SMART PARKING IoT PROJECT

By Ali Haider & Giacomo Pedemonte

TABLE OF CONTENTS

01

SMART PARKING

Introduction

02

WOKWI

Simulation

03

TELEGRAF

Collecting and storing metrics

04

InfluxDB

Storing Time-Series data

05

GRAFANA

Visualization & Alerting

06

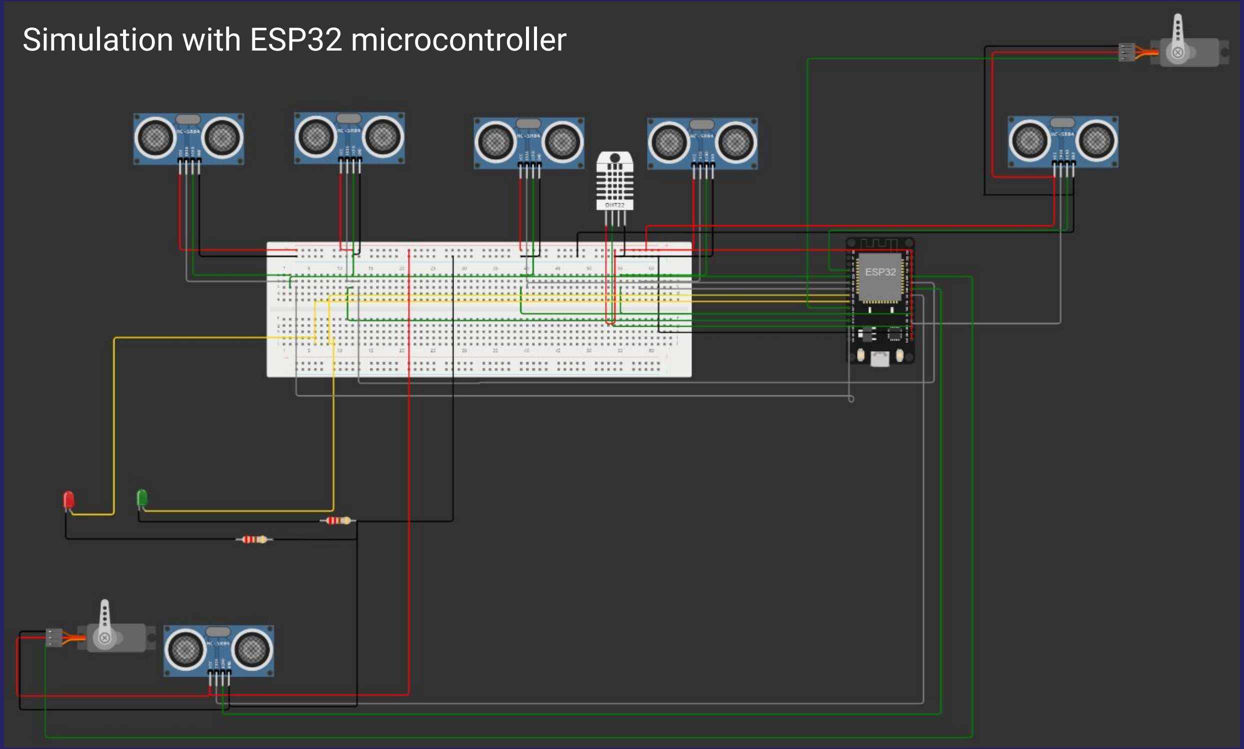
CONCLUSION

And challenging parts

SMART PARKING

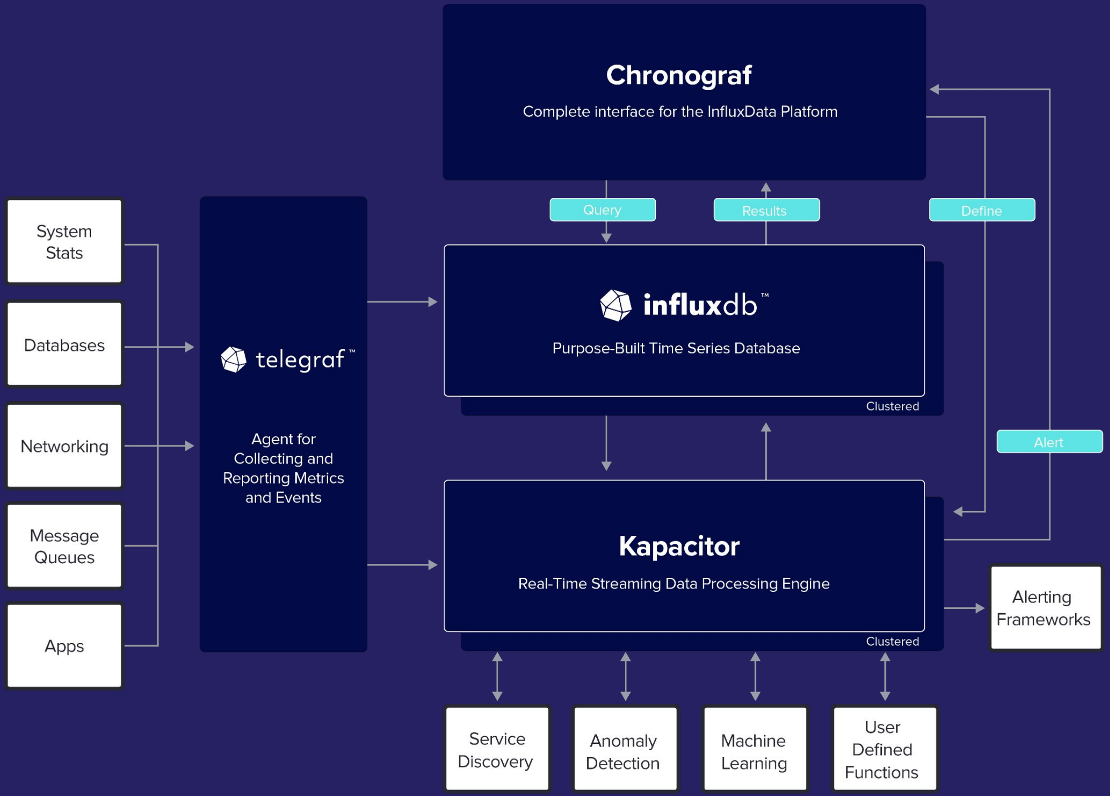
- Entrances & Exits Monitoring
- Plate Recognition
- Monitoring Slots - Occupation & Environment

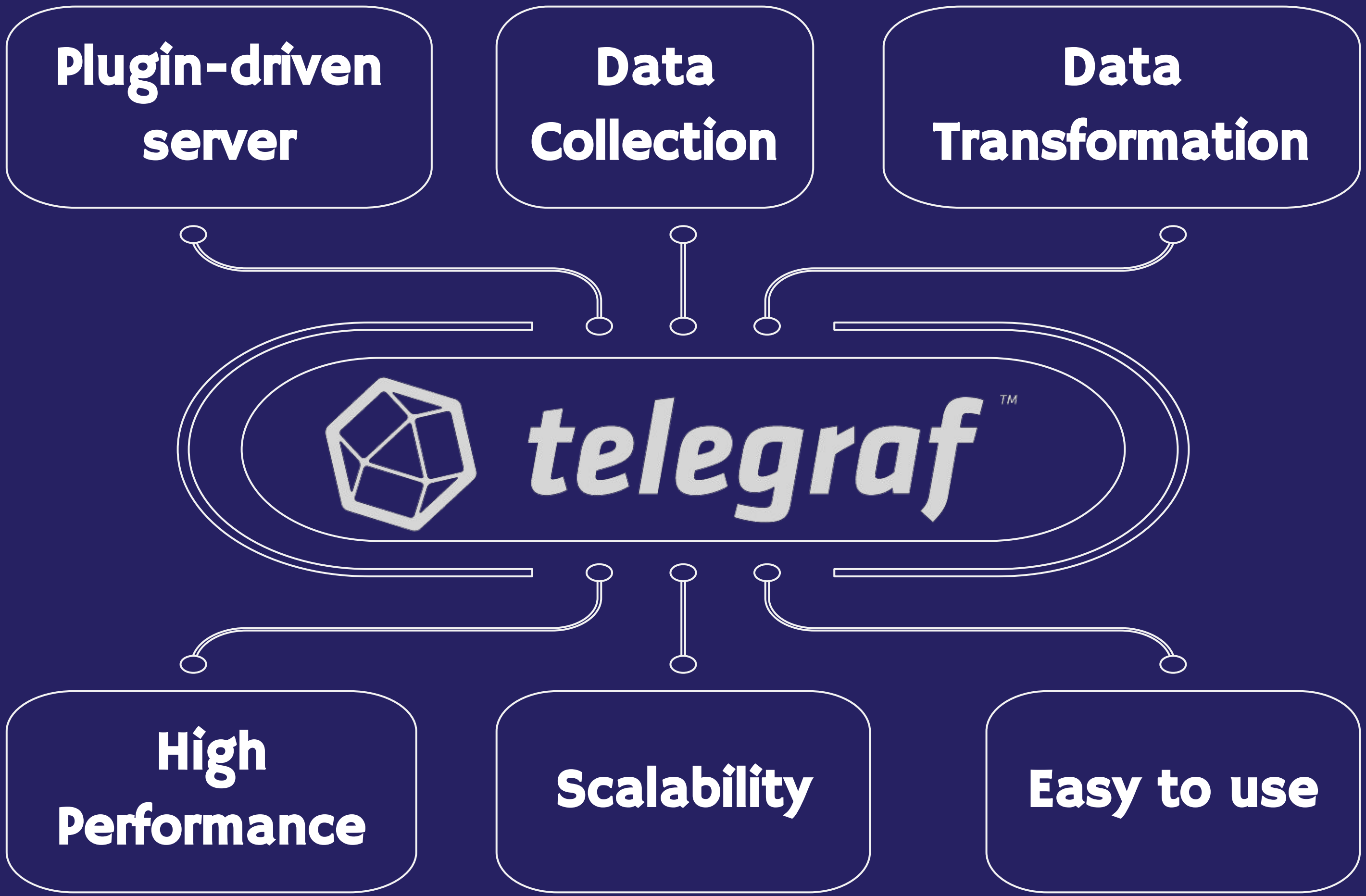
WOKWI



Simulation with ESP32 microcontroller

TICK Stack


Chronograf
Complete interface for the InfluxData Platform
Query
Results
Define
System Stats
Databases
Networking
Message Queues
Apps
telegraf™
Agent for Collecting and Reporting Metrics and Events
influxdb™
Purpose-Built Time Series Database
Clustered
Alert
Kapacitor
Real-Time Streaming Data Processing Engine
Clustered
Alerting Frameworks
Service Discovery
Anomaly Detection
Machine Learning
User Defined Functions



telegraf™

Plugin-driven server

Data Collection

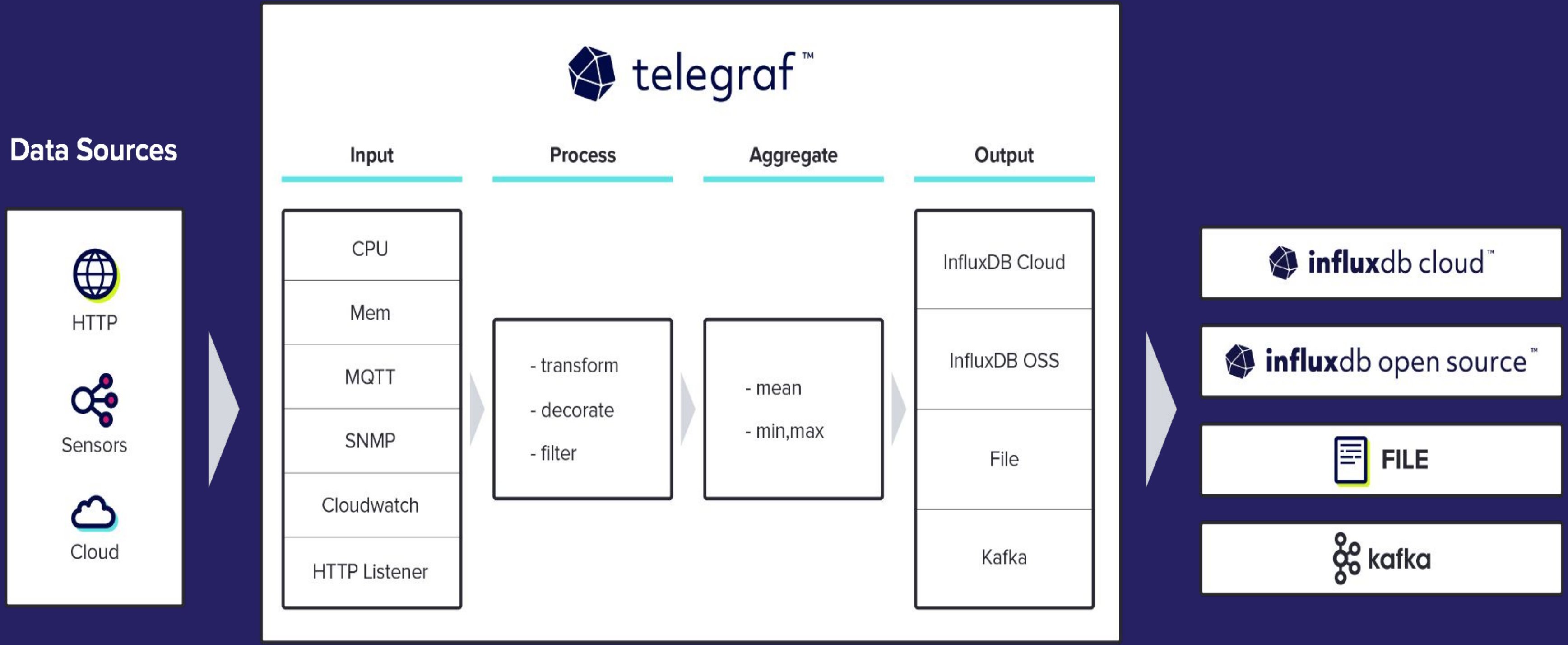
Data Transformation

High Performance

Scalability

Easy to use

TELEGRAF



Data Sources

HTTP

Sensors

Cloud

telegraf™

Input

- CPU
- Mem
- MQTT
- SNMP
- Cloudwatch
- HTTP Listener

Process

- transform
- decorate
- filter

Aggregate

- mean
- min,max

Output

- InfluxDB Cloud
- InfluxDB OSS
- File
- Kafka

influxdb cloud™

influxdb open source™

FILE

kafka

InfluxDB

- High performance Time Series Database
- It can handle millions of data points per second
- Helpful for DevOps monitoring, IoT monitoring, and real-time analytics

InfluxDB - How it works

BUCKETS

Buckets can be seen as databases where measurements will be stored

MEASUREMENT

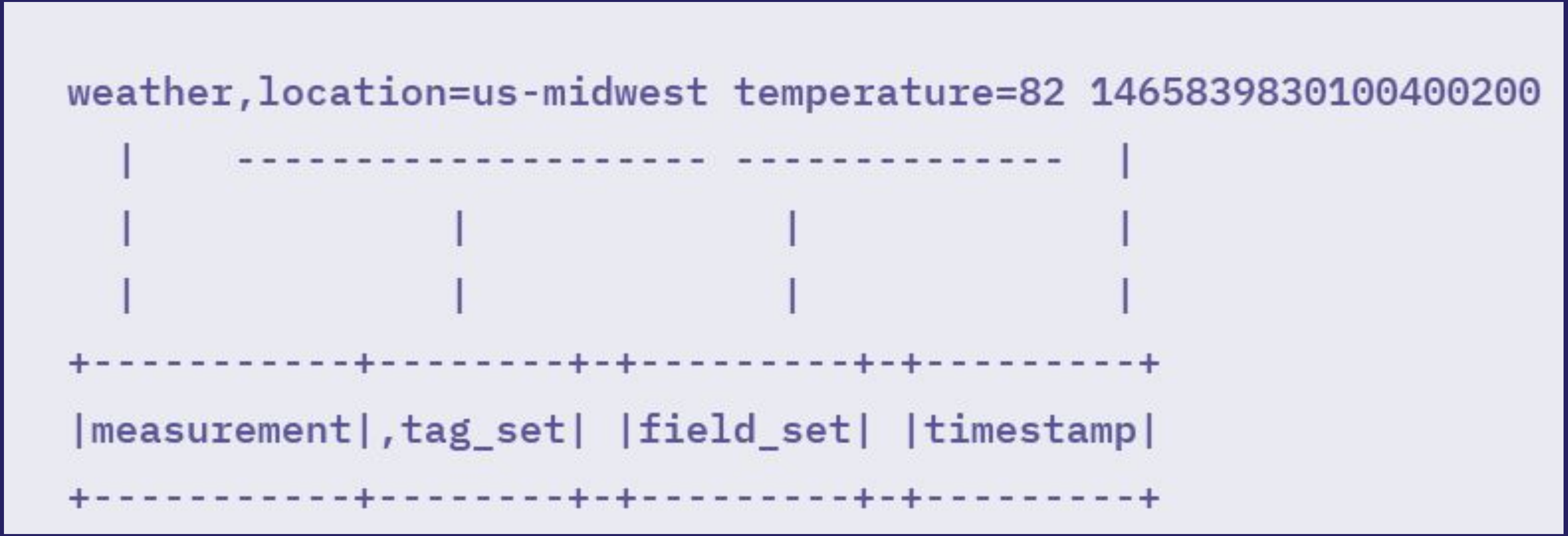
This will represent what we are measuring.
Could represent a table inside a Bucket.
Can be accessed with SQL or with InfluxQL

LineProtocol

InfluxDB stores data into Buckets using LP(LineProtocol) to represent data as time series

InfluxDB - LP example

This is an example of how InfluxDb save data with the Line Protocol format:

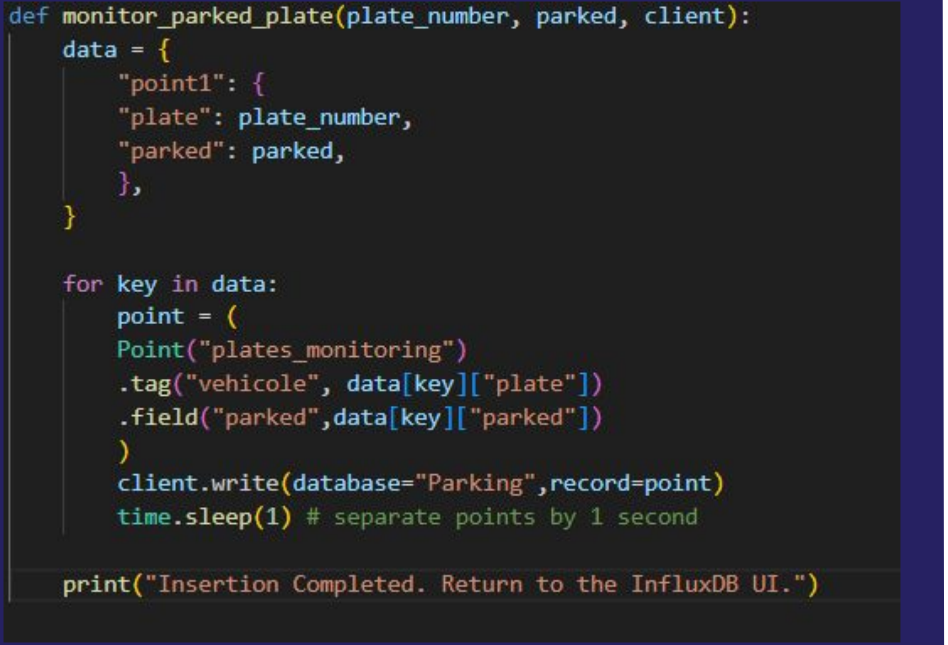


```
weather,location=us-midwest temperature=82 1465839830100400200
  |    -------------------- --------------  |
  |             |             |             |
  |             |             |             |
+-----------+--------+-+---------+-+---------+
|measurement|,tag_set| |field_set| |timestamp|
+-----------+--------+-+---------+-+---------+
```

And this is more simple that what it seems...

InfluxDB - Collecting Data

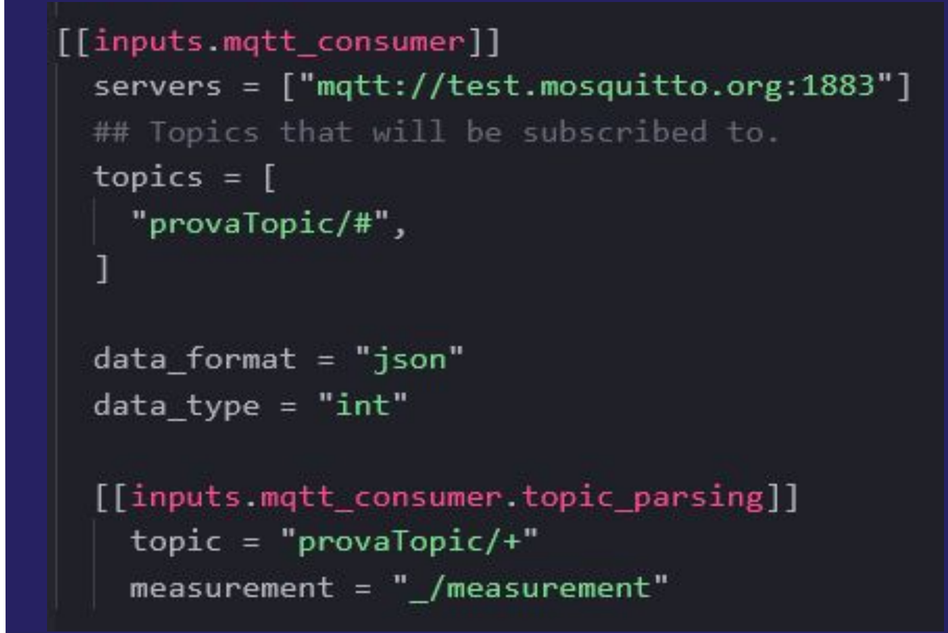
Python



```python
def monitor_parked_plate(plate_number, parked, client):
    data = {
        "point1": {
        "plate": plate_number,
        "parked": parked,
        },
    }

    for key in data:
        point = (
        Point("plates_monitoring")
        .tag("vehicole", data[key]["plate"])
        .field("parked",data[key]["parked"])
        )
        client.write(database="Parking",record=point)
        time.sleep(1) # separate points by 1 second

    print("Insertion Completed. Return to the InfluxDB UI.")
```

InfluxDB provides tutorial steps depending on the programming language needed.

Telegraf

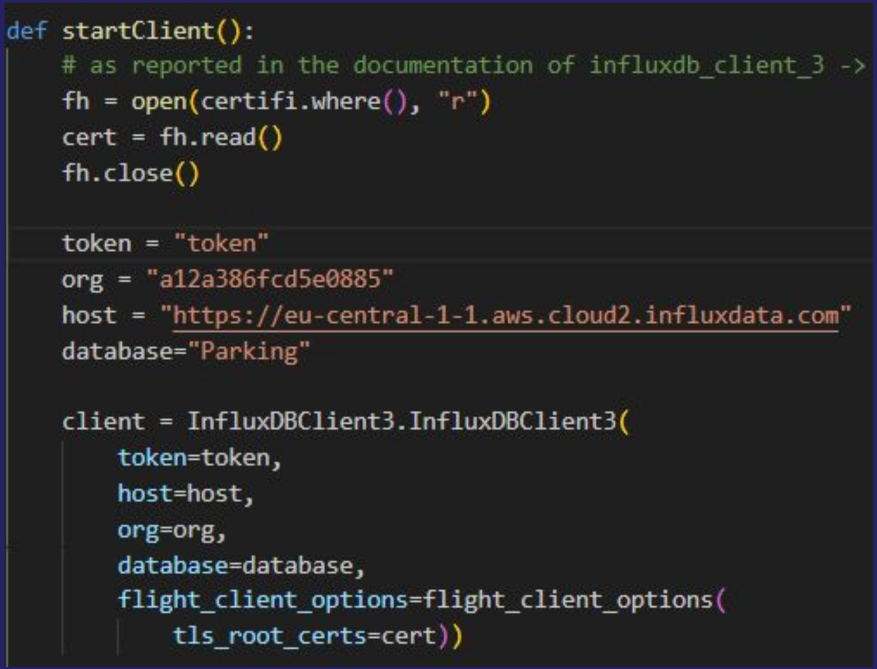


```toml
[[inputs.mqtt_consumer]]
  servers = ["mqtt://test.mosquitto.org:1883"]
  ## Topics that will be subscribed to.
  topics = [
    "provaTopic/#",
  ]

  data_format = "json"
  data_type = "int"

  [[inputs.mqtt_consumer.topic_parsing]]
    topic = "provaTopic/+"
    measurement = "_/measurement"
```

Run telgraf as a MQTT consumer.

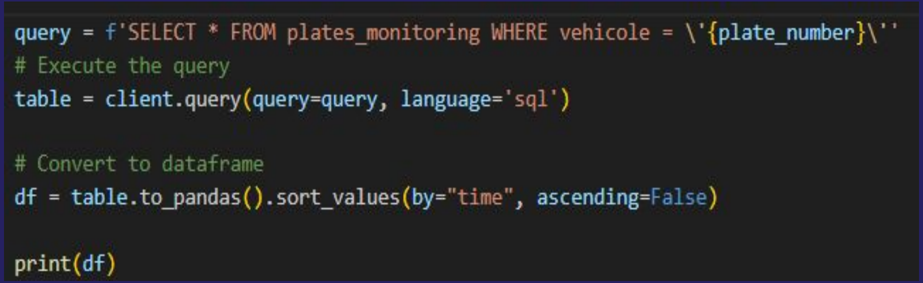
InfluxDB - Explore Data 1/2

In python we can explore the data following the documentation of InfluxDb:



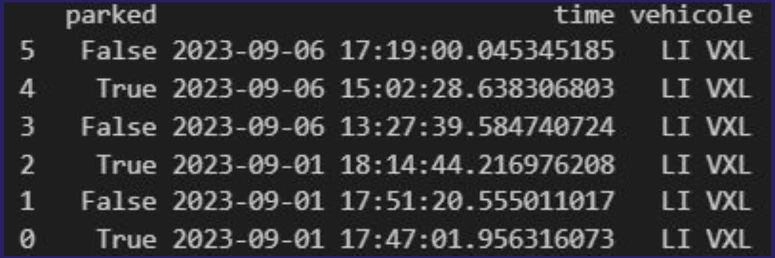
```python
def startClient():
    # as reported in the documentation of influxdb_client_3 ->
    fh = open(certifi.where(), "r")
    cert = fh.read()
    fh.close()

    token = "token"
    org = "a12a386fcd5e0885"
    host = "https://eu-central-1-1.aws.cloud2.influxdata.com"
    database="Parking"

    client = InfluxDBClient3.InfluxDBClient3(
        token=token,
        host=host,
        org=org,
        database=database,
        flight_client_options=flight_client_options(
            tls_root_certs=cert))
```

1) Set the InfluxDB client



```python
query = f'SELECT * FROM plates_monitoring WHERE vehicole = \'{plate_number}\''
# Execute the query
table = client.query(query=query, language='sql')

# Convert to dataframe
df = table.to_pandas().sort_values(by="time", ascending=False)

print(df)
```

2) Perform the query

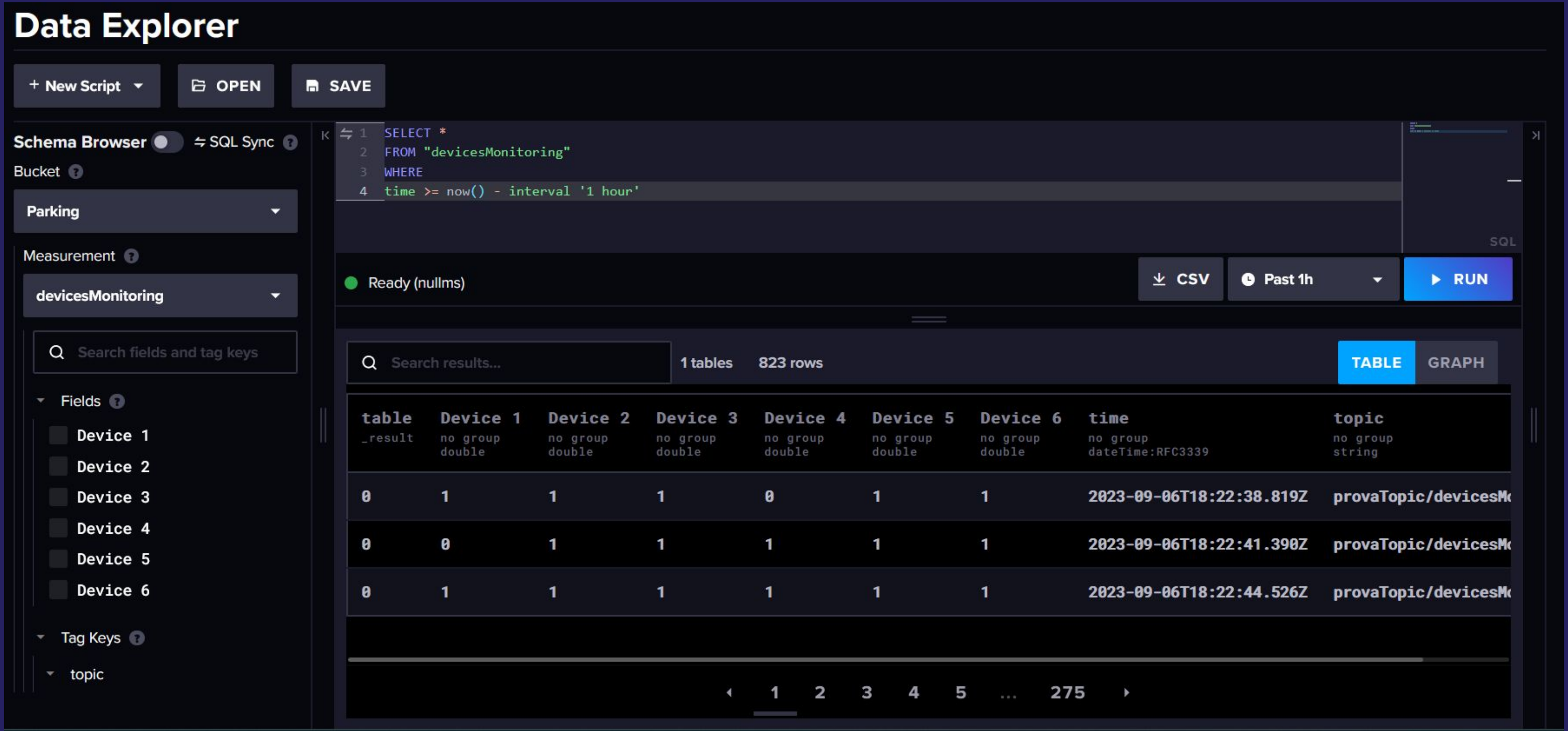


```
  parked                          time vehicole
5  False 2023-09-06 17:19:00.045345185   LI VXL
4   True 2023-09-06 15:02:28.638306803   LI VXL
3  False 2023-09-06 13:27:39.584740724   LI VXL
2   True 2023-09-01 18:14:44.216976208   LI VXL
1  False 2023-09-01 17:51:20.555011017   LI VXL
0   True 2023-09-01 17:47:01.956316073   LI VXL
```

3) Obtain the results

InfluxDB - Explore Data 2/2

Otherwise we can simply use InfluxDB UI to make simple SQL queries:

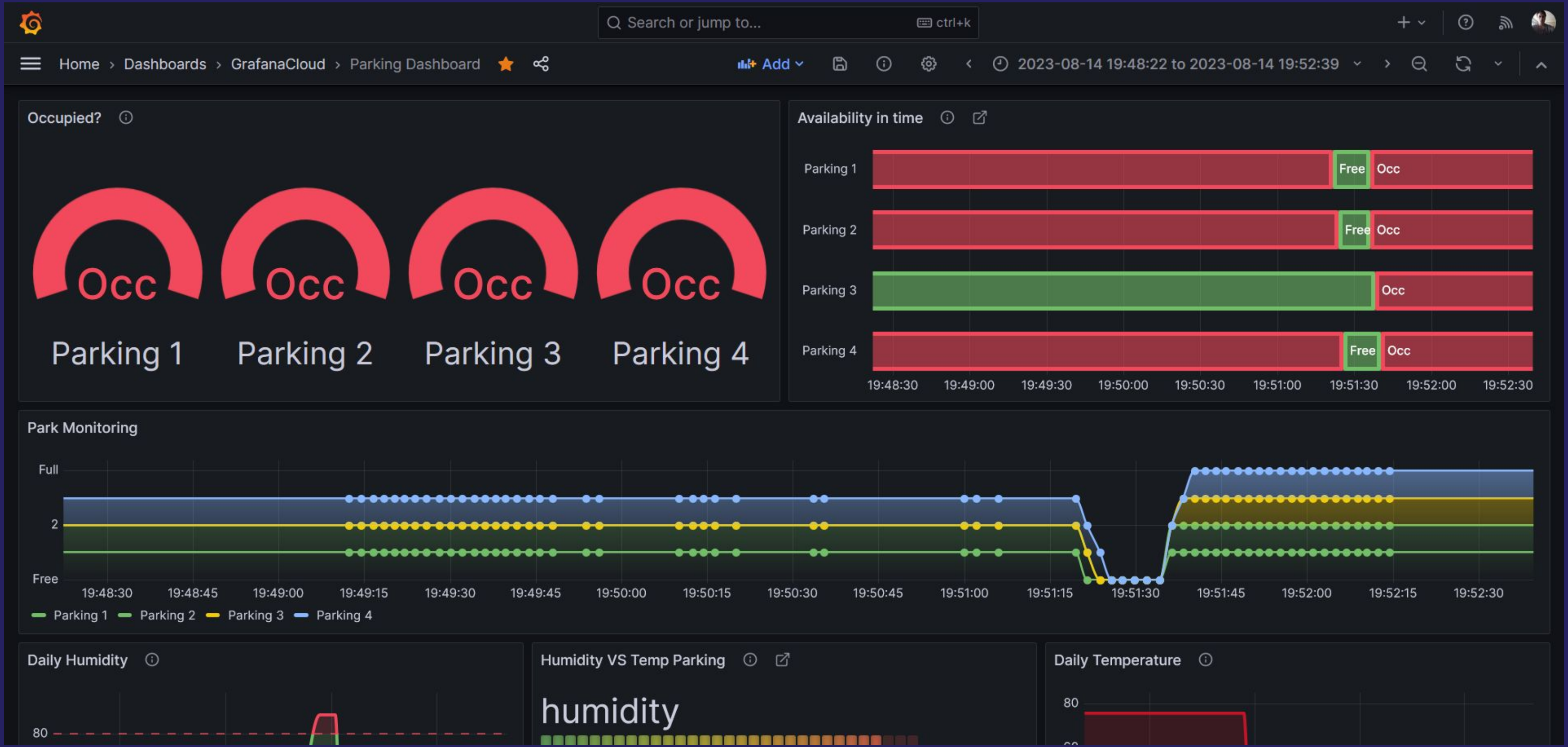


Data Explorer

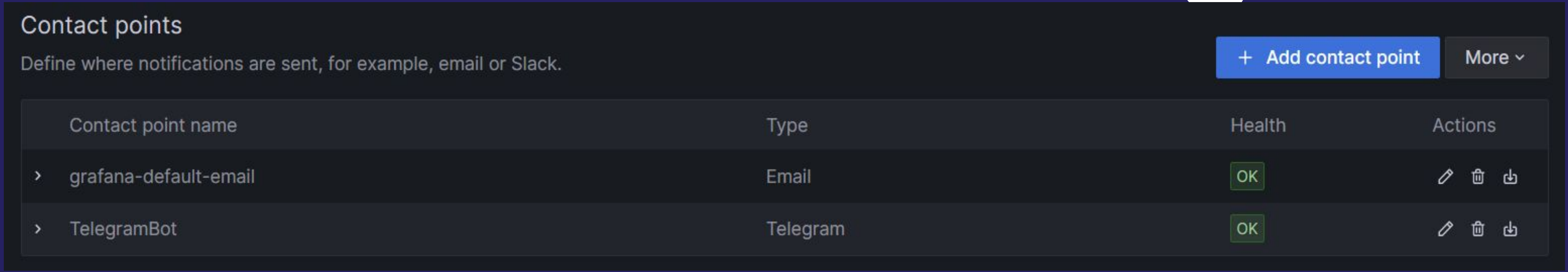
```
SELECT *
FROM "devicesMonitoring"
WHERE
time >= now() - interval '1 hour'
```

table _result	Device 1 no group double	Device 2 no group double	Device 3 no group double	Device 4 no group double	Device 5 no group double	Device 6 no group double	time no group dateTime:RFC3339	topic no group string
0	1	1	1	0	1	1	2023-09-06T18:22:38.819Z	provaTopic/devicesM
0	0	1	1	1	1	1	2023-09-06T18:22:41.390Z	provaTopic/devicesM
0	1	1	1	1	1	1	2023-09-06T18:22:44.526Z	provaTopic/devicesM

Grafana - Monitoring



Grafana - Alerting

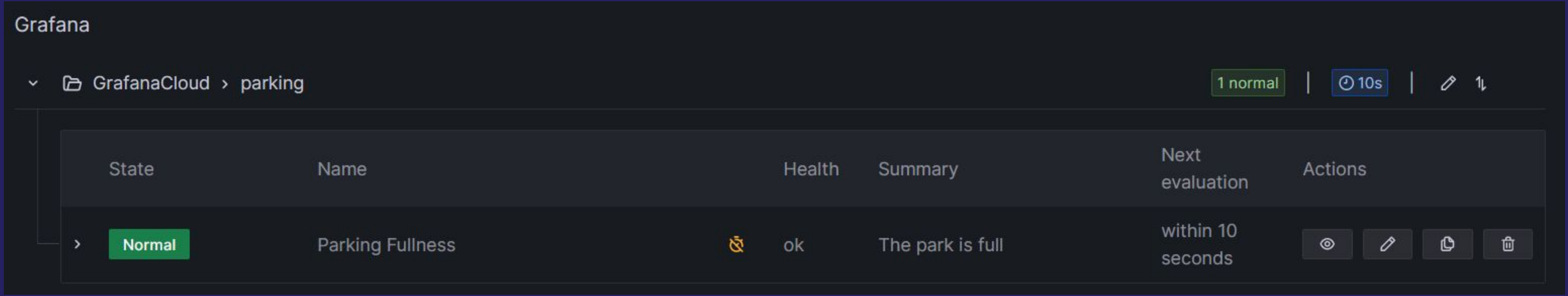


Contact points

Define where notifications are sent, for example, email or Slack.

	Contact point name	Type	Health	Actions
›	grafana-default-email	Email	OK	
›	TelegramBot	Telegram	OK	

1) Define the contact points



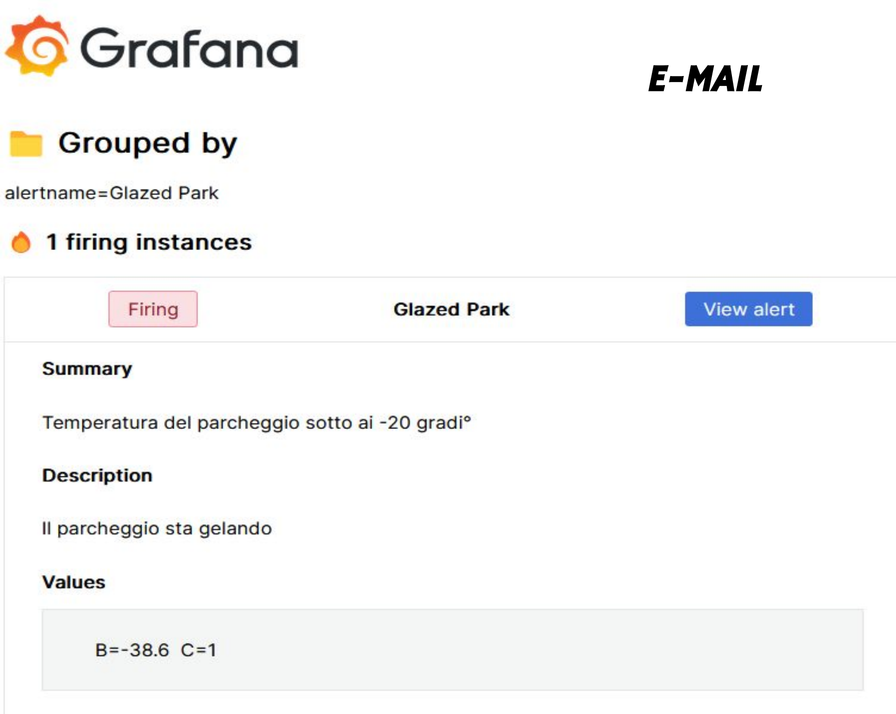
Grafana

GrafanaCloud › parking

	State	Name	Health	Summary	Next evaluation	Actions
›	Normal	Parking Fullness	ok	The park is full	within 10 seconds	

2) Define alert rules

Grafana - Alerting



E-MAIL

Grafana

Grouped by

alertname=Glazed Park

1 firing instances

Firing	Glazed Park	View alert

Summary

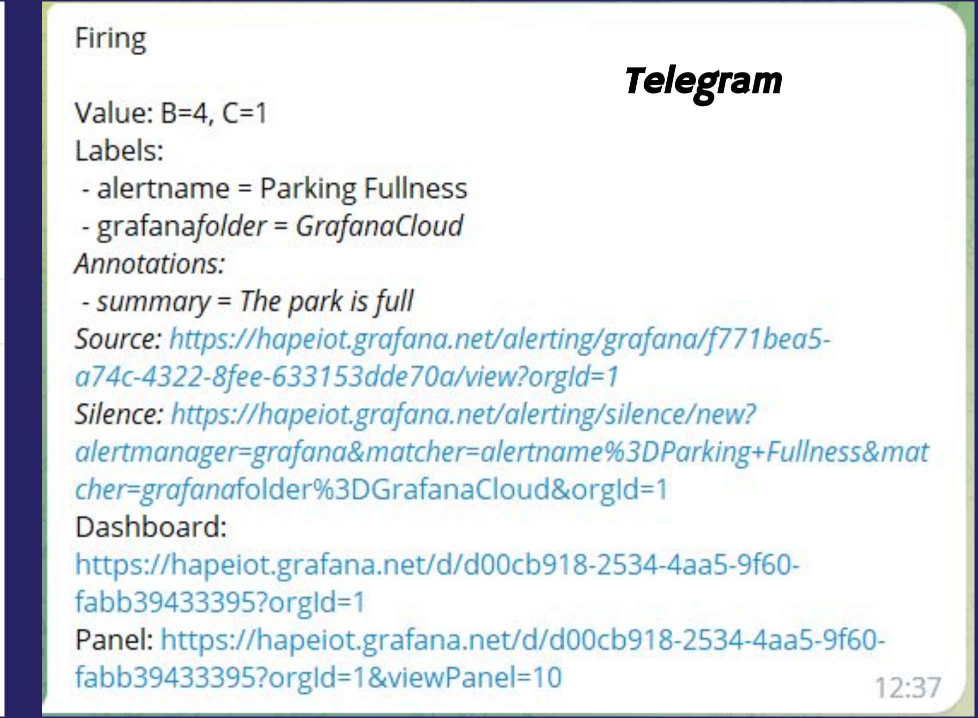
Temperatura del parcheggio sotto ai -20 gradi°

Description

Il parcheggio sta gelando

Values

```
B=-38.6  C=1
```



Telegram

Firing

Value: B=4, C=1
Labels:
- alertname = Parking Fullness
- grafanafolder = GrafanaCloud

Annotations:
- *summary = The park is full*

Source: https://hapeiot.grafana.net/alerting/grafana/f771bea5-a74c-4322-8fee-633153dde70a/view?orgId=1
Silence: https://hapeiot.grafana.net/alerting/silence/new?alertmanager=grafana&matcher=alertname%3DParking+Fullness&matcher=grafanafolder%3DGrafanaCloud&orgId=1
Dashboard: https://hapeiot.grafana.net/d/d00cb918-2534-4aa5-9f60-fabb39433395?orgId=1
Panel: https://hapeiot.grafana.net/d/d00cb918-2534-4aa5-9f60-fabb39433395?orgId=1&viewPanel=10

12:37

3) Grafana will fire alerts when the rules are matched by the data

WOKWI - on VS with PIO

Simulation with ESP32 microcontroller

That's All!

Have a nice Parking!