influxdata®

Building a Plant Monitoring App with InfluxDB, Python, and Flask with Edge to cloud replication

Anais Dotis Georgiou

Anais Dotis-Georgiou
Developer Advocate

LinkedIn

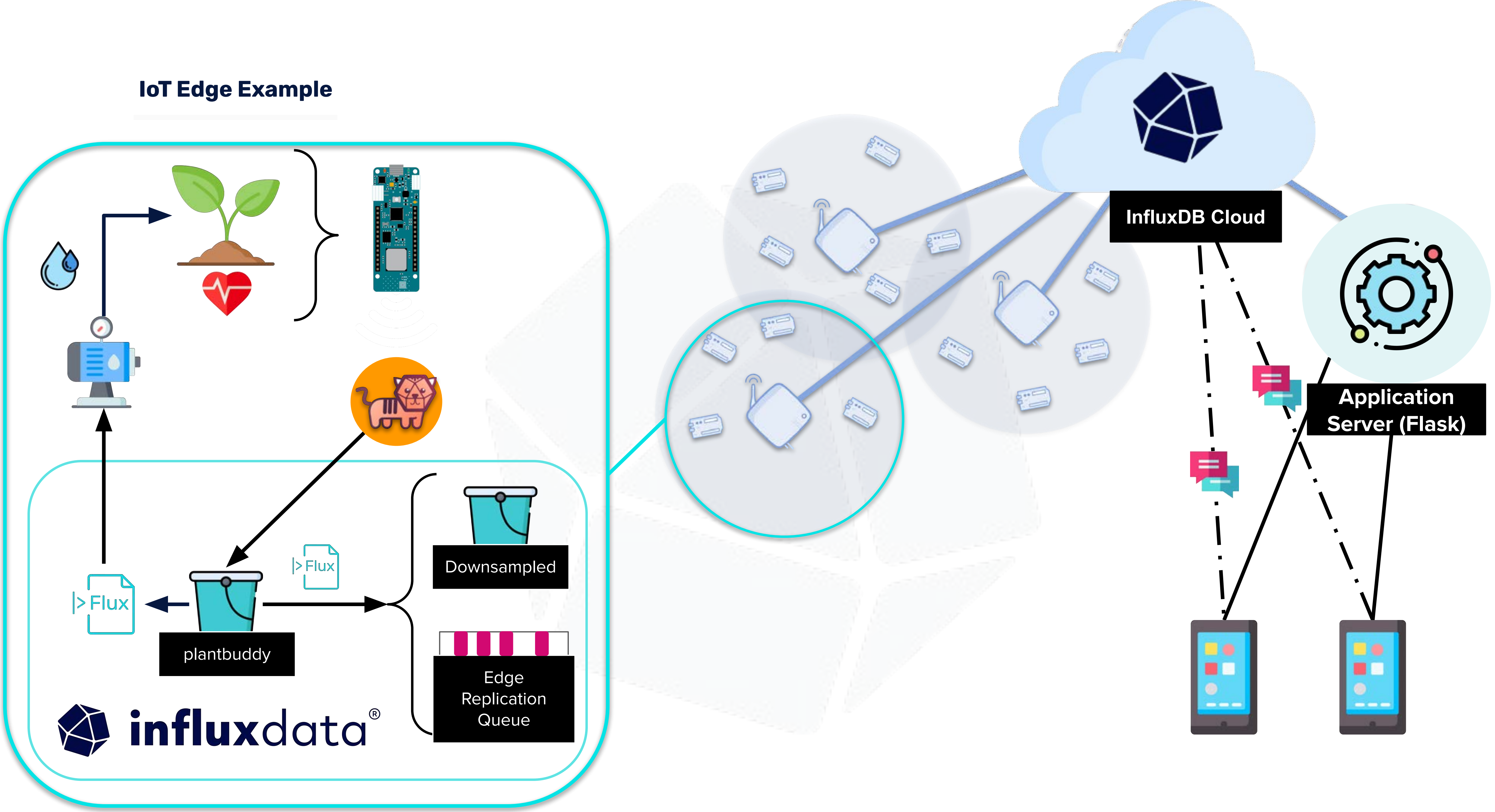
The Overview:

This will be a walkthrough in how to build this plant monitoring project:

- IOT Hardware setup
- Tools
- InfluxDB overview
- Data Ingestion Setup
- Flux + SQL
- Setup EDR
- Data Request
- Github Code Base + Q&A

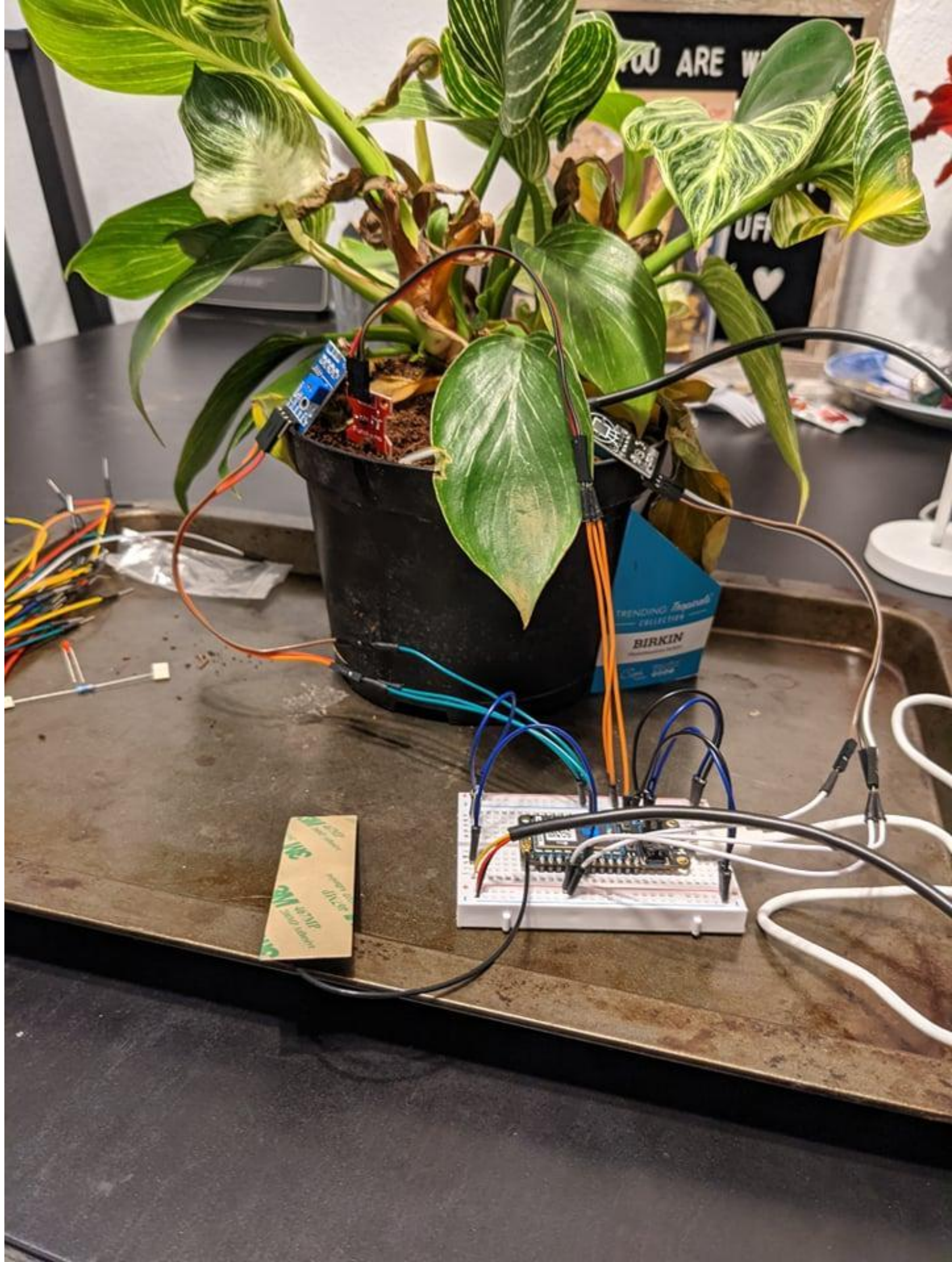
Set Up IOT Device

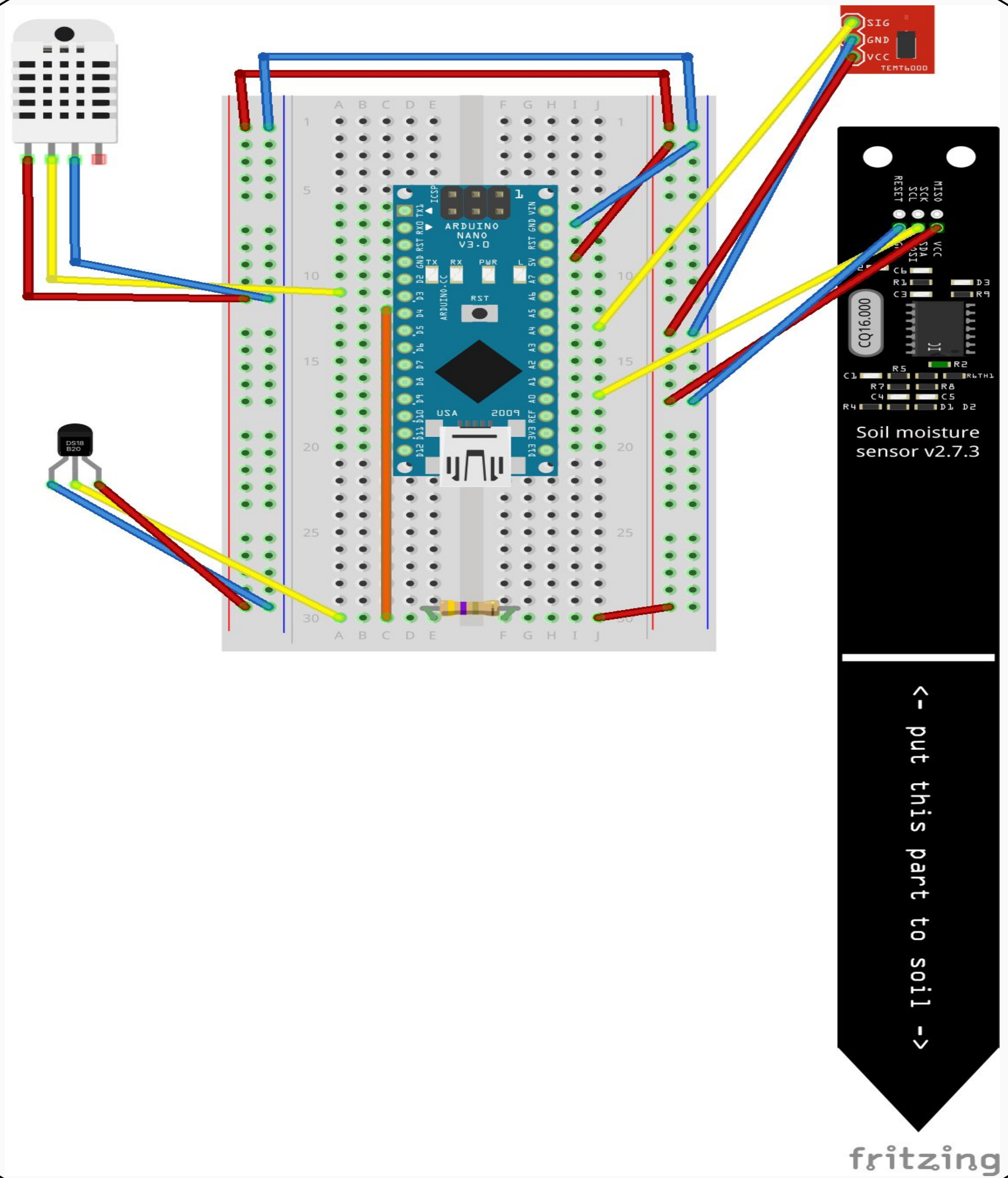
IoT Edge Example



You will need in no particular order:

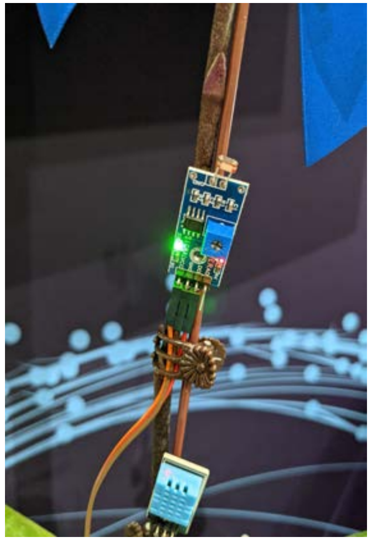
- A plant, preferably alive
- A particle boron microcontroller, or another compatible microcontroller
- At least one IOT sensor for your plant
- A breadboard with jump wires and terminal strips



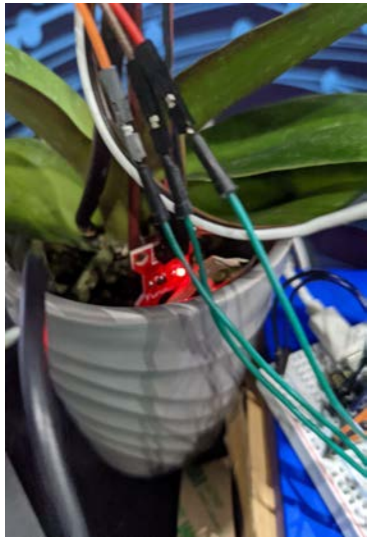


Schematics & Sensors

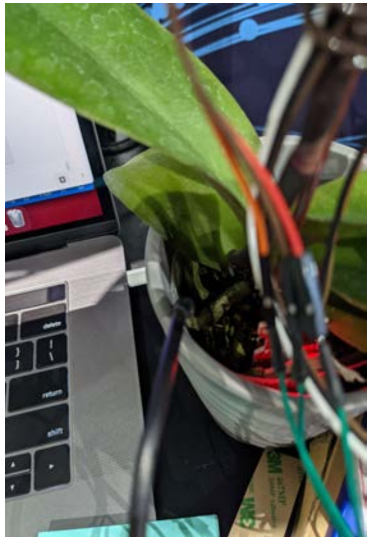
Temperature & Humidity



Light



Soil Moisture



Soil Temperature

Tools

Flask Framework

InfluxDB for Storage

1

POWERFUL

API & Toolset

for real-time apps

2

HIGH PERFORMANCE

Time Series Engine

for real-time data workloads

3

MASSIVE

Community & Ecosystem

of cloud & open source developers

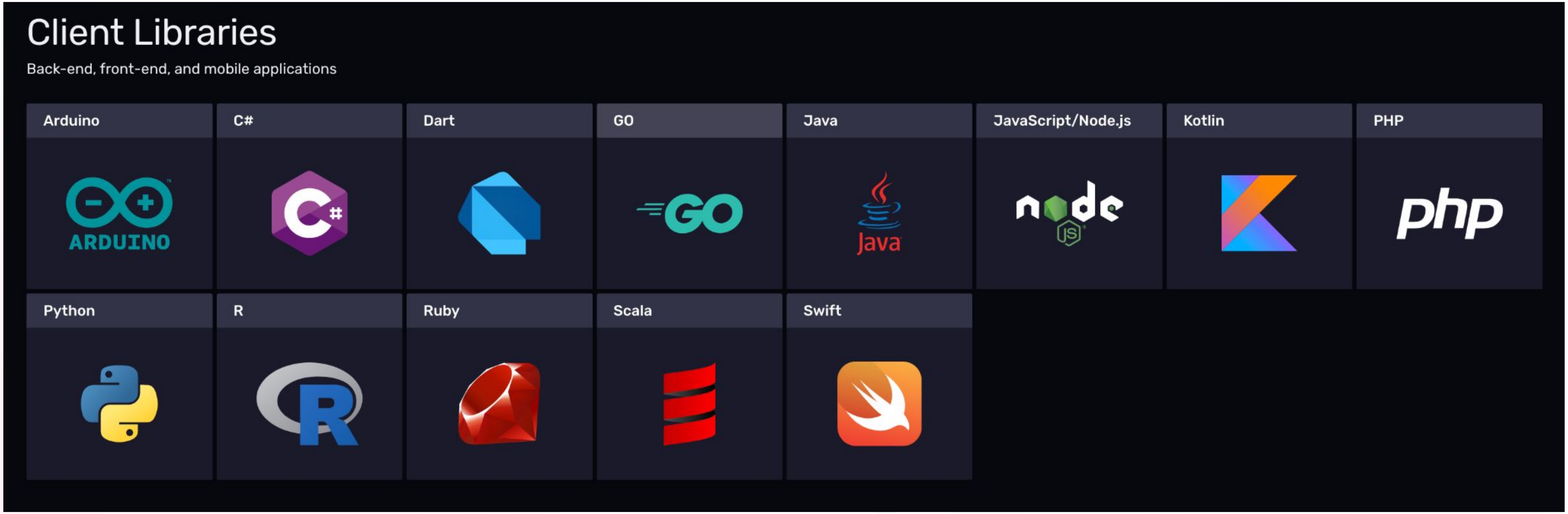
Telegraf for Ingestion

The open-source agent for collecting metrics

Driven by the community (600+ contributors)

Simple to configure, extremely flexible

Client Libraries

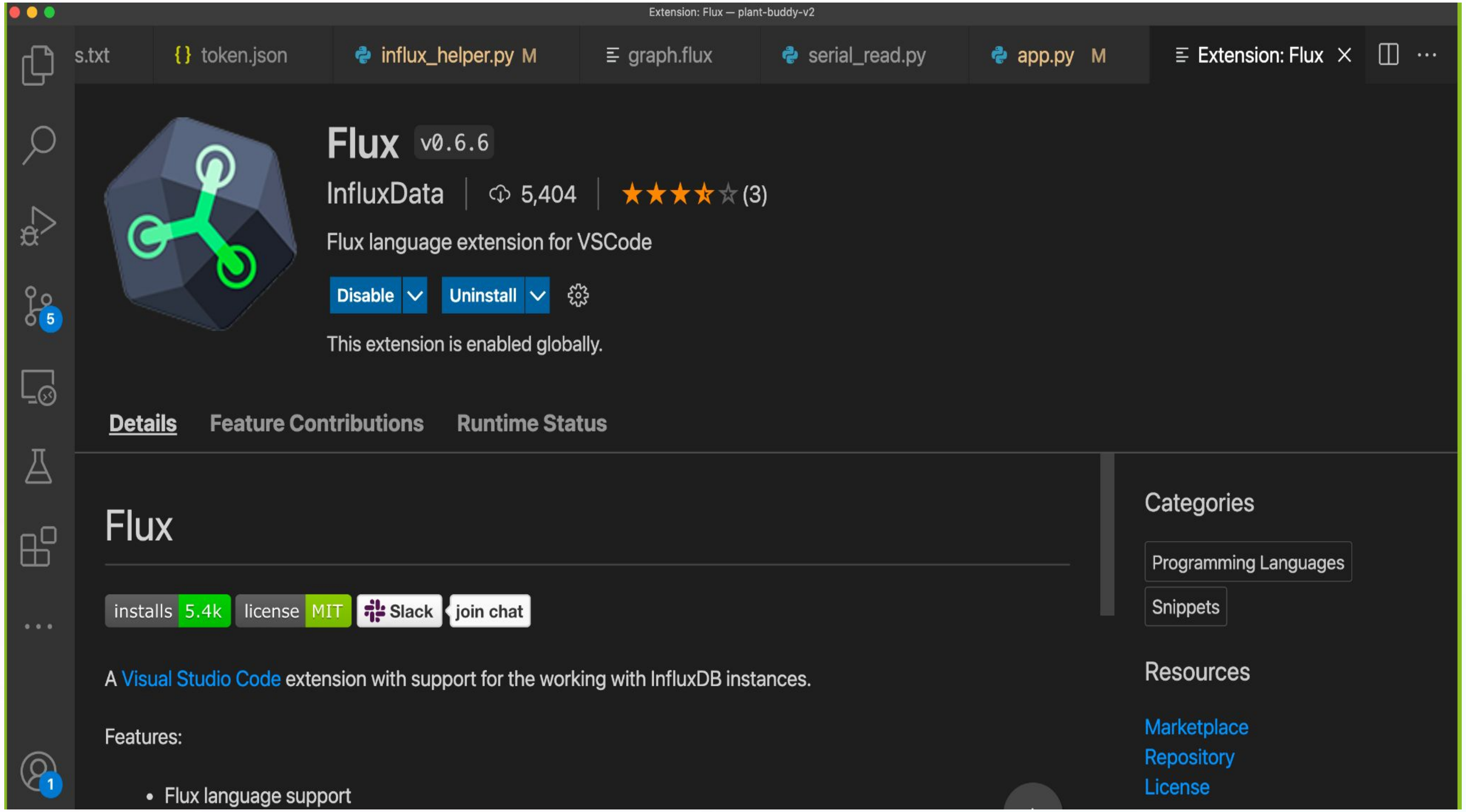


Client Libraries

Back-end, front-end, and mobile applications

Arduino	C#	Dart	GO	Java	JavaScript/Node.js	Kotlin	PHP
Python	R	Ruby	Scala	Swift			

Flux Extension for VS code



Flux v0.6.6

InfluxData | 5,404 | ★★★★☆ (3)

Flux language extension for VSCode

This extension is enabled globally.

Flux

A Visual Studio Code extension with support for the working with InfluxDB instances.

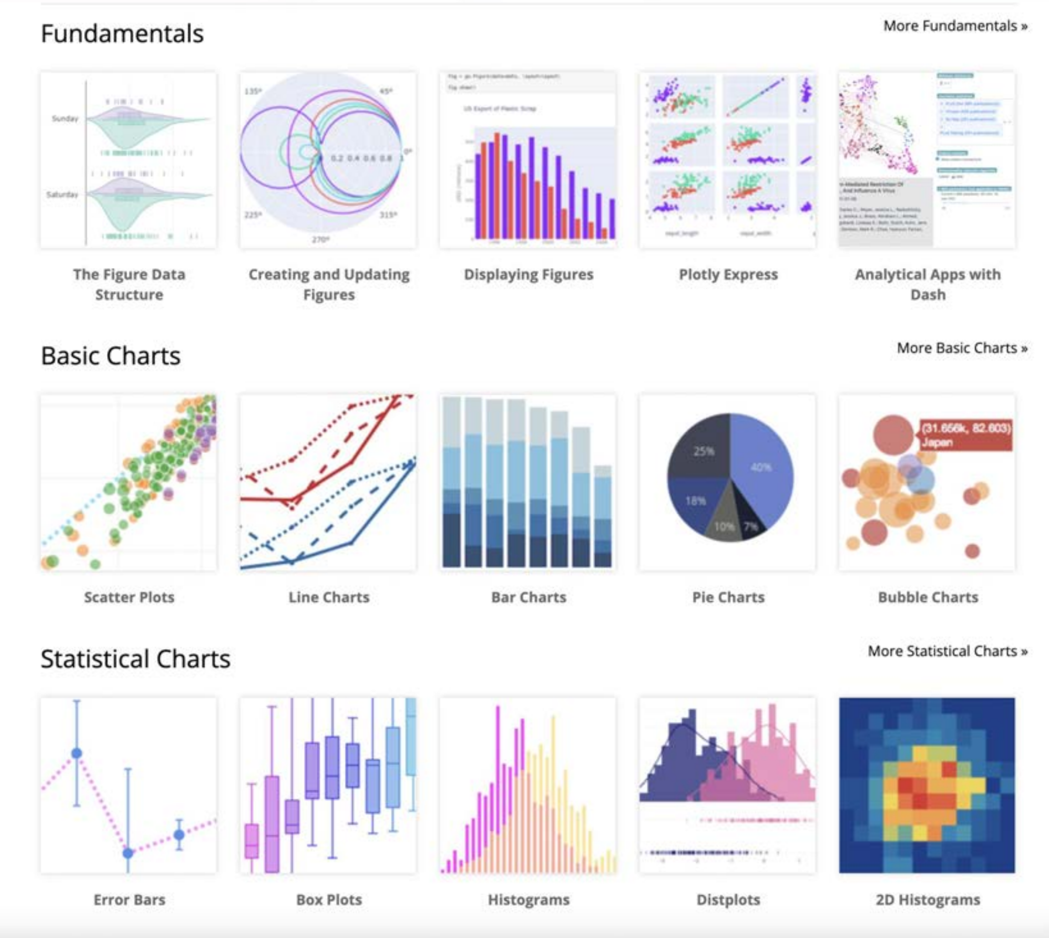
Features:

- Flux language support

Categories

Resources

Plotly for Graphing



InfluxDB Overview

Time Series Data, what is it?

A sequence of data points, typically consisting of successive measurements made from the same source over a time interval.

Examples:

- Weather condition
- Stock exchange
- Cluster monitoring
- Healthcare
- Logs
- Traces

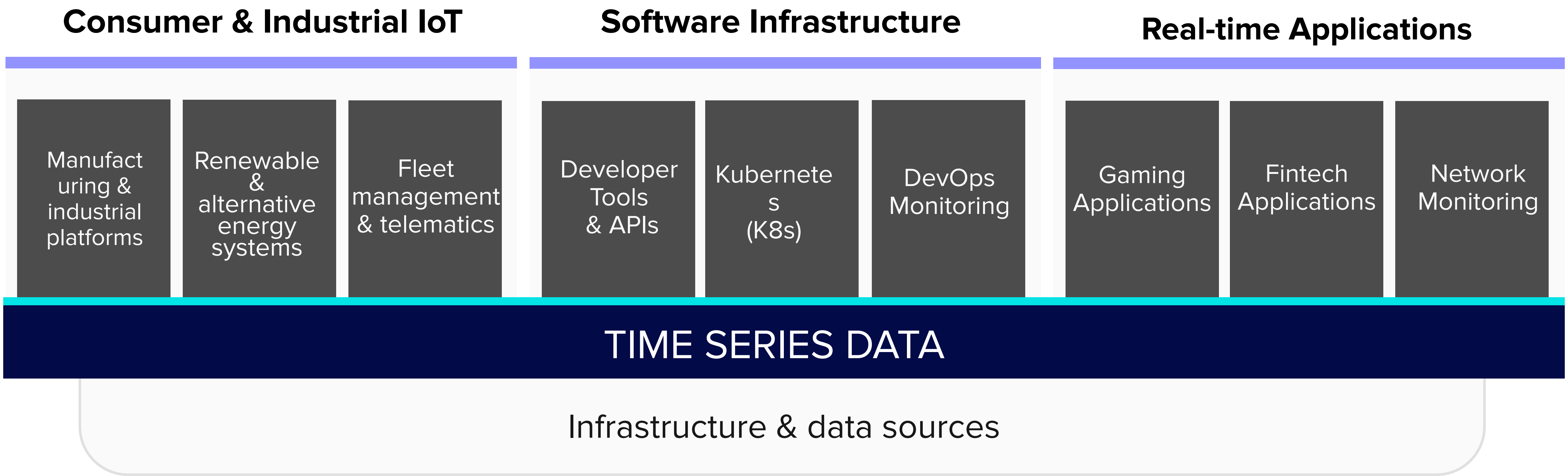
Metrics (Regular)

Measurements gathered at regular time intervals

Events (Irregular)

Measurements gathered at irregular time intervals

Time series in every application



Consumer & Industrial IoT

- Manufacturing & industrial platforms
- Renewable & alternative energy systems
- Fleet management & telematics

Software Infrastructure

- Developer Tools & APIs
- Kubernetes (K8s)
- DevOps Monitoring

Real-time Applications

- Gaming Applications
- Fintech Applications
- Network Monitoring

TIME SERIES DATA

Infrastructure & data sources

Time Series DB

RELATIONAL

- Orders
- Customers
- Records

PostgreSQL

DOCUMENT

- High throughput
- Large document

MongoDB.

SEARCH

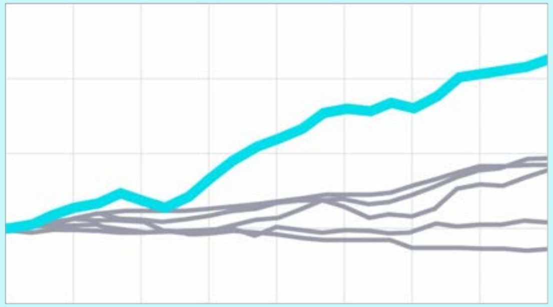
- Distributed search
- Logs
- Geo

elastic

TIME SERIES

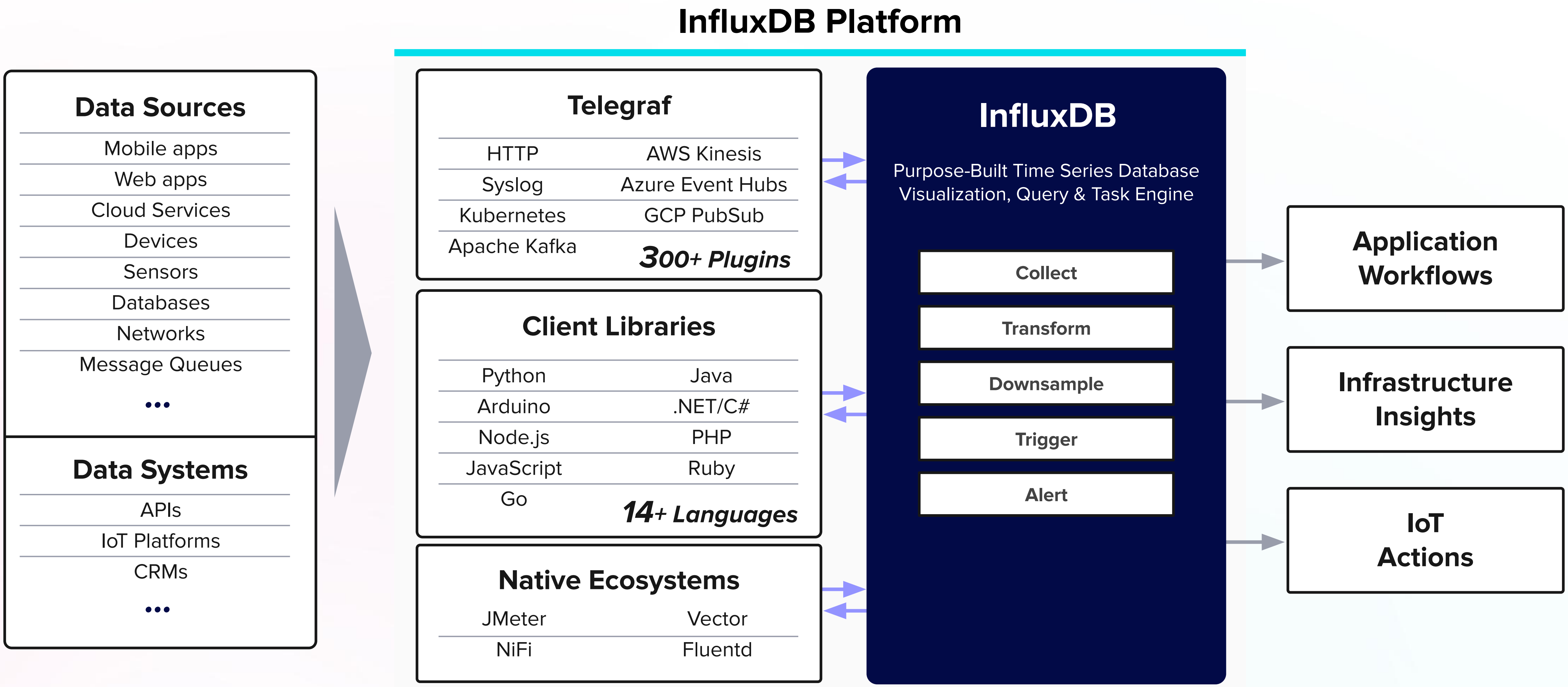
- Events, metrics, time stamped
- for IoT, analytics, cloud native

Time Series Category Trend



influxdata®

InfluxDB + Telegraf + Flux



InfluxDB Platform

Data Sources

- Mobile apps
- Web apps
- Cloud Services
- Devices
- Sensors
- Databases
- Networks
- Message Queues
- •••

Data Systems

- APIs
- IoT Platforms
- CRMs
- •••

Telegraf

HTTP	AWS Kinesis
Syslog	Azure Event Hubs
Kubernetes	GCP PubSub
Apache Kafka	***300+ Plugins***

Client Libraries

Python	Java
Arduino	.NET/C#
Node.js	PHP
JavaScript	Ruby
Go	***14+ Languages***

Native Ecosystems

JMeter	Vector
NiFi	Fluentd

InfluxDB

Purpose-Built Time Series Database
Visualization, Query & Task Engine

- Collect
- Transform
- Downsample
- Trigger
- Alert

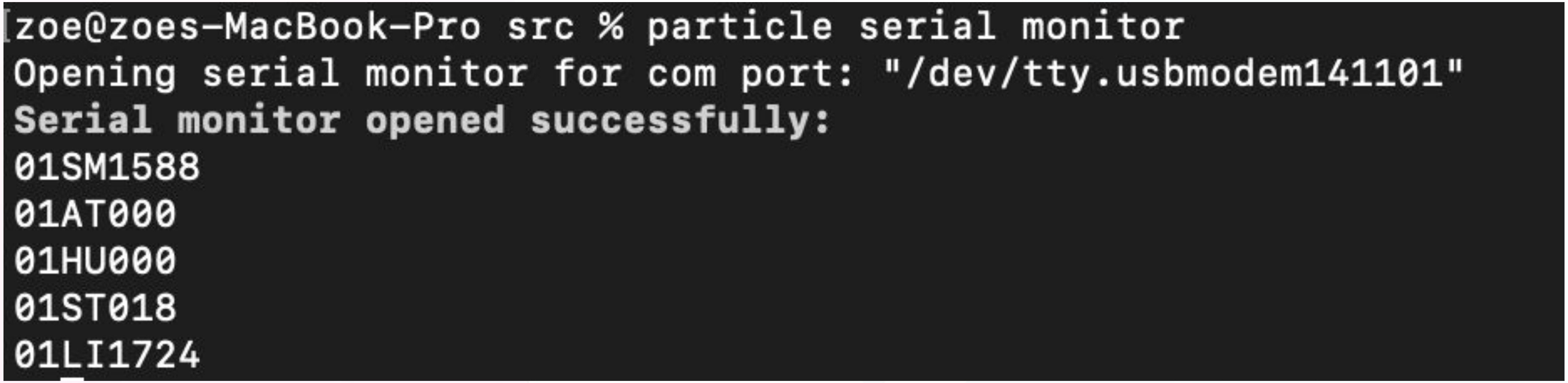
Application Workflows

Infrastructure Insights

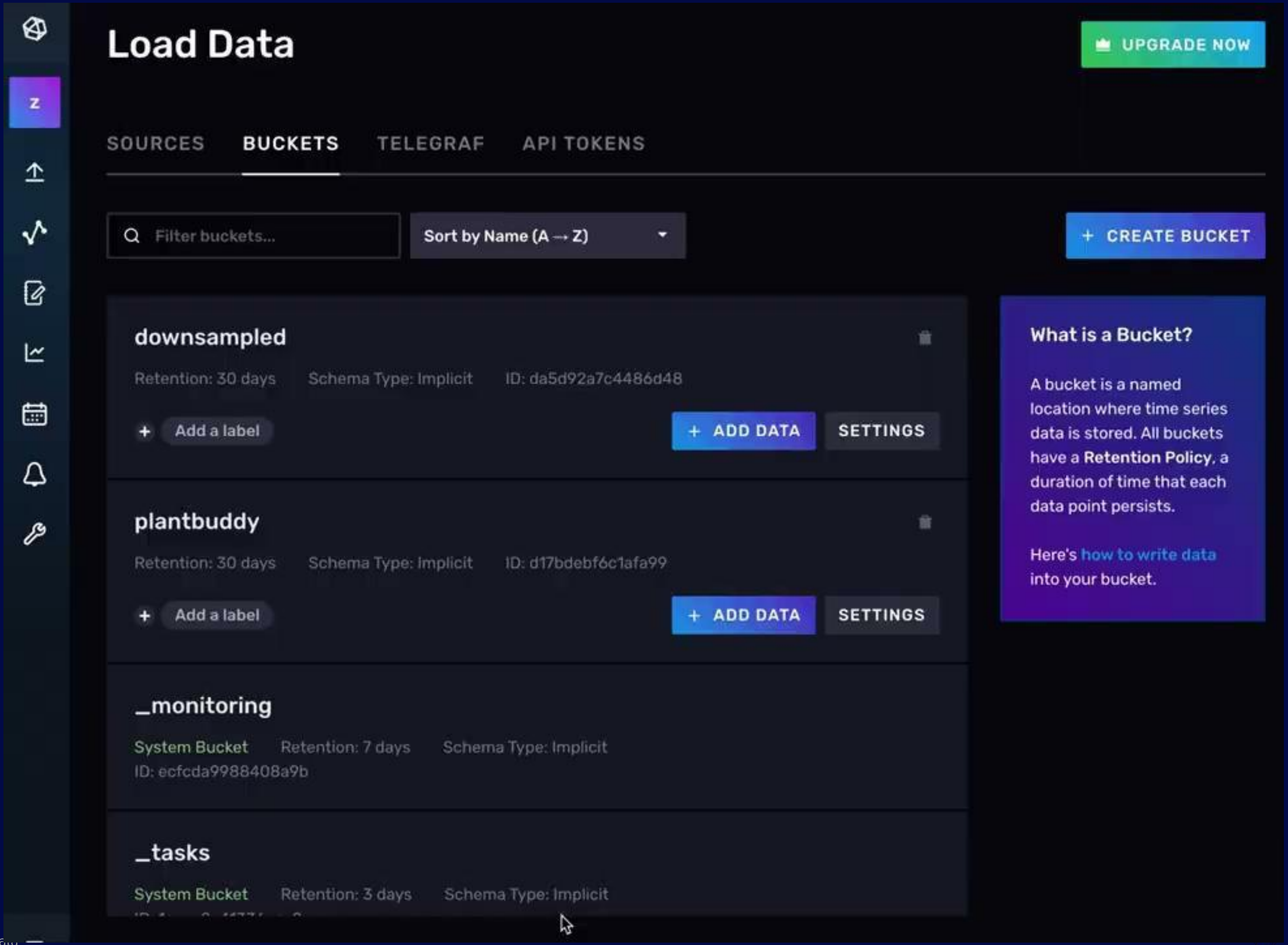
IoT Actions

Data Ingestion Setup

Connecting to the microcontroller



```
zoe@zoes-MacBook-Pro src % particle serial monitor
Opening serial monitor for com port: "/dev/tty.usbmodem141101"
Serial monitor opened successfully:
01SM1588
01AT000
01HU000
01ST018
01LI1724
```



Load Data

UPGRADE NOW

SOURCES | BUCKETS | TELEGRAF | API TOKENS

Filter buckets...

Sort by Name (A → Z)

+ CREATE BUCKET

downsampled

Retention: 30 days   Schema Type: Implicit   ID: da5d92a7c4486d48

+ Add a label   + ADD DATA   SETTINGS

plantbuddy

Retention: 30 days   Schema Type: Implicit   ID: d17bdebf6c1afa99

+ Add a label   + ADD DATA   SETTINGS

_monitoring

System Bucket   Retention: 7 days   Schema Type: Implicit

ID: ecfcda9988408a9b

_tasks

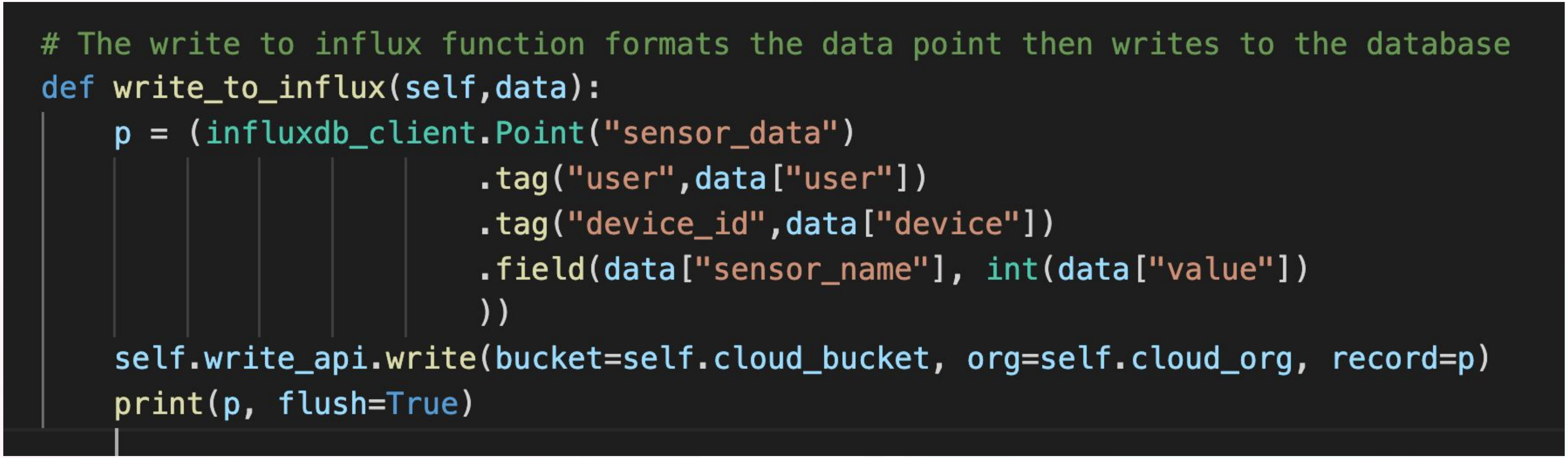
System Bucket   Retention: 3 days   Schema Type: Implicit

What is a Bucket?

A bucket is a named location where time series data is stored. All buckets have a **Retention Policy**, a duration of time that each data point persists.

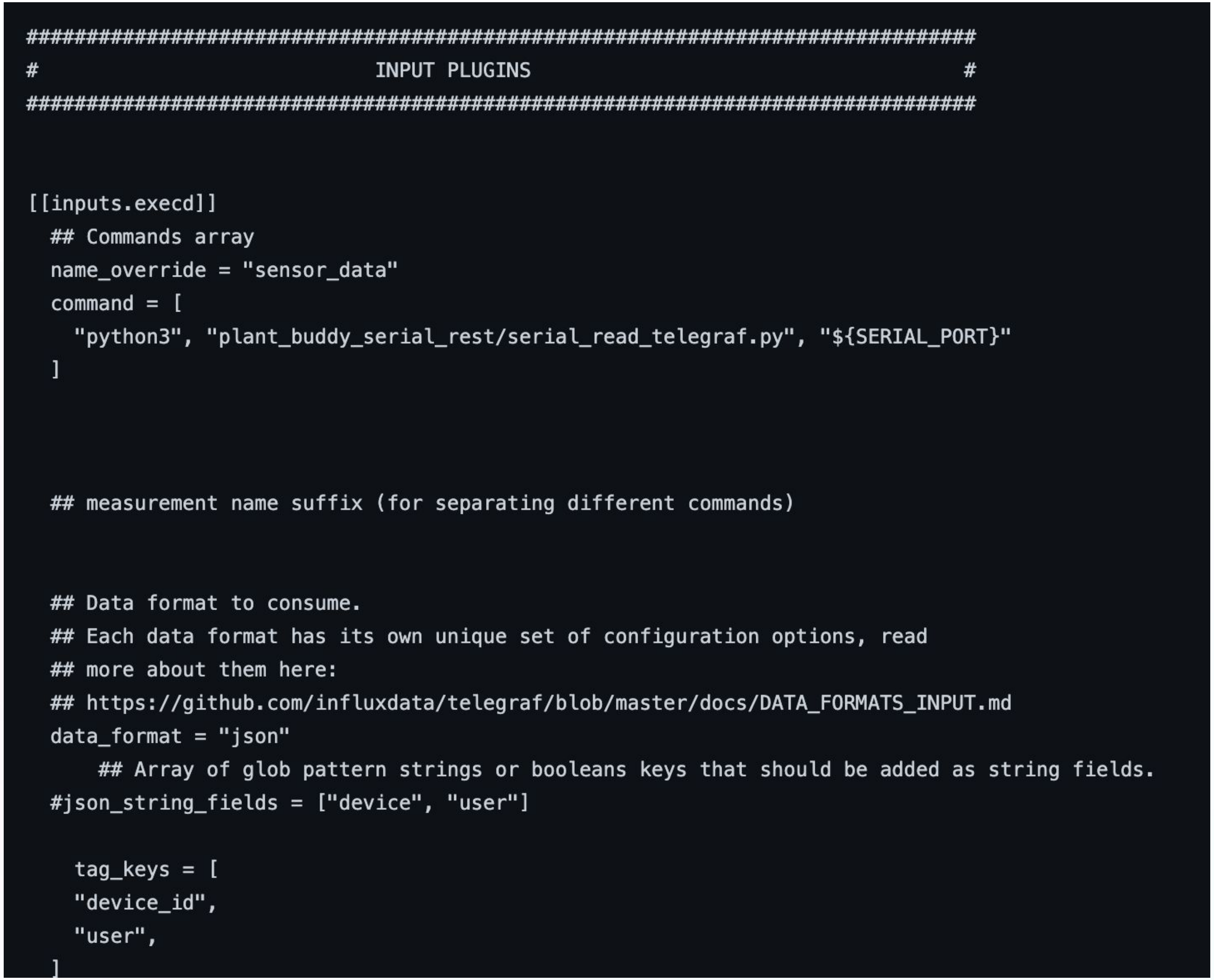
Here's how to write data into your bucket.

Writing the data into influxdb



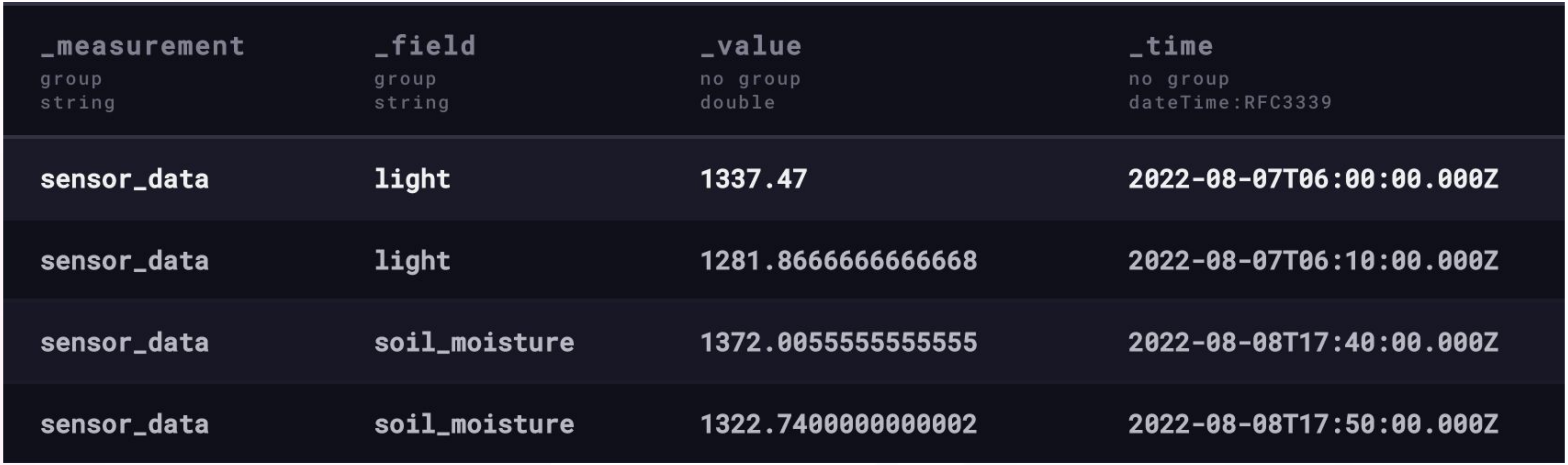
```python
# The write to influx function formats the data point then writes to the database
def write_to_influx(self,data):
    p = (influxdb_client.Point("sensor_data")
                        .tag("user",data["user"])
                        .tag("device_id",data["device"])
                        .field(data["sensor_name"], int(data["value"]))
                        ))
    self.write_api.write(bucket=self.cloud_bucket, org=self.cloud_org, record=p)
    print(p, flush=True)
```

Writing the data into influxdb with Telegraf



```
###############################################################################
#                            INPUT PLUGINS                                    #
###############################################################################


[[inputs.execd]]
  ## Commands array
  name_override = "sensor_data"
  command = [
    "python3", "plant_buddy_serial_rest/serial_read_telegraf.py", "${SERIAL_PORT}"
  ]



  ## measurement name suffix (for separating different commands)


  ## 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_FORMATS_INPUT.md
  data_format = "json"
      ## Array of glob pattern strings or booleans keys that should be added as string fields.
  #json_string_fields = ["device", "user"]

    tag_keys = [
    "device_id",
    "user",
  ]
```

Table example of the resulting data points

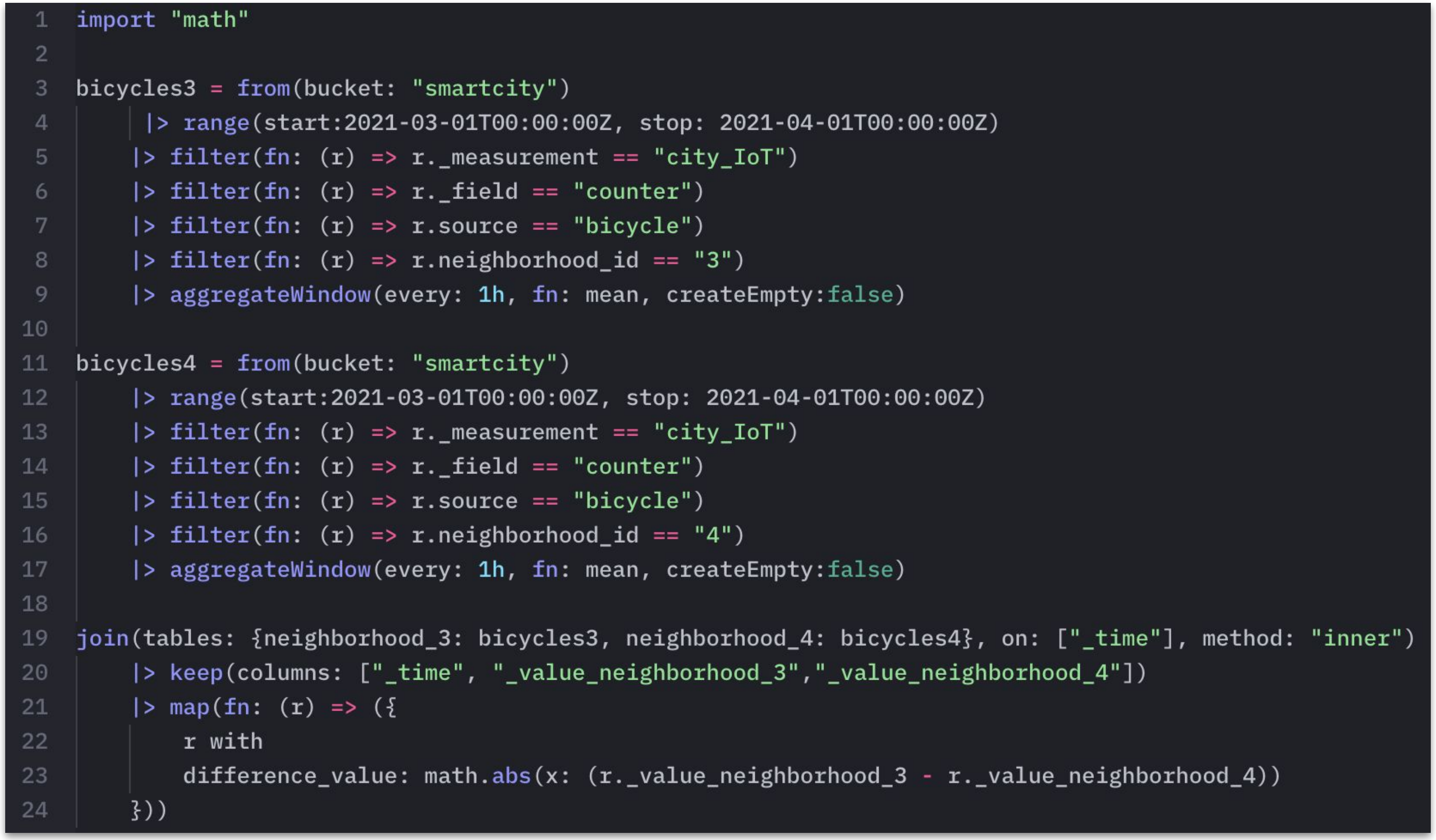


_measurement group string	_field group string	_value no group double	_time no group dateTime:RFC3339
sensor_data	light	1337.47	2022-08-07T06:00:00.000Z
sensor_data	light	1281.8666666666668	2022-08-07T06:10:00.000Z
sensor_data	soil_moisture	1372.0055555555555	2022-08-08T17:40:00.000Z
sensor_data	soil_moisture	1322.7400000000002	2022-08-08T17:50:00.000Z

Flux -> SQL

Introducing Flux

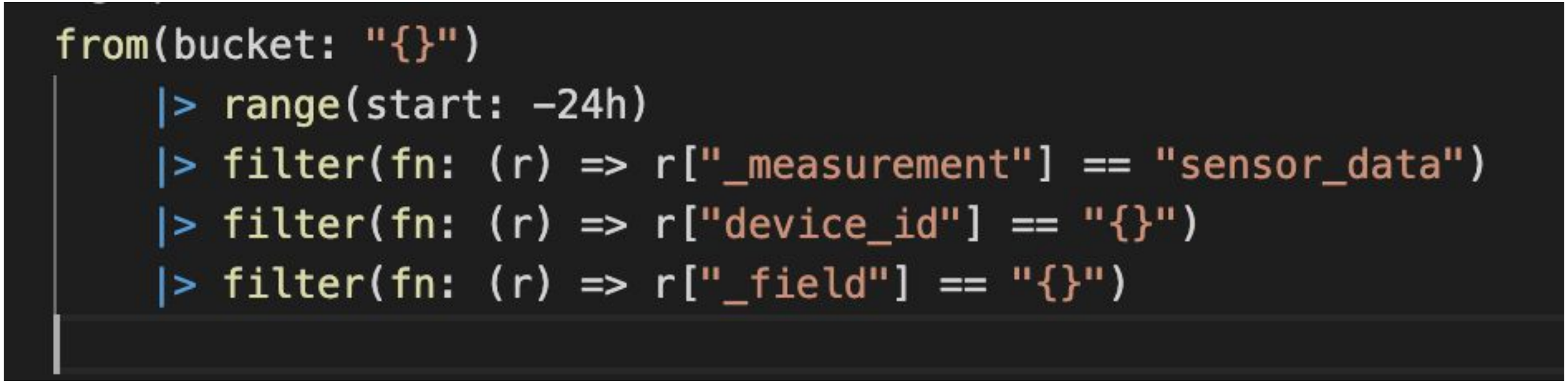
A functional language designed for querying, analyzing, and acting on data.



```
import "math"

bicycles3 = from(bucket: "smartcity")
     |> range(start:2021-03-01T00:00:00Z, stop: 2021-04-01T00:00:00Z)
    |> filter(fn: (r) => r._measurement == "city_IoT")
    |> filter(fn: (r) => r._field == "counter")
    |> filter(fn: (r) => r.source == "bicycle")
    |> filter(fn: (r) => r.neighborhood_id == "3")
    |> aggregateWindow(every: 1h, fn: mean, createEmpty:false)

bicycles4 = from(bucket: "smartcity")
    |> range(start:2021-03-01T00:00:00Z, stop: 2021-04-01T00:00:00Z)
    |> filter(fn: (r) => r._measurement == "city_IoT")
    |> filter(fn: (r) => r._field == "counter")
    |> filter(fn: (r) => r.source == "bicycle")
    |> filter(fn: (r) => r.neighborhood_id == "4")
    |> aggregateWindow(every: 1h, fn: mean, createEmpty:false)

join(tables: {neighborhood_3: bicycles3, neighborhood_4: bicycles4}, on: ["_time"], method: "inner")
    |> keep(columns: ["_time", "_value_neighborhood_3","_value_neighborhood_4"])
    |> map(fn: (r) => ({
        r with
        difference_value: math.abs(x: (r._value_neighborhood_3 - r._value_neighborhood_4))
    }))
```

|

Flux Query



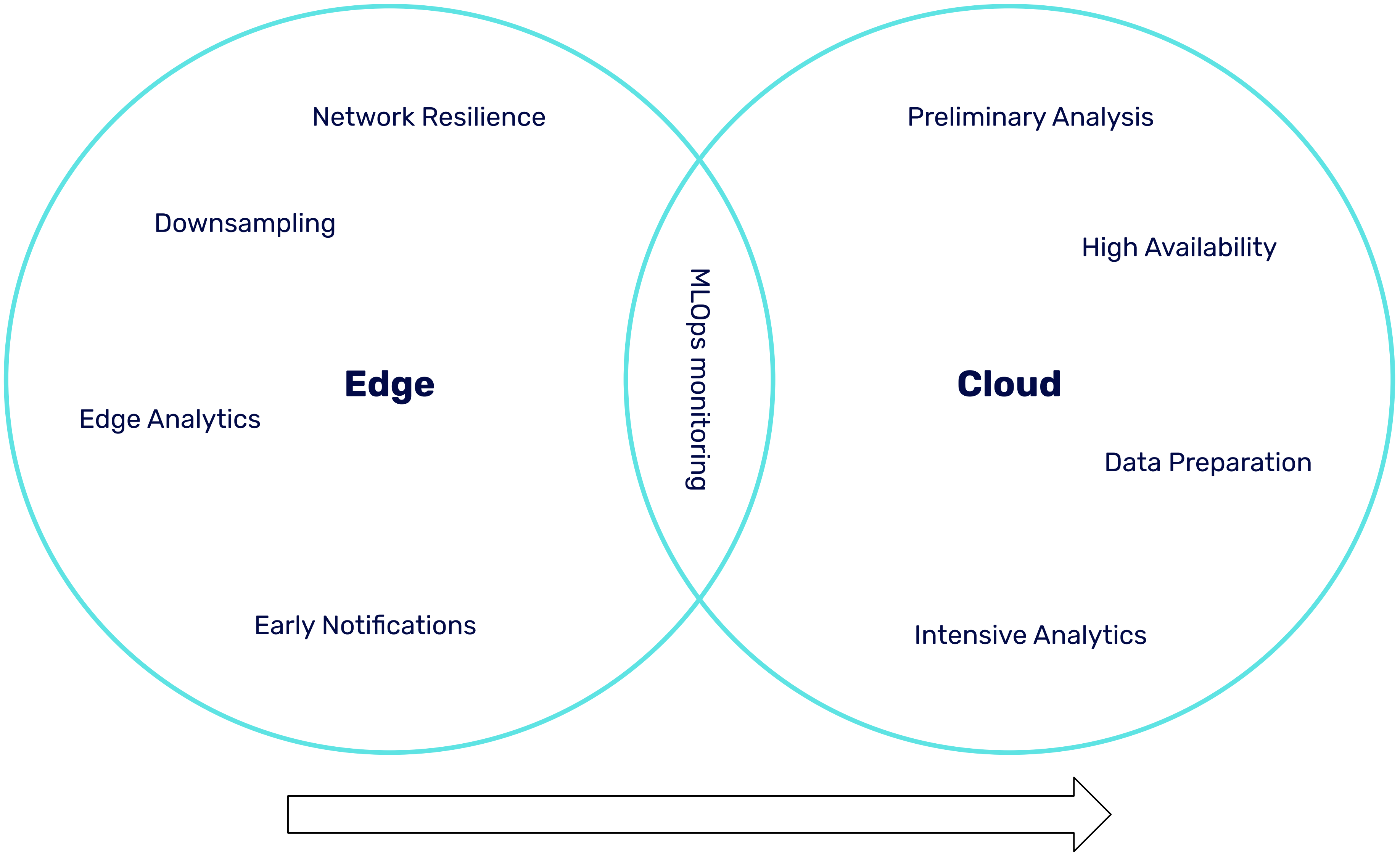
```
from(bucket: "{}")
    |> range(start: -24h)
    |> filter(fn: (r) => r["_measurement"] == "sensor_data")
    |> filter(fn: (r) => r["device_id"] == "{}")
    |> filter(fn: (r) => r["_field"] == "{}")
```

Change is here!

The future of InfluxDB Cloud and in the future Open Source

- IOx powered InfluxDB Cloud brings SQL support
- SQL editor within InfluxDB Cloud in development
- FlightSQL plugins (Present + Future):
  - Apache Superset
  - Tableau
  - PowerBI
  - Grafana

Edge Data Replication



Network Resilience

Downsampling

Edge

Edge Analytics

Early Notifications

MLOps monitoring

Preliminary Analysis

High Availability

Cloud

Data Preparation

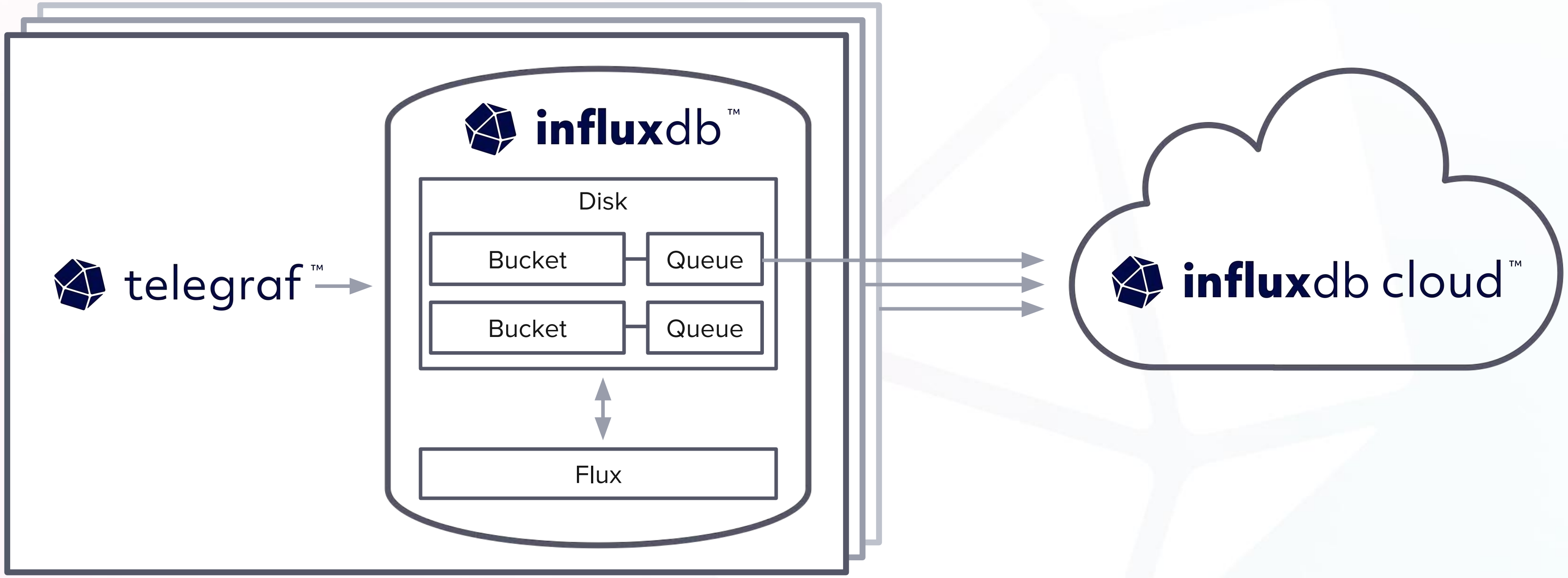
Intensive Analytics

Hybrid Solution

InfluxData Edge Data Replication

Edge (InfluxDB OSS) Databases

Cloud (InfluxDB Cloud) Database(s)



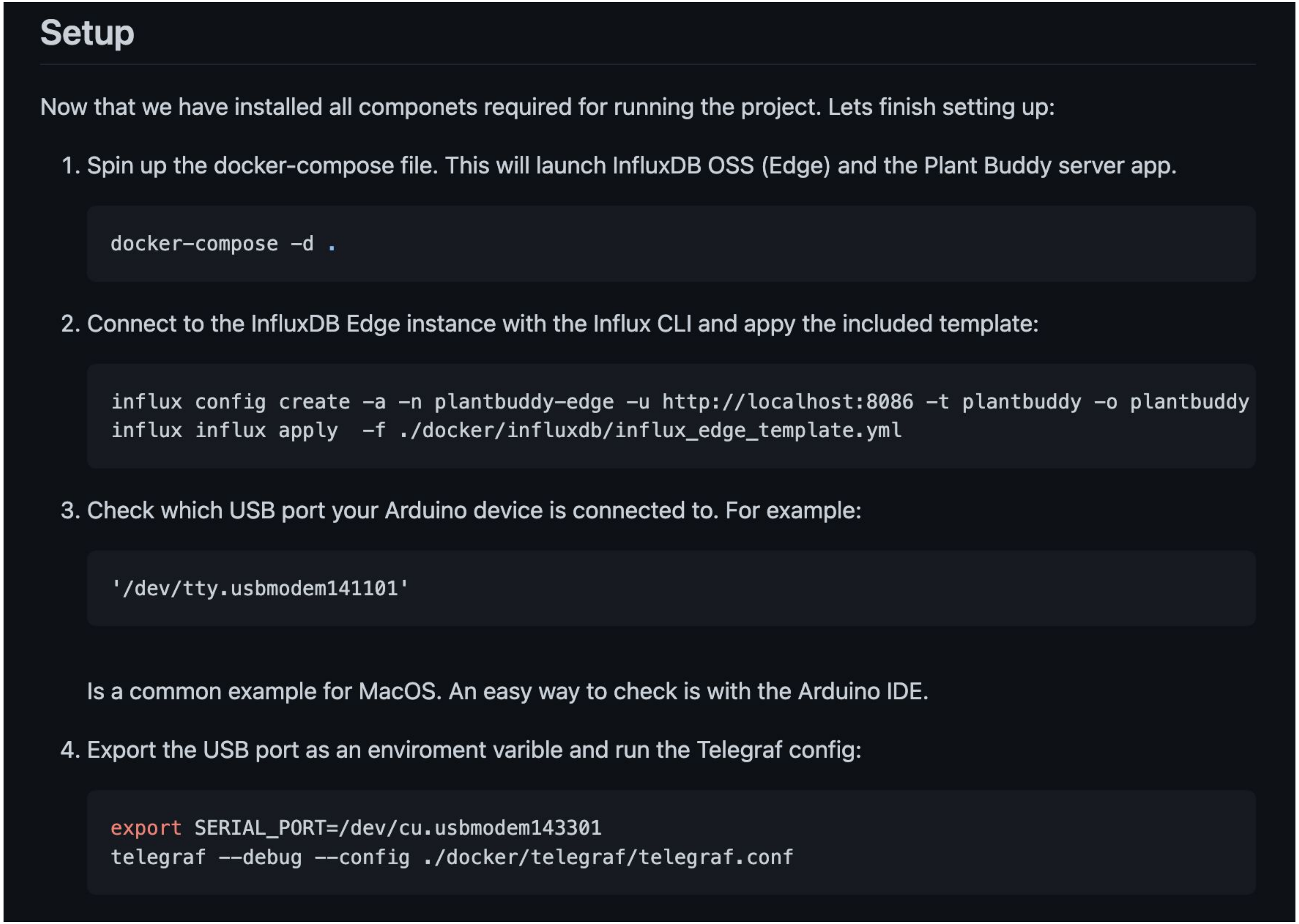
Enables:

- Raw data replication
- Downsampling
- Eventual consistency

Build:

- Distributed databases
- Hybrid apps
- ML pipelines

API	CLI
/api/v2/remotes	`influx remote [create,delete,list,update]`
/api/v2/replications	`influx replication [create,delete,list,update]`



Setup

Now that we have installed all componets required for running the project. Lets finish setting up:

1. Spin up the docker-compose file. This will launch InfluxDB OSS (Edge) and the Plant Buddy server app.

   ```
   docker-compose -d .
   ```

2. Connect to the InfluxDB Edge instance with the Influx CLI and appy the included template:

   ```
   influx config create -a -n plantbuddy-edge -u http://localhost:8086 -t plantbuddy -o plantbuddy
   influx influx apply  -f ./docker/influxdb/influx_edge_template.yml
   ```

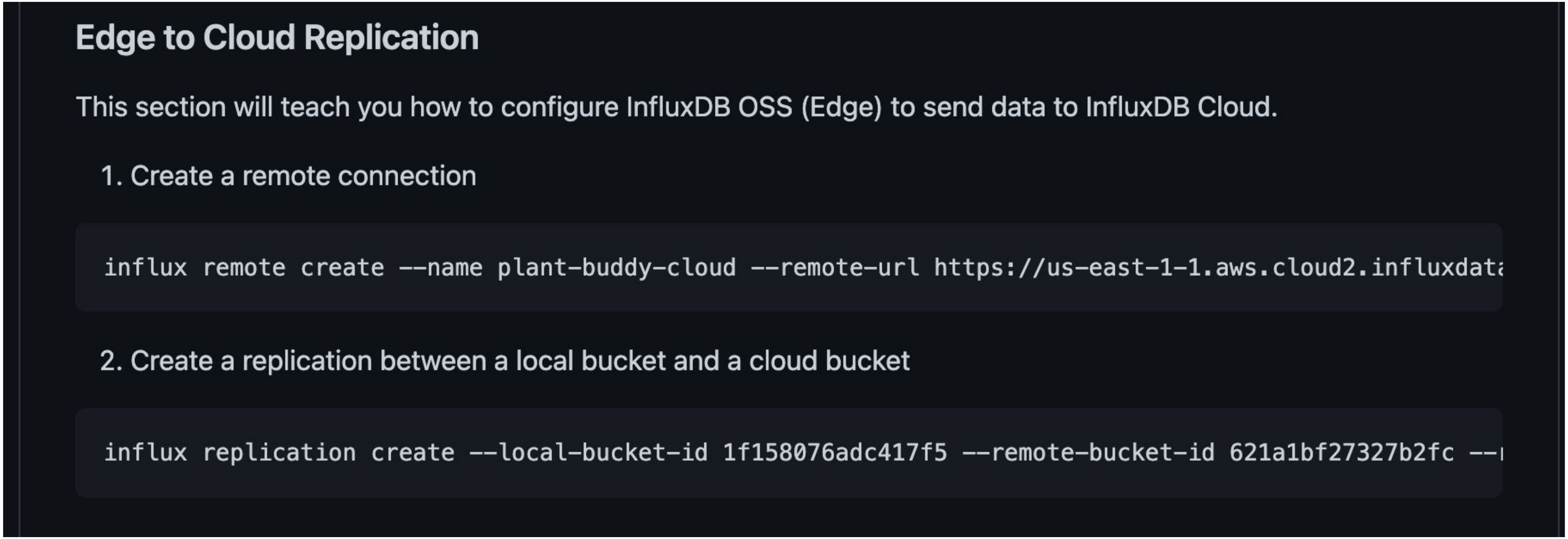
3. Check which USB port your Arduino device is connected to. For example:

   ```
   '/dev/tty.usbmodem141101'
   ```

   Is a common example for MacOS. An easy way to check is with the Arduino IDE.

4. Export the USB port as an enviroment varible and run the Telegraf config:

   ```
   export SERIAL_PORT=/dev/cu.usbmodem143301
   telegraf --debug --config ./docker/telegraf/telegraf.conf
   ```



Edge to Cloud Replication

This section will teach you how to configure InfluxDB OSS (Edge) to send data to InfluxDB Cloud.

1. Create a remote connection

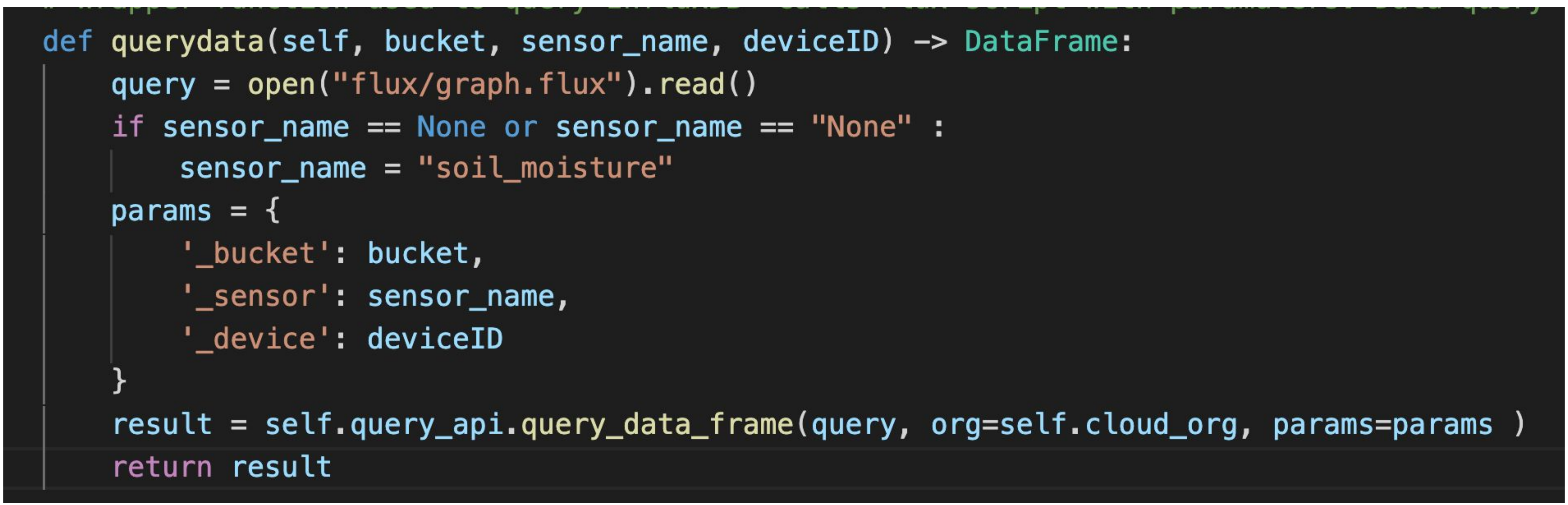
```
influx remote create --name plant-buddy-cloud --remote-url https://us-east-1-1.aws.cloud2.influxdata
```

2. Create a replication between a local bucket and a cloud bucket

```
influx replication create --local-bucket-id 1f158076adc417f5 --remote-bucket-id 621a1bf27327b2fc --
```

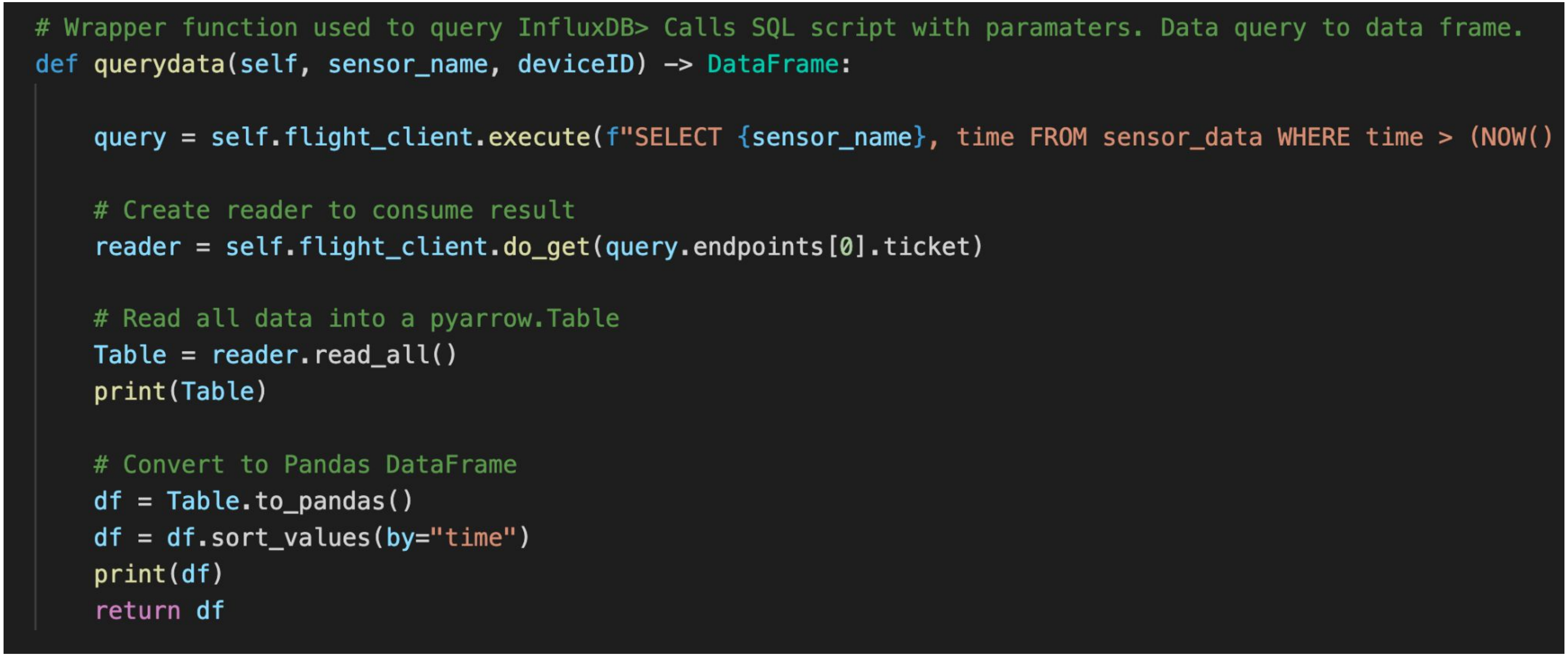
Data Request & Visualization

Query data from Influx



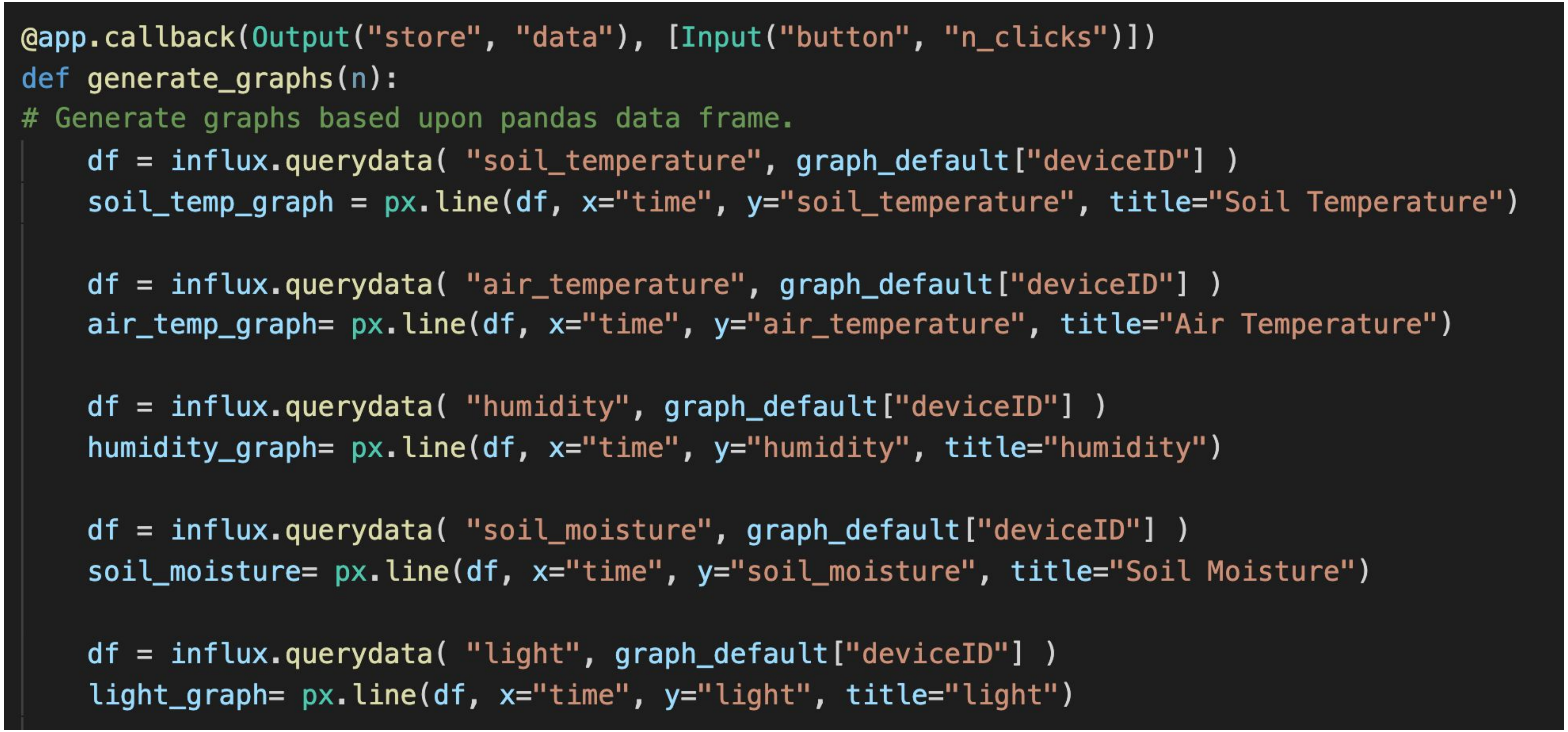
```python
# wrapper function used to query InfluxDB  calls Flux script with parameters: data query
def querydata(self, bucket, sensor_name, deviceID) -> DataFrame:
    query = open("flux/graph.flux").read()
    if sensor_name == None or sensor_name == "None" :
        sensor_name = "soil_moisture"
    params = {
        '_bucket': bucket,
        '_sensor': sensor_name,
        '_device': deviceID
    }
    result = self.query_api.query_data_frame(query, org=self.cloud_org, params=params )
    return result
```

Query SQL from Influx



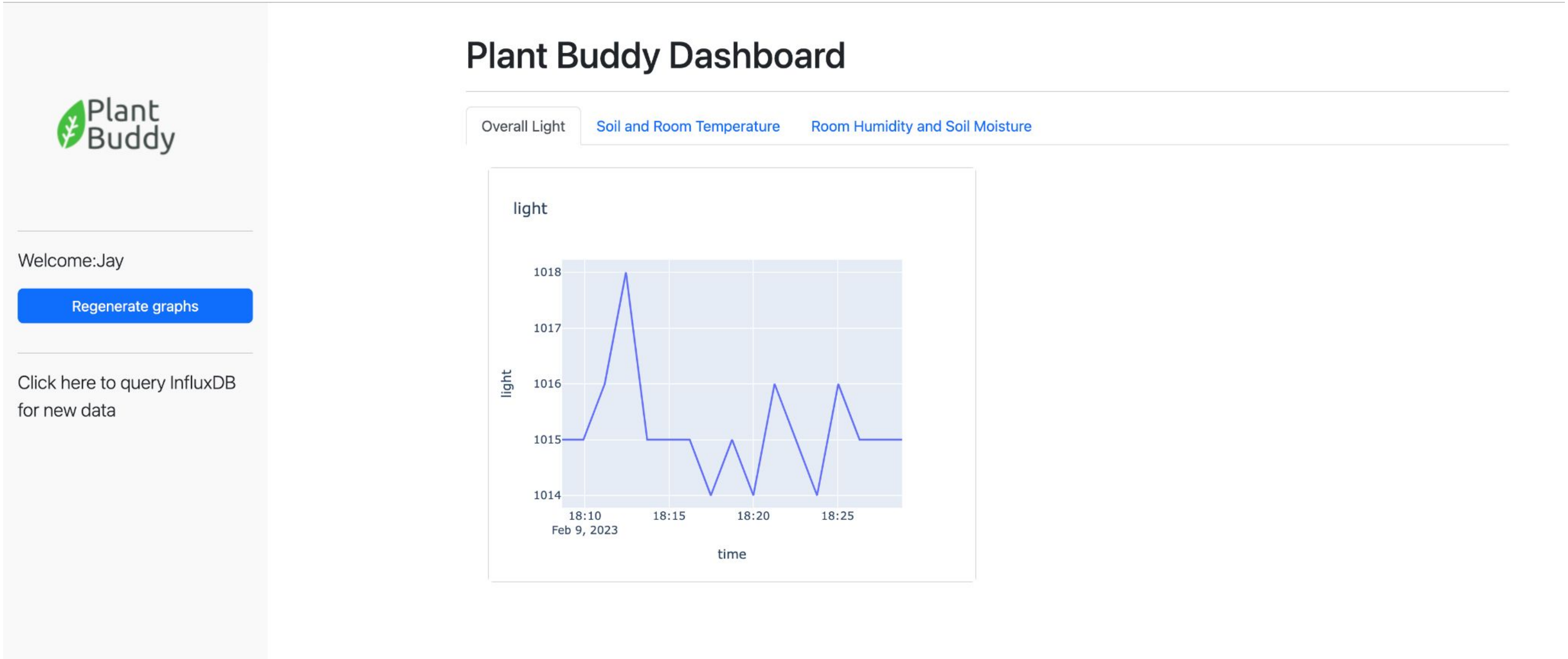
```
# Wrapper function used to query InfluxDB> Calls SQL script with paramaters. Data query to data frame.
def querydata(self, sensor_name, deviceID) -> DataFrame:

    query = self.flight_client.execute(f"SELECT {sensor_name}, time FROM sensor_data WHERE time > (NOW()

    # Create reader to consume result
    reader = self.flight_client.do_get(query.endpoints[0].ticket)

    # Read all data into a pyarrow.Table
    Table = reader.read_all()
    print(Table)

    # Convert to Pandas DataFrame
    df = Table.to_pandas()
    df = df.sort_values(by="time")
    print(df)
    return df
```

Graph the Data



```
@app.callback(Output("store", "data"), [Input("button", "n_clicks")])
def generate_graphs(n):
# Generate graphs based upon pandas data frame.
    df = influx.querydata( "soil_temperature", graph_default["deviceID"] )
    soil_temp_graph = px.line(df, x="time", y="soil_temperature", title="Soil Temperature")

    df = influx.querydata( "air_temperature", graph_default["deviceID"] )
    air_temp_graph= px.line(df, x="time", y="air_temperature", title="Air Temperature")

    df = influx.querydata( "humidity", graph_default["deviceID"] )
    humidity_graph= px.line(df, x="time", y="humidity", title="humidity")

    df = influx.querydata( "soil_moisture", graph_default["deviceID"] )
    soil_moisture= px.line(df, x="time", y="soil_moisture", title="Soil Moisture")

    df = influx.querydata( "light", graph_default["deviceID"] )
    light_graph= px.line(df, x="time", y="light", title="light")
```

Overall Light



Plant Buddy

Welcome:Jay

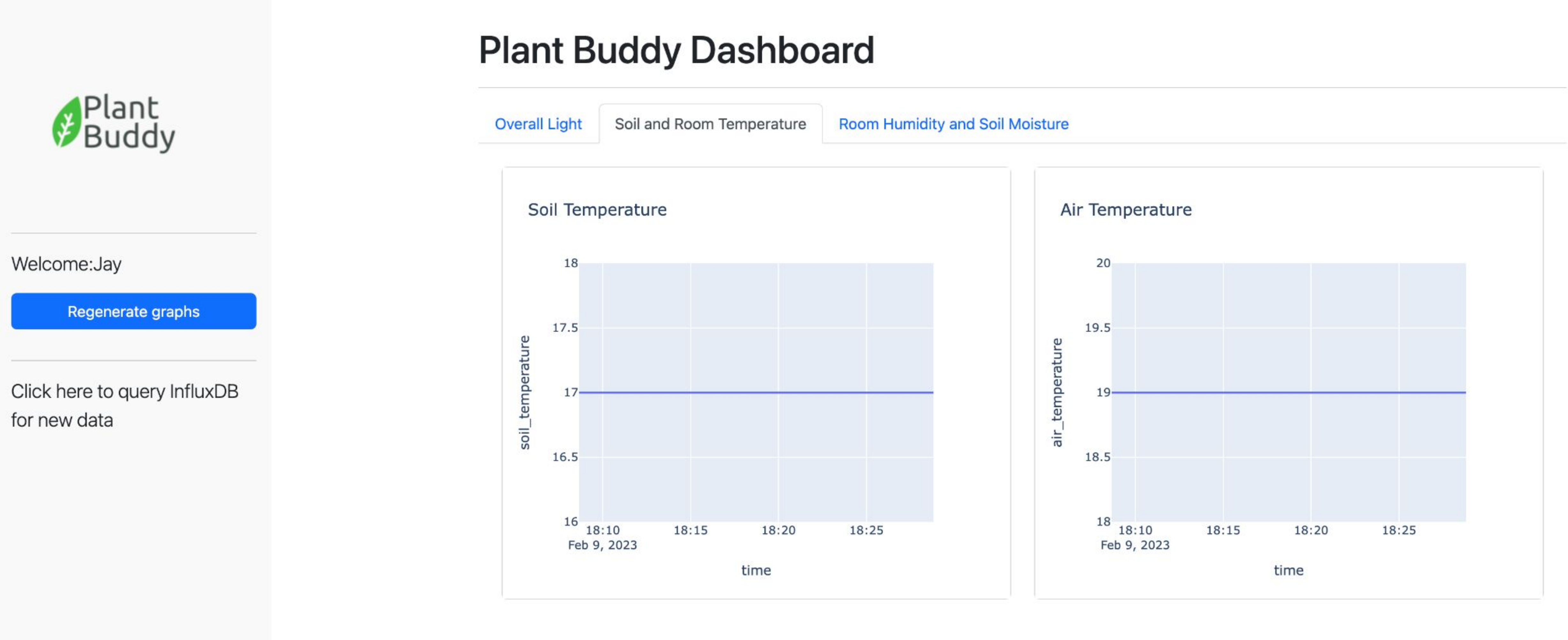
Regenerate graphs

Click here to query InfluxDB for new data

Plant Buddy Dashboard

Overall Light | Soil and Room Temperature | Room Humidity and Soil Moisture

Soil and Room Temperature

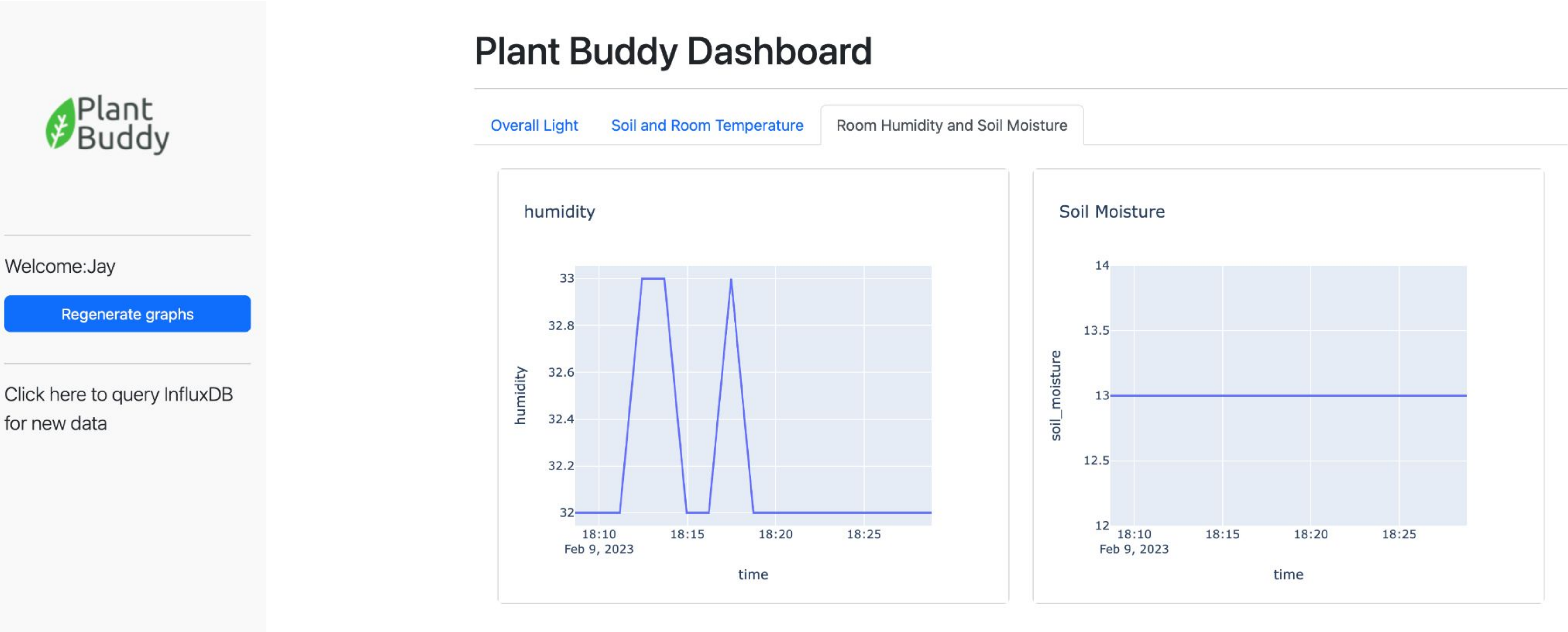


Plant Buddy Dashboard

Welcome:Jay

Click here to query InfluxDB for new data

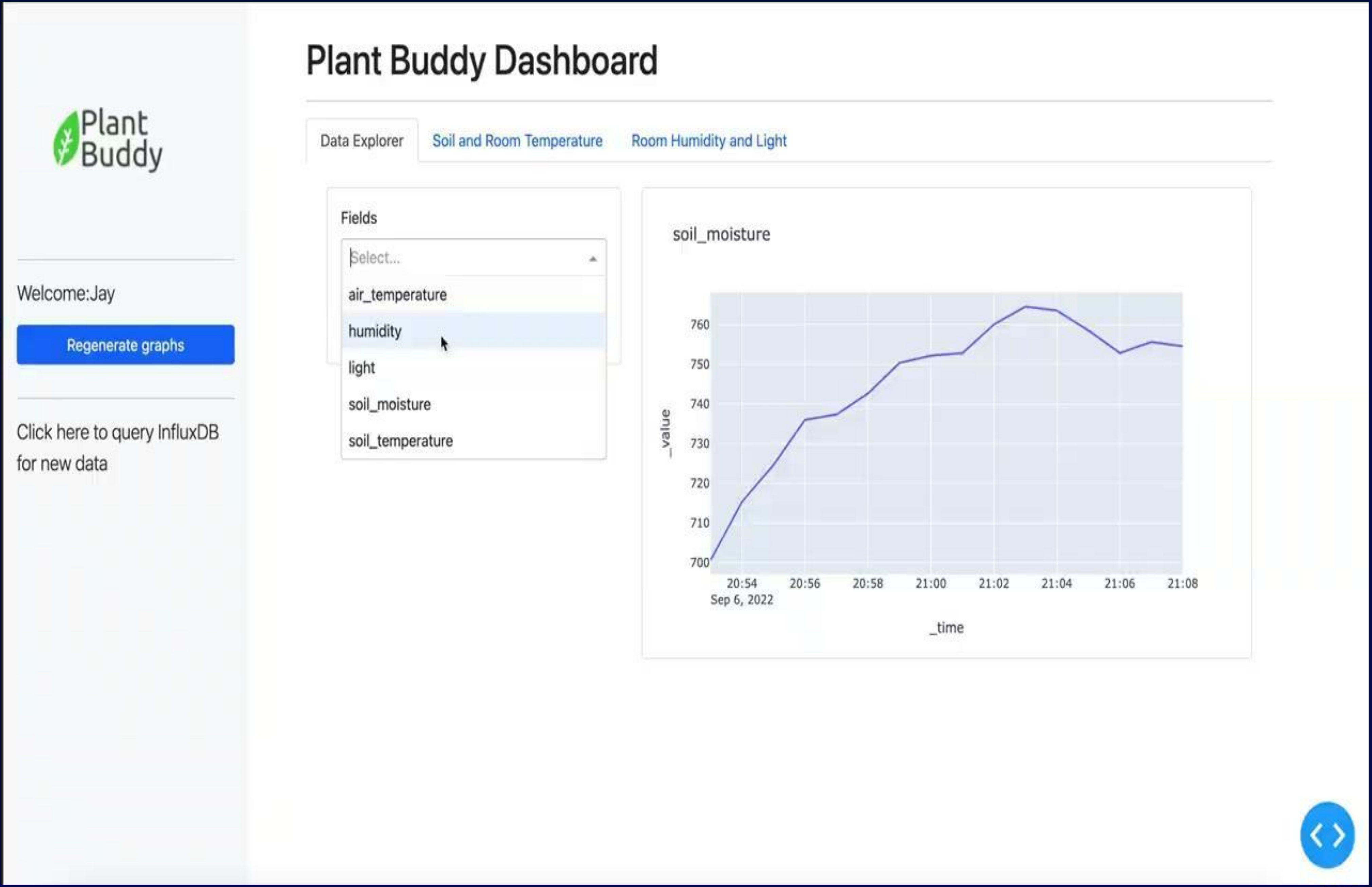
Room Humidity and Soil Moisture



Welcome:Jay

Click here to query InfluxDB for new data

Plant Buddy Dashboard



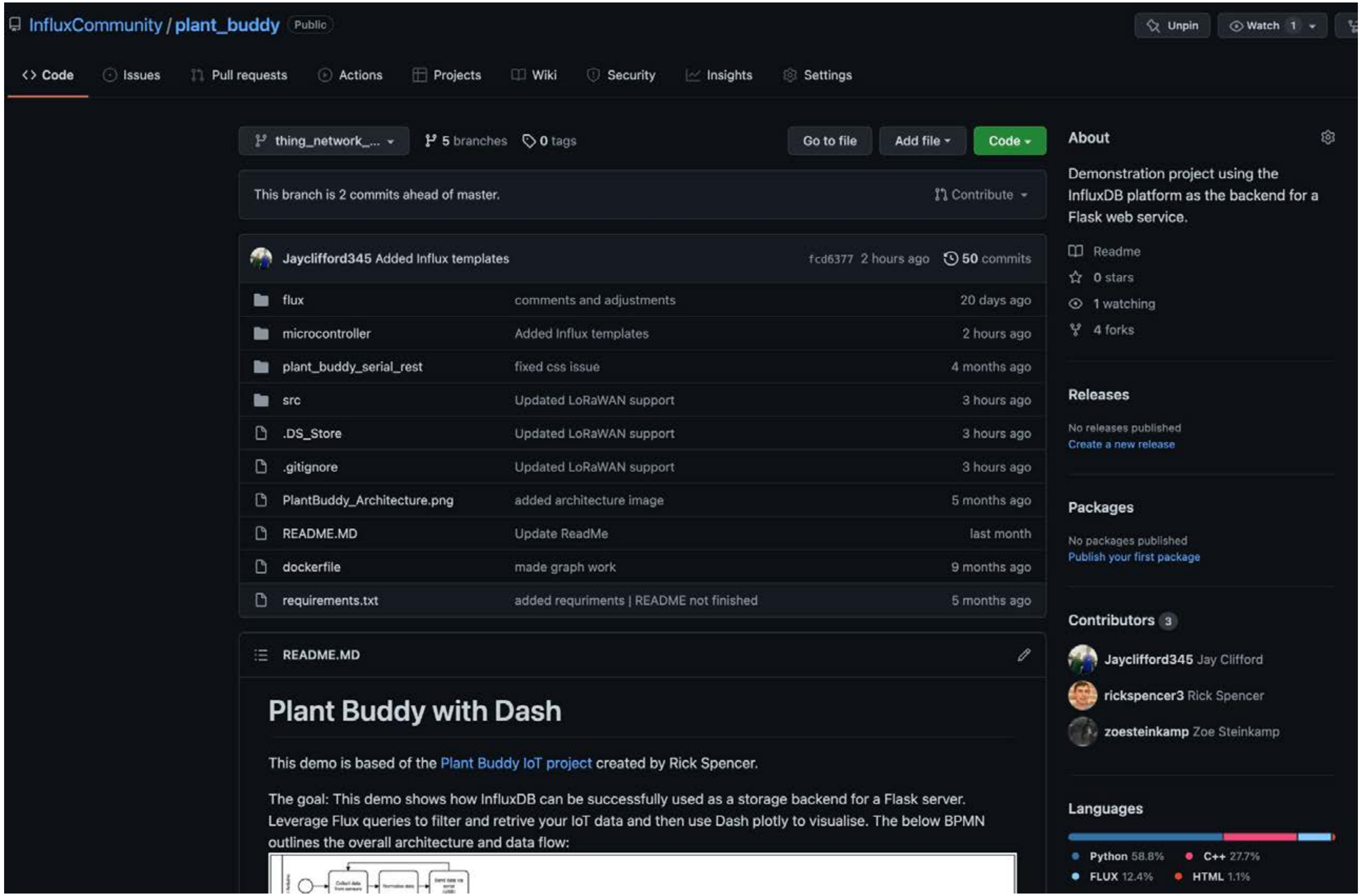
Plant Buddy Dashboard

Welcome:Jay

Click here to query InfluxDB for new data

Further Resources

Try it yourself



InfluxCommunity / plant_buddy Public

5 branches 0 tags

This branch is 2 commits ahead of master. Contribute

Jayclifford345 Added Influx templates fcd6377 2 hours ago 50 commits

flux	comments and adjustments	20 days ago
microcontroller	Added Influx templates	2 hours ago
plant_buddy_serial_rest	fixed css issue	4 months ago
src	Updated LoRaWAN support	3 hours ago
.DS_Store	Updated LoRaWAN support	3 hours ago
.gitignore	Updated LoRaWAN support	3 hours ago
PlantBuddy_Architecture.png	added architecture image	5 months ago
README.MD	Update ReadMe	last month
dockerfile	made graph work	9 months ago
requirements.txt	added requriments | README not finished	5 months ago

README.MD

Plant Buddy with Dash

This demo is based of the Plant Buddy IoT project created by Rick Spencer.

The goal: This demo shows how InfluxDB can be successfully used as a storage backend for a Flask server. Leverage Flux queries to filter and retrive your IoT data and then use Dash plotly to visualise. The below BPMN outlines the overall architecture and data flow:

About

Demonstration project using the InfluxDB platform as the backend for a Flask web service.

Readme
0 stars
1 watching
4 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 3

Jayclifford345 Jay Clifford
rickspencer3 Rick Spencer
zoesteinkamp Zoe Steinkamp

Languages

Python 58.8% C++ 27.7%
FLUX 12.4% HTML 1.1%

GITHUB

https://github.com/InfluxCommunity/plant_buddy

https://github.com/InfluxCommunity/plant_buddy_iox

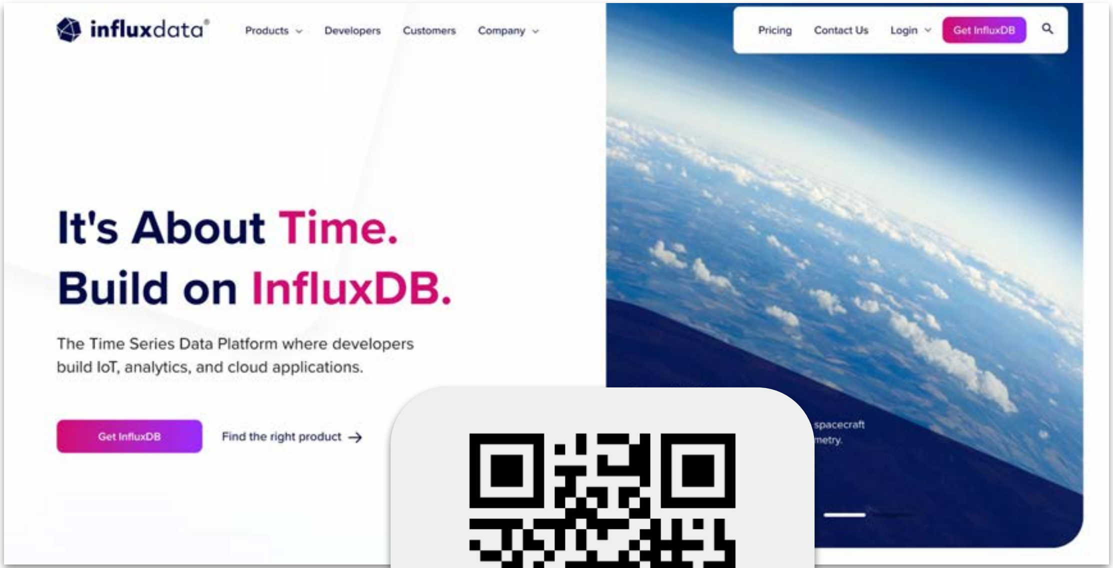
InfluxDB Community Slack workspace

Please join us in the InfluxDB Community Slack at www.influxdata.com/slack.

To participate in conversations, join the #influxdb_iox channel.

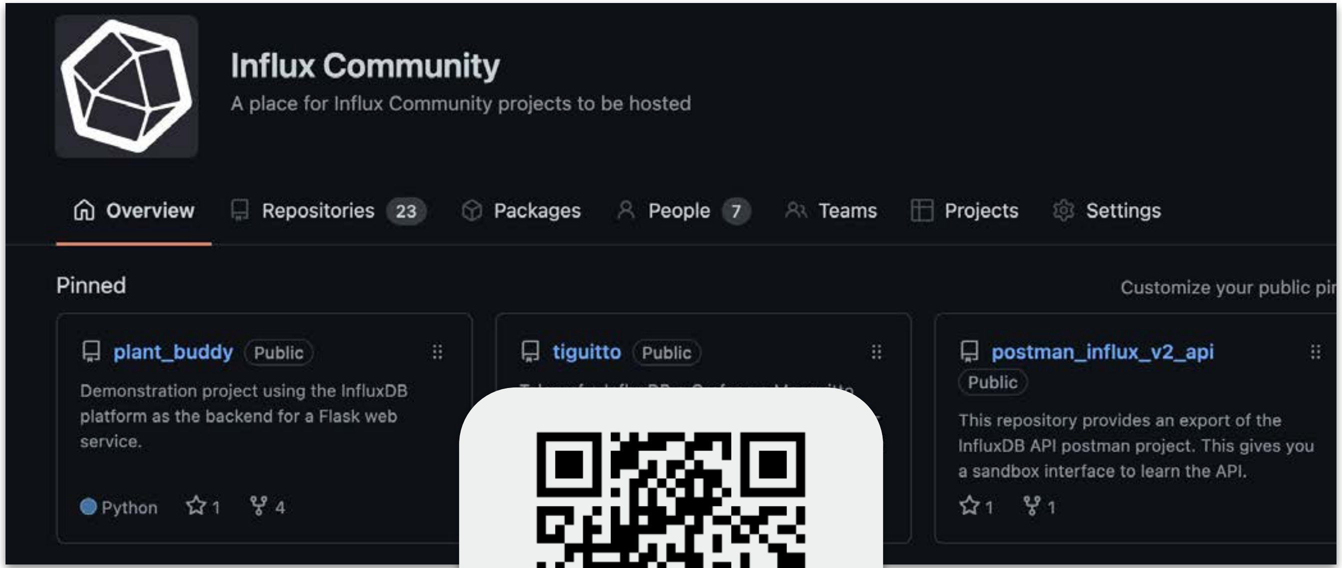
Try it yourself

Get Started



It's About Time.
Build on InfluxDB.

The Time Series Data Platform where developers build IoT, analytics, and cloud applications.



Influx Community

A place for Influx Community projects to be hosted

plant_buddy Public
Demonstration project using the InfluxDB platform as the backend for a Flask web service.
Python ☆ 1 4

tiguitto Public

postman_influx_v2_api Public
This repository provides an export of the InfluxDB API postman project. This gives you a sandbox interface to learn the API.
☆ 1 1

Further Resources

Get started: influxdata.com/cloud
Forums: community.influxdata.com
Slack: influxcommunity.slack.com
GH: github.com/InfluxCommunity
Book: awesome.influxdata.com
Docs: docs.influxdata.com
Blogs: influxdata.com/blog
InfluxDB University:
influxdata.com/university

Questions with a side of answers?