/kafka_2.13-3.7.0

bin/zookeeper-server-start.sh config/zookeeper.properties

bin/kafka-server-start.sh config/server.properties
```

3. Next, start the telegraf service. And you can see the streamed data in InfluxDB on the cloud server.

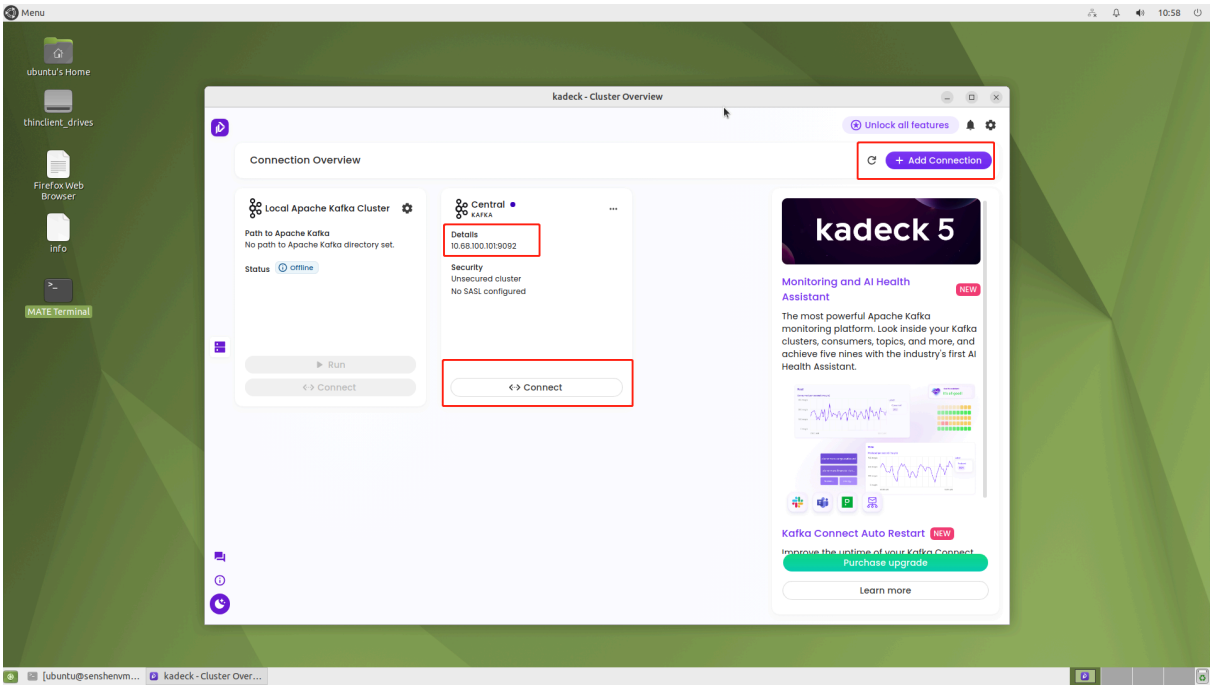
```
[[inputs.kafka_consumer]]
  ## Kafka brokers
  brokers = ["cloud-server-ip:9092"]

  ## Topics to consume
  topics = ["Voyager"]

  ## Data format to consume
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMA
TS_INPUT.md
  data_format = "influx

  [[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["http://cloud-server-ip:8086"]

  ## Token for authentication.
  token = "your All-access API token generated from InfluxDB"

  ## Organization is the name of the organization you wish to write to.
  organization = "your organisation, i.e, University of Bristol"

  ## Destination bucket to write into.
  bucket = "your bucket, i.e, Lab"
```

```
cd /etc/telegraf
telegraf --config telegraf.conf
```

Furthermore, you can install Kadeck to visualize data streaming through Kafka-based telemetry pipeline.

- Make sure that you have OpenJDK installed before installing Kadeck since Kadeck requires a Java environment to run.
- Visit Kadeck's official website to download the latest version of Kadeck. Choose the version for Linux. 🔗Kadeck Download
- After downloading, unzip the file.
- Go into the unzipped directory and run Kadeck. Then, you can connect it with your Kafka server.

```
cd ~/Downloads
tar -xzf kadeck-<version>.tar.gz

cd kadeck-<version>
./kadeck
```



5. Use Case

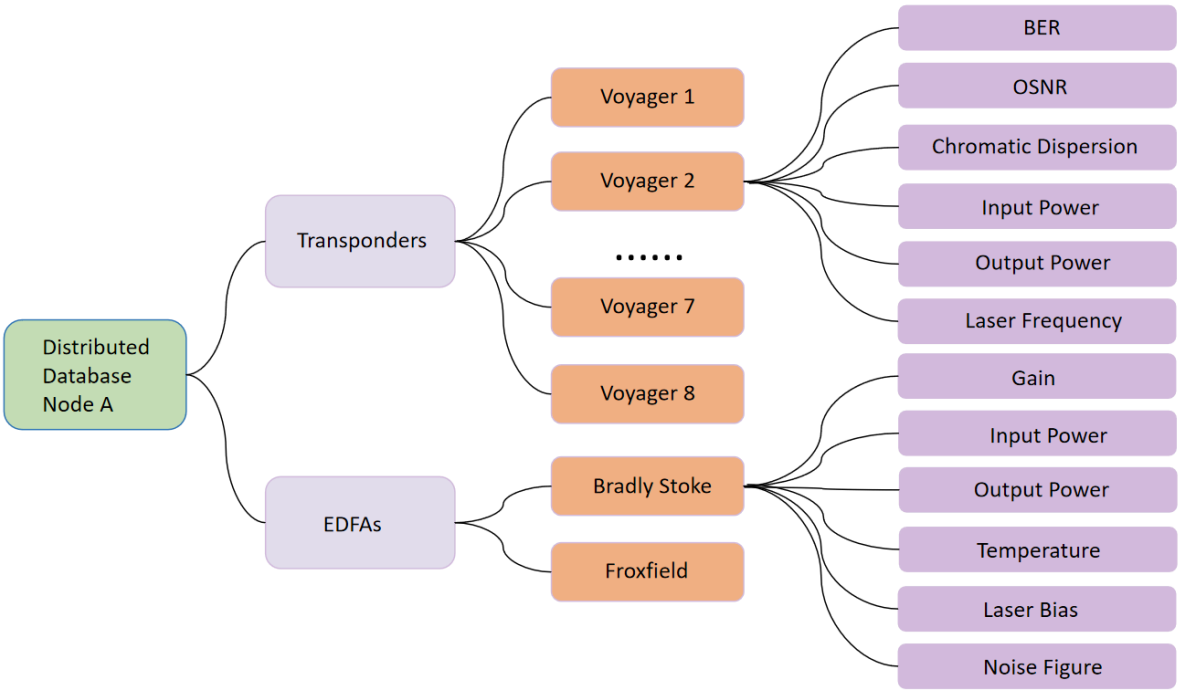
This section briefly outlines two use cases of this unified platform on the testbed: retrieving end-to-end data from the cloud and providing streaming data for ML applications.

In these use cases, we demonstrate the proposed architecture on a field-trial testbed over the part of the UK NDFF through Bradley Stoke, Froxfield and

Reading. In the experimental setup, eight optical signals are generated by two Facebook Voyager Transponders in our lab. These signals are then transmitted from Node A (Bristol) and looped back to Node B (Bristol) via the NDFF link.

5.1 Retrieving End-to-End Data

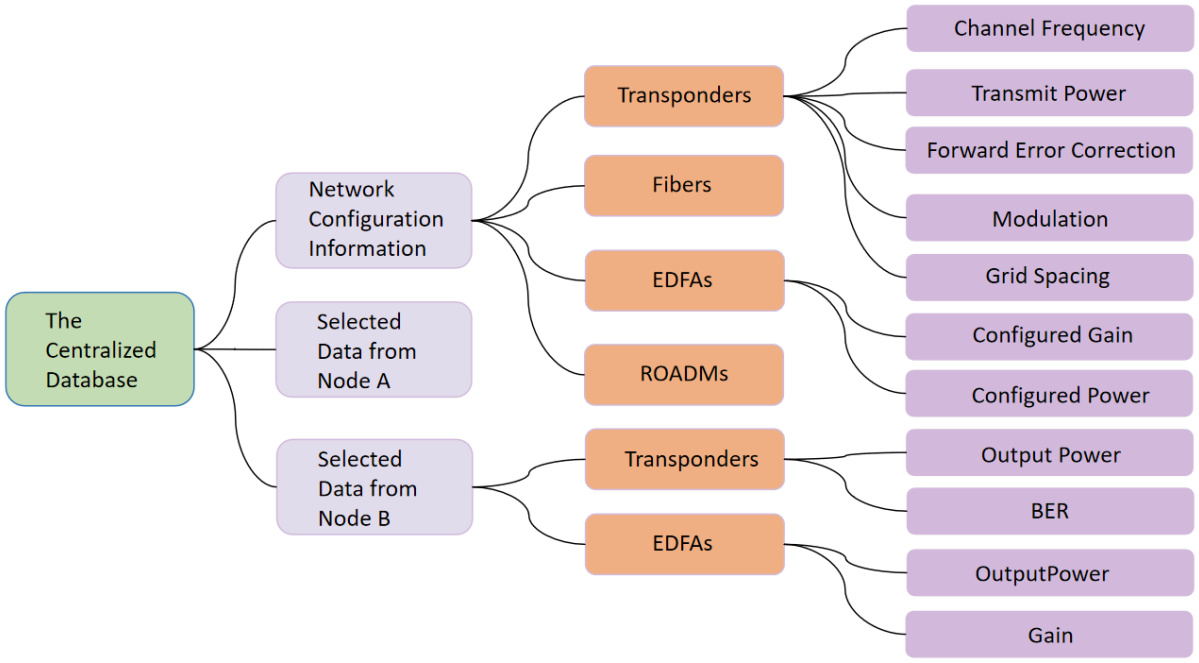
As shown in the figure below, performance monitoring data from Voyagers and EDFAs on Node A is stored in the distributed database. This distributed database on Node A is responsible for storing Voyagers' performance metrics before signal transmission including Bit Error rate (BER), Optical Signal-to-Noise Ratio (OSNR), Chromatic Dispersion (CD), input power, output power, laser frequency, and EDFA parameters at nodes Bradley Stoke and Froxfield including gain, input power, output power, temperature, laser bias and noise figure. Meanwhile, the distributed database on Node B is responsible for storing Voyagers' performance metrics after transmission, as well as the EDFA parameters at node Reading.



Distributed database design on Node A

Unlike distributed databases, the centralized database shown below stores network configuration information, including channel frequency, transmit power, forward error correction, modulation, and grid spacing from Voyagers, along with EDFA gain and output power (Automatic Gain Control and Automatic Power Control). Moreover, in order to support network-level ML applications

such as end-to-end QoT prediction, performance data like Voyagers' BER values and output power from two nodes are also stored in the centralized database.

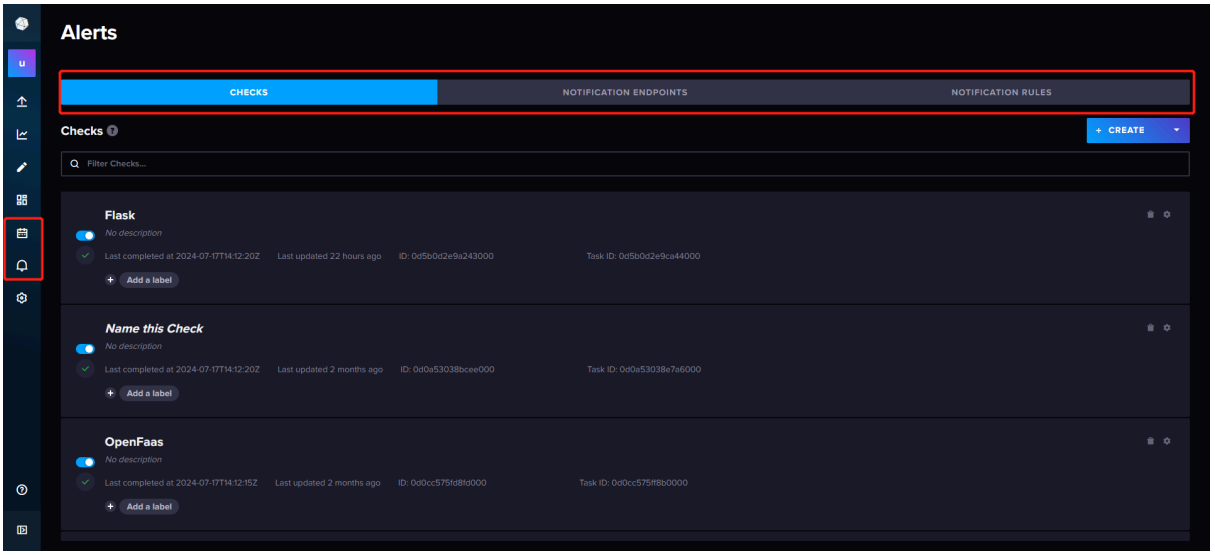


Centralised database design

5.2 Streaming data for network functions (ML services)

InfluxDB provides **Tasks**, **Alerts**, and **Notifications** to automatically and continuously push data to network functions. Here, we use Alerts and Notifications to stream data to our AI engine deployed on OpenFaas as an example.

1. **Log into the InfluxDB 2.0 UI, and navigate to the Alerts & Notifications section.**



2. **Setting up the CHECKS**

1) Click "Create Check".

2) Select the alert type (e.g. "Threshold Check").

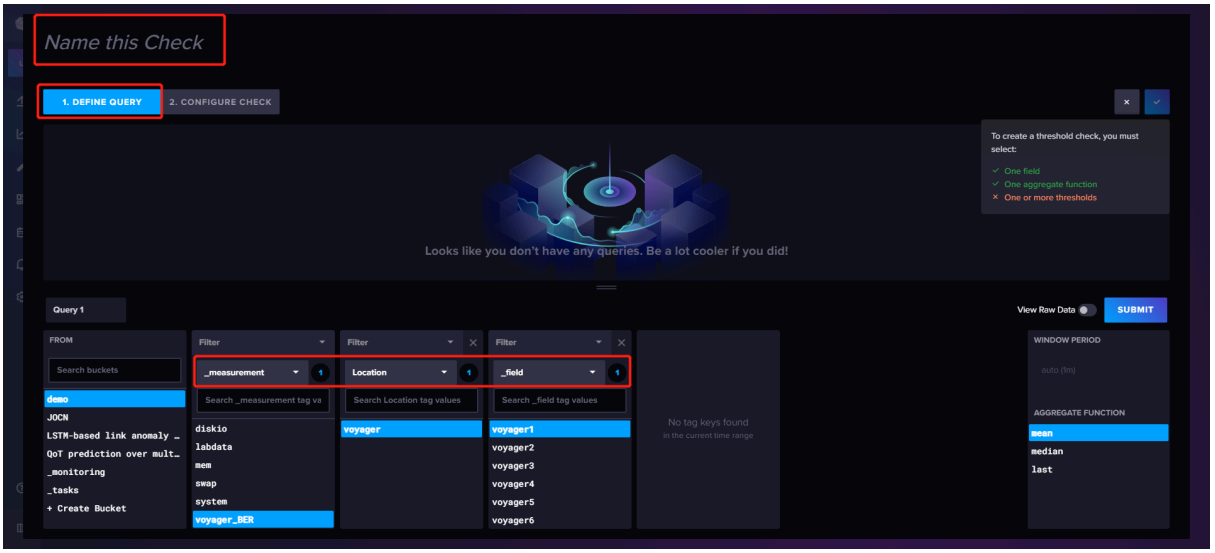
3) Select the Bucket, Measurement, Location and Field.

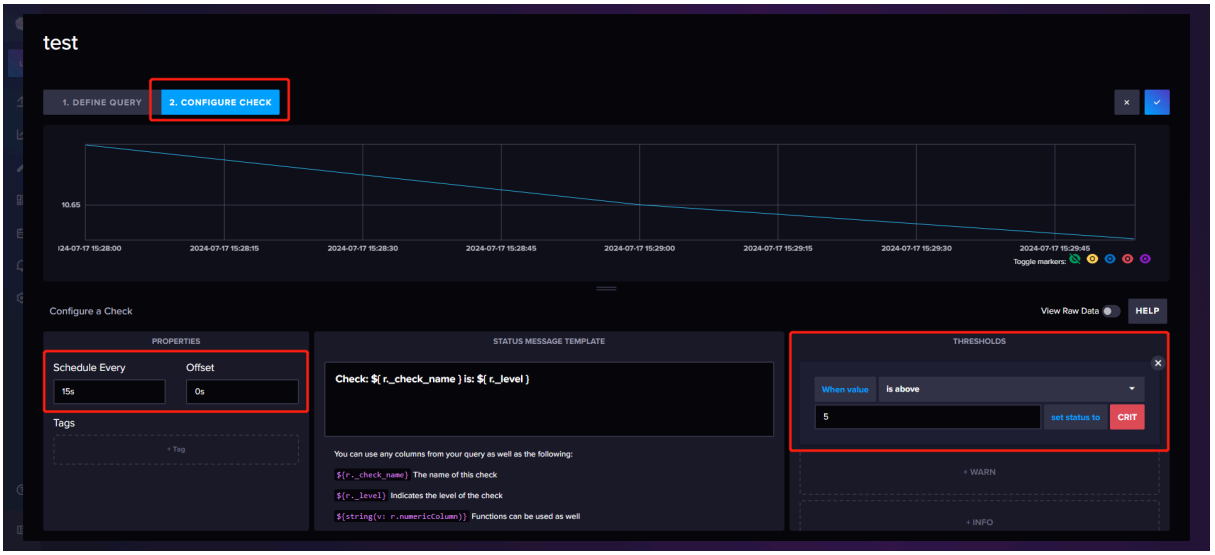
4) Enter the alert name and description.

5) Select the checking frequency.

6) Configure the alert conditions (e.g. when a certain measurement exceeds a certain threshold).

7) Save the configuration.





3. **Setting up NOTIFICATION ENDPOINTS**

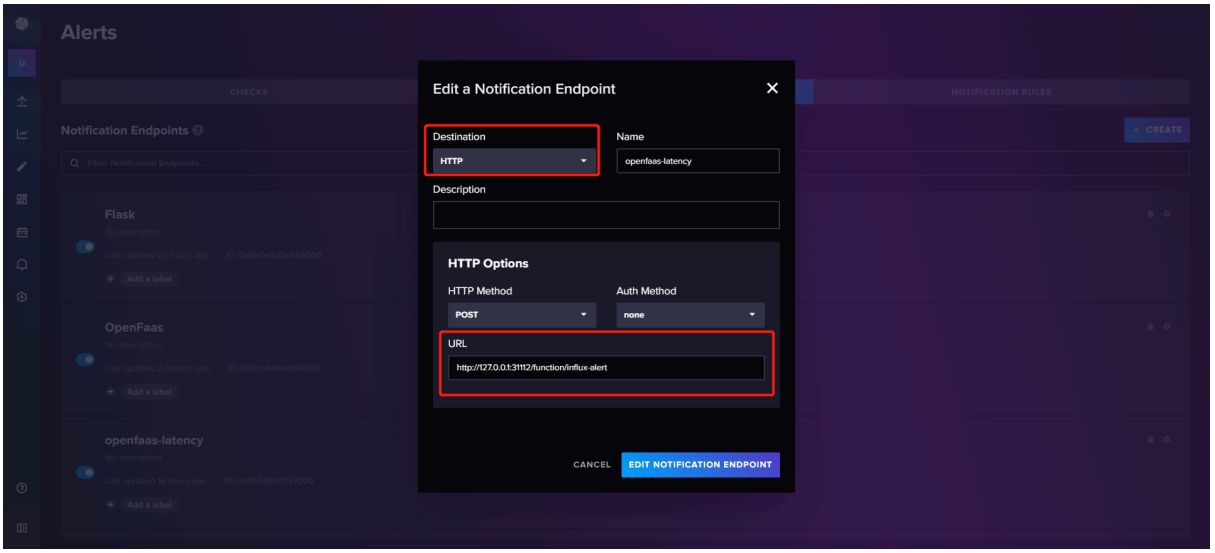
1) Click Create Notification Endpoint.

2) Select HTTP as the type.

3) Enter the endpoint name and the URL of your OpenFaaS function (for example: http://<openfaas_gateway>/function/<function_name>).

4) Configure authentication if required.

5) Save the configuration.



4. **Setting up NOTIFICATION RULES**

1) Click Create Notification Rule.

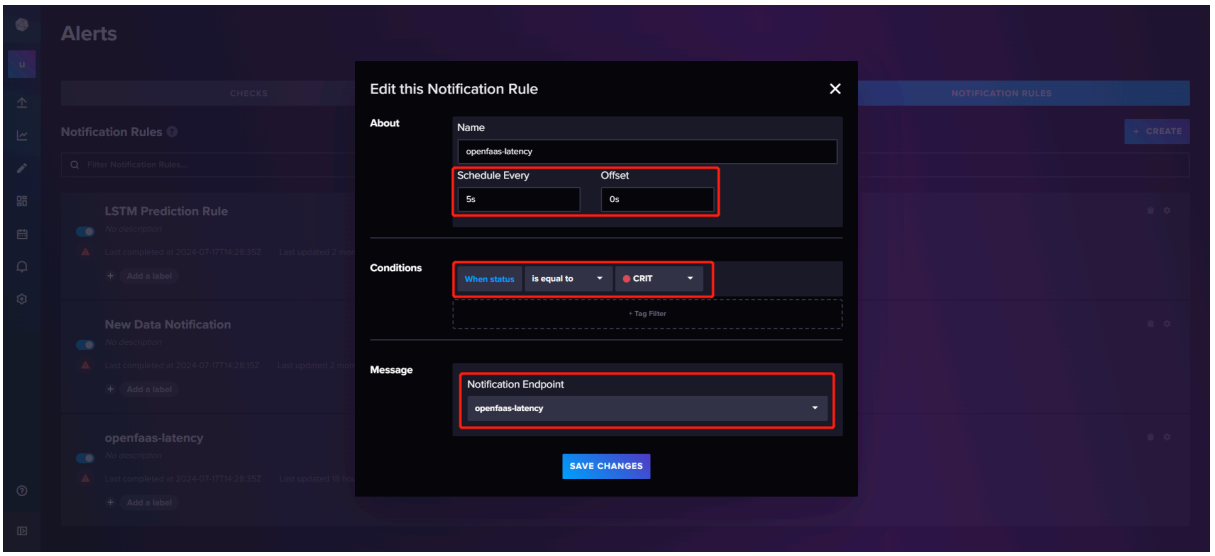
2) Enter a rule name.

3) Select a previously created notification endpoint.

4) Configure the rule conditions (for example, when a certain threshold is exceeded).

5) Set the notification content and format.

6) Save the configuration.



5. **Trigger and test**

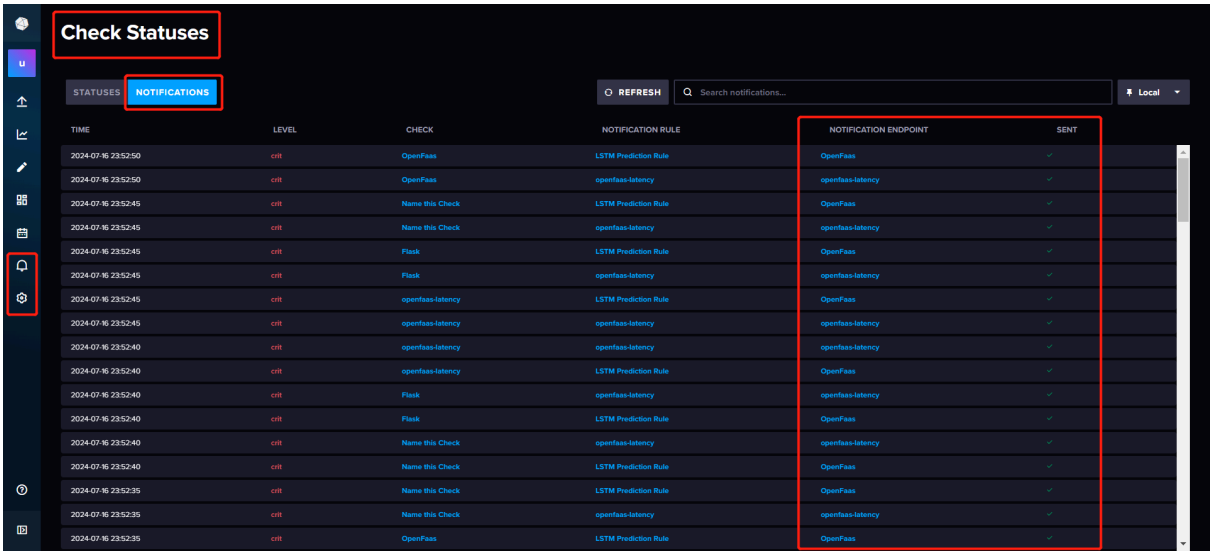
After completing the above configuration, you can test the entire process by triggering related conditions.

1) Trigger alert conditions:

Simulate or actually generate data to meet the configured alert conditions.

Confirm that the notification rules in InfluxDB are triggered.

You can check Alert History in InfluxDB 2.0.



2) Verify OpenFaaS function execution:

View the execution log of the function in OpenFaaS to confirm that the received data is correct.
Verify that the function processes the data as expected.

With these steps, you can configure Alerts and Notifications in the InfluxDB 2.0 user interface and continuously send data to the OpenFaaS function to achieve automated data processing and response.

```
jason@IT079516: ~
2024-05-28T10:34:50Z 2024/05/28 10:34:50 POST / - 200 - ContentLength: 702B (0.0163s)
2024-05-28T10:34:50Z 2024/05/28 10:34:50 Started logging: stderr from function.
2024-05-28T10:34:50Z 2024/05/28 10:34:50 POST / - 200 - ContentLength: 699B (0.0168s)
2024-05-28T10:34:55Z 2024/05/28 10:34:55 Started logging: stderr from function.
2024-05-28T10:34:55Z 2024/05/28 10:34:55 POST / - 200 - ContentLength: 703B (0.0176s)
2024-05-28T10:35:00Z 2024/05/28 10:35:00 Started logging: stderr from function.
2024-05-28T10:35:00Z 2024/05/28 10:35:00 POST / - 200 - ContentLength: 703B (0.0171s)
2024-05-28T10:35:05Z 2024/05/28 10:35:05 Started logging: stderr from function.
2024-05-28T10:35:05Z 2024/05/28 10:35:05 POST / - 200 - ContentLength: 703B (0.0188s)
2024-05-28T10:35:05Z 2024/05/28 10:35:05 Started logging: stderr from function.
2024-05-28T10:35:05Z 2024/05/28 10:35:05 POST / - 200 - ContentLength: 689B (0.0168s)
2024-05-28T10:35:10Z 2024/05/28 10:35:10 Started logging: stderr from function.
2024-05-28T10:35:10Z 2024/05/28 10:35:10 POST / - 200 - ContentLength: 703B (0.0182s)
2024-05-28T10:35:15Z 2024/05/28 10:35:15 Started logging: stderr from function.
2024-05-28T10:35:15Z 2024/05/28 10:35:15 POST / - 200 - ContentLength: 703B (0.0165s)
2024-05-28T10:35:20Z 2024/05/28 10:35:20 Started logging: stderr from function.
2024-05-28T10:35:20Z 2024/05/28 10:35:20 POST / - 200 - ContentLength: 703B (0.0166s)
2024-05-28T10:35:20Z 2024/05/28 10:35:20 Started logging: stderr from function.
2024-05-28T10:35:20Z 2024/05/28 10:35:20 POST / - 200 - ContentLength: 699B (0.0173s)
2024-05-28T10:35:25Z 2024/05/28 10:35:25 Started logging: stderr from function.
2024-05-28T10:35:25Z 2024/05/28 10:35:25 POST / - 200 - ContentLength: 703B (0.0168s)
2024-05-28T10:35:30Z 2024/05/28 10:35:30 Started logging: stderr from function.
2024-05-28T10:35:30Z 2024/05/28 10:35:30 POST / - 200 - ContentLength: 703B (0.0179s)
2024-05-28T10:35:35Z 2024/05/28 10:35:35 Started logging: stderr from function.
2024-05-28T10:35:35Z 2024/05/28 10:35:35 POST / - 200 - ContentLength: 703B (0.0181s)
2024-05-28T10:35:35Z 2024/05/28 10:35:35 Started logging: stderr from function.
2024-05-28T10:35:35Z 2024/05/28 10:35:35 POST / - 200 - ContentLength: 699B (0.0174s)
2024-05-28T10:35:40Z 2024/05/28 10:35:40 Started logging: stderr from function.
2024-05-28T10:35:40Z 2024/05/28 10:35:40 POST / - 200 - ContentLength: 703B (0.0192s)
```

6. About Us

This work is leaded by Dr. Shuangyi, and supported by Sen Shen and Jing Han. If you have any questions, please feel free to contact us 👇

Dr Shuangyi Yan, E: shuangyi.yan@bristol.ac.uk

Mr Sen Shen, E: sen.shen@bristol.ac.uk

Miss Jing Han, E: ia19023@bristol.ac.uk