



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

Zoe Steinkamp



Zoe Steinkamp
Developer Advocate



LinkedIn



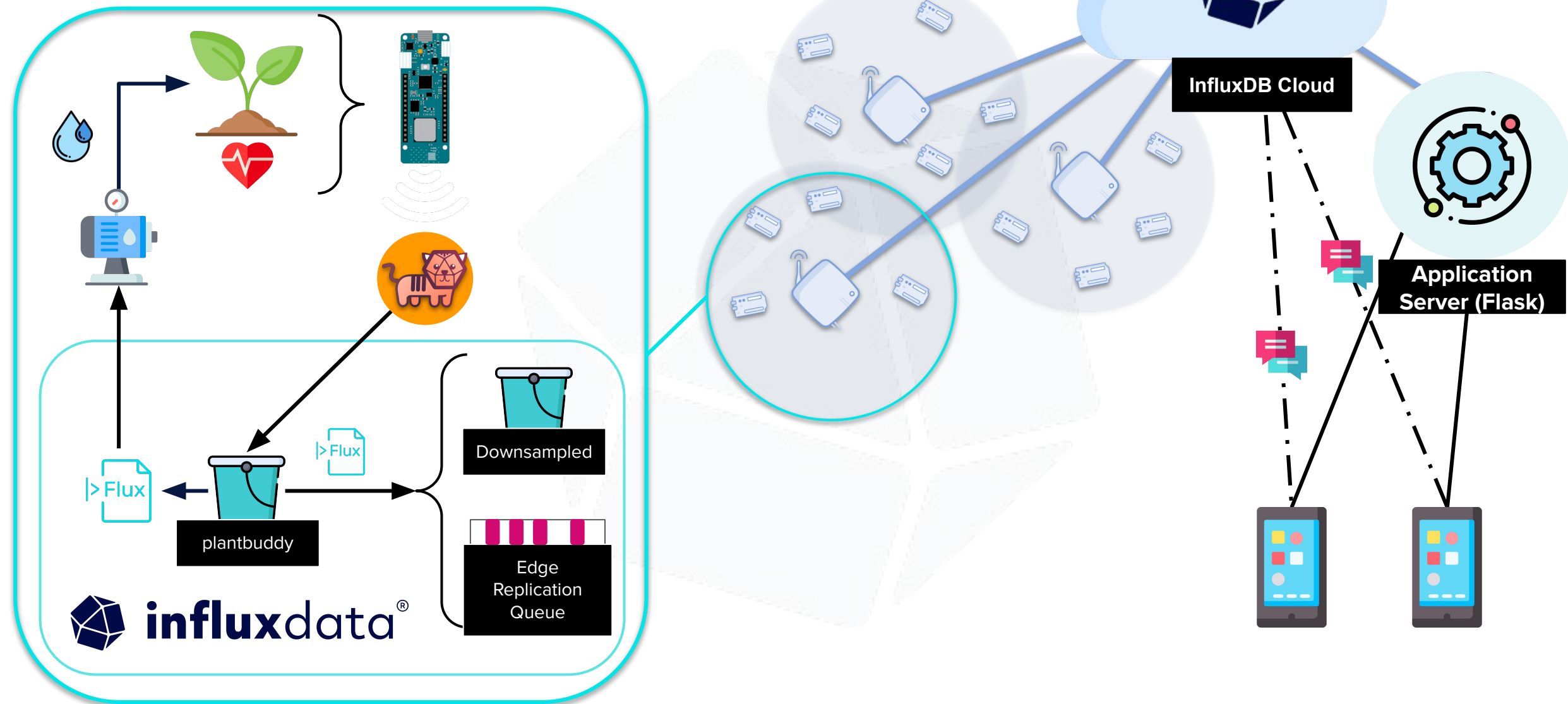
The Overview:

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

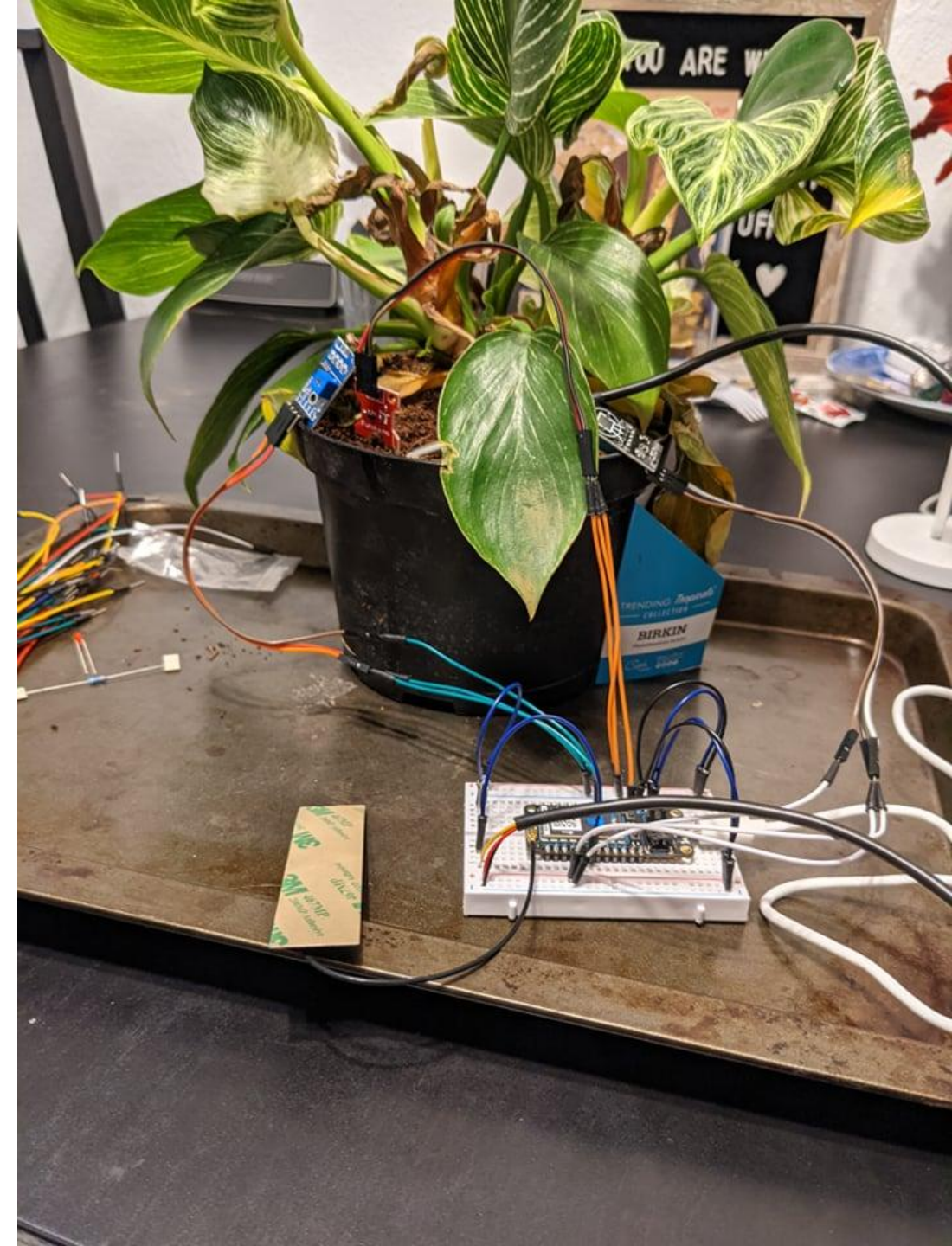
Set Up IOT Device

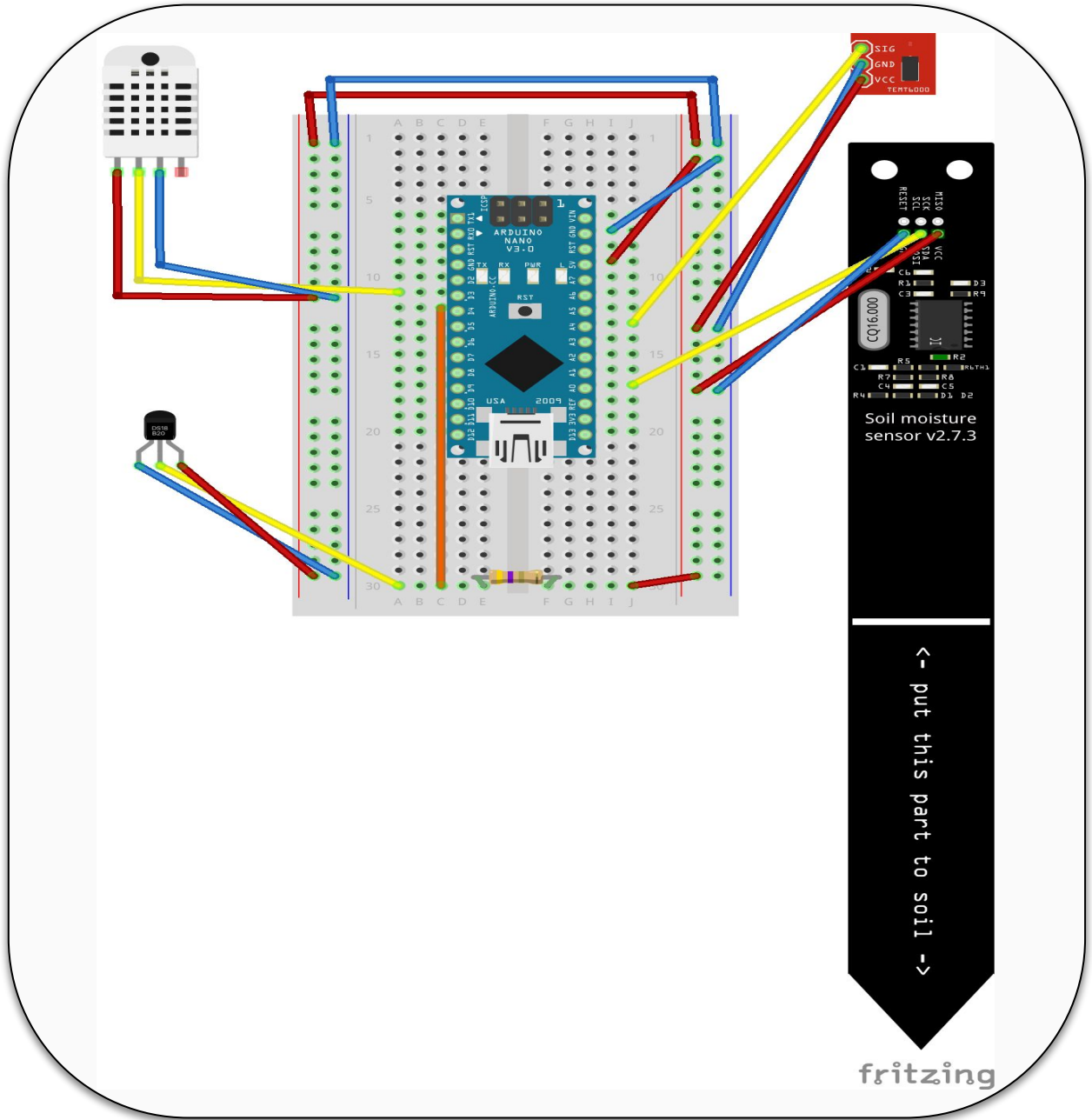
IoT Edge Example



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



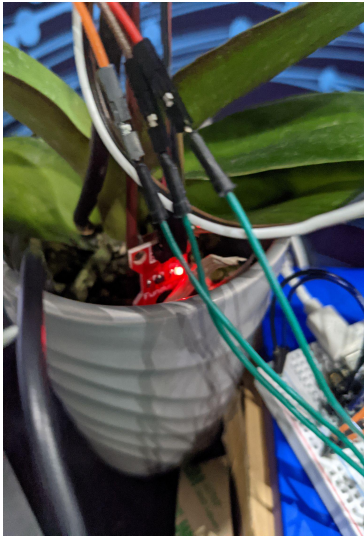


Schematics & Sensors

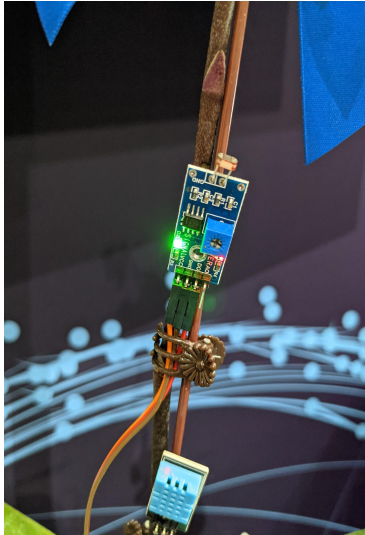
Temperature & Humidity



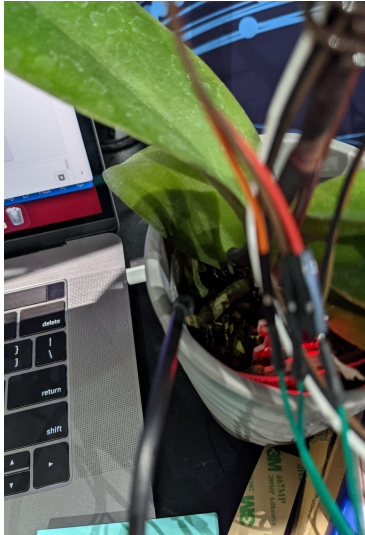
Soil Moisture



Light



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



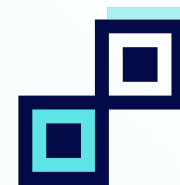
Telegraf for Ingestion



**The open-source
agent for
collecting metrics**



**Driven by the
community (600+
contributors)**



**Simple to
configure,
extremely flexible**

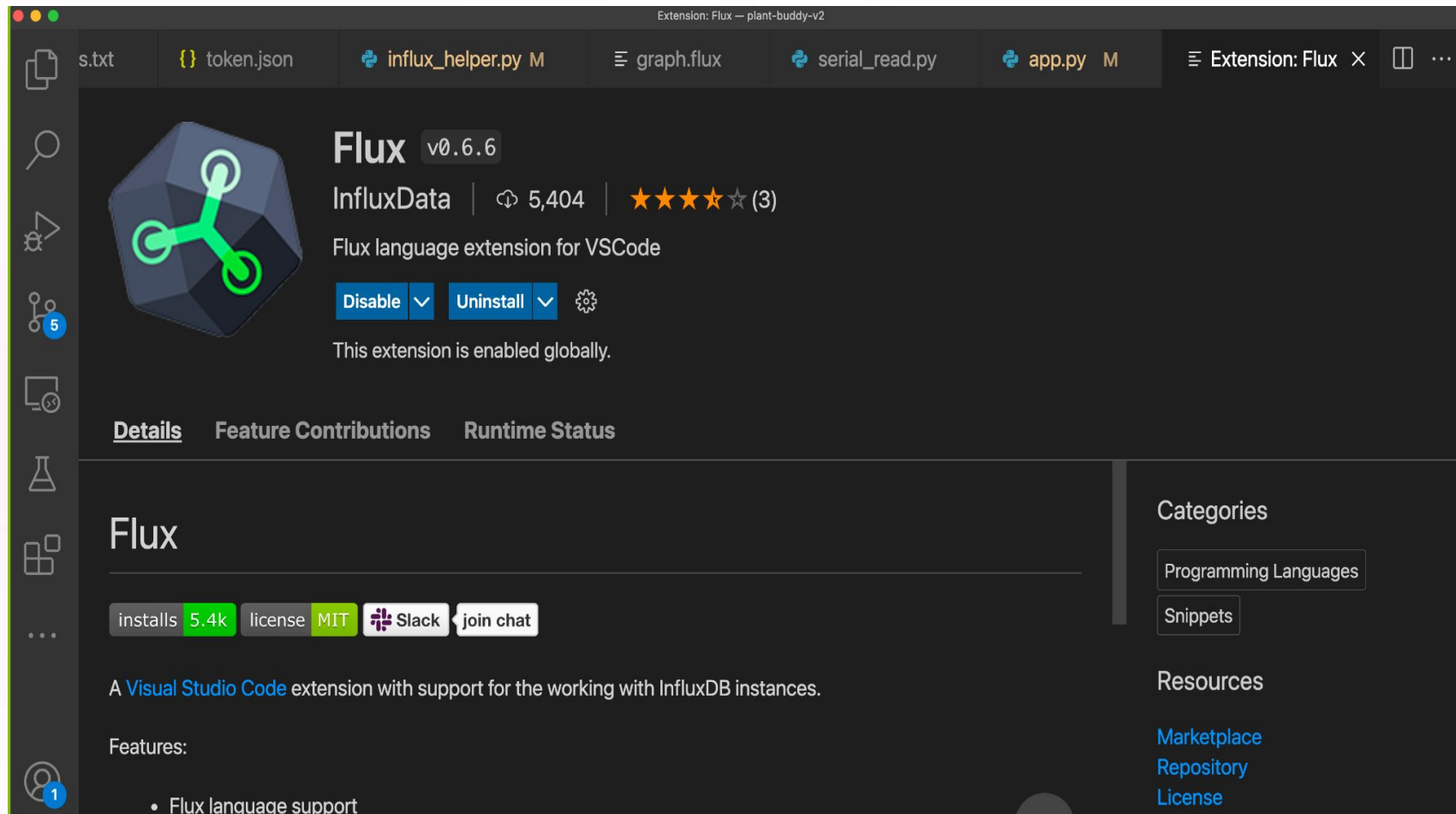
Client Libraries

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



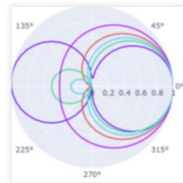
Plotly for Graphing

Fundamentals

[More Fundamentals »](#)



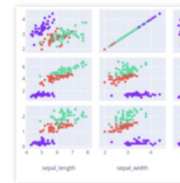
The Figure Data Structure



Creating and Updating Figures



Displaying Figures



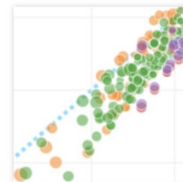
Plotly Express



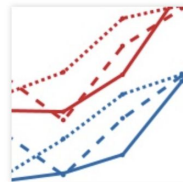
Analytical Apps with Dash

Basic Charts

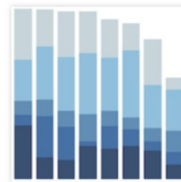
[More Basic Charts »](#)



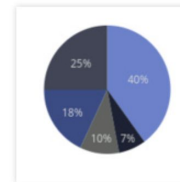
Scatter Plots



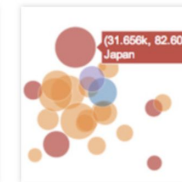
Line Charts



Bar Charts



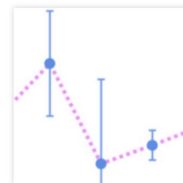
Pie Charts



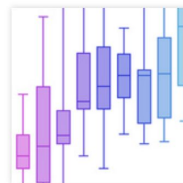
Bubble Charts

Statistical Charts

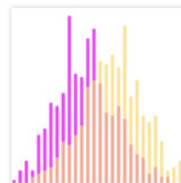
[More Statistical Charts »](#)



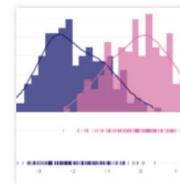
Error Bars



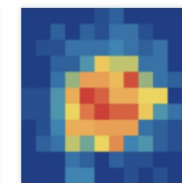
Box Plots



Histograms



Distplots



2D Histograms



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



DOCUMENT

- High throughput
- Large document



SEARCH

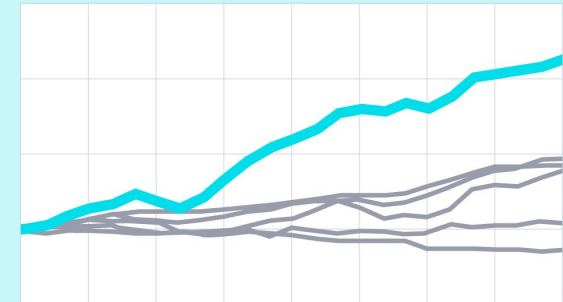
- Distributed search
- Logs
- Geo



TIME SERIES

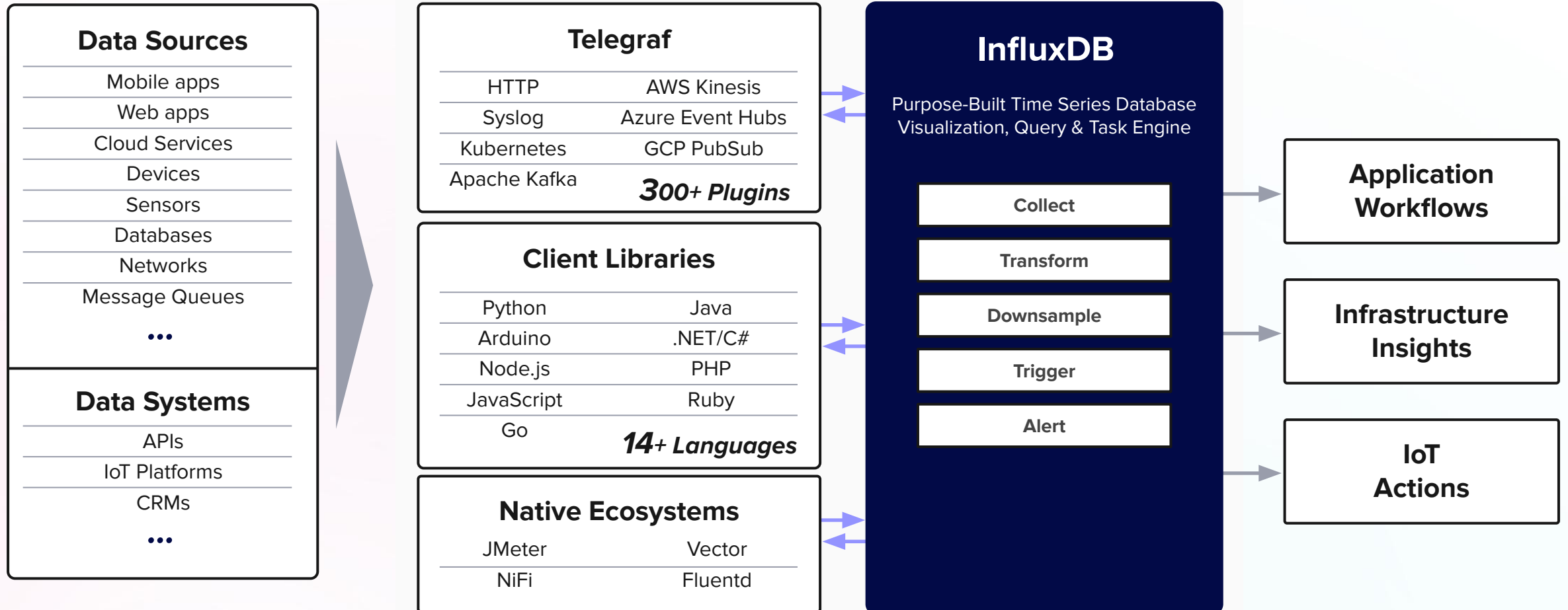
- Events, metrics, time stamped
- for IoT, analytics, cloud native

Time Series Category Trend



InfluxDB + Telegraf + Flux

InfluxDB Platform



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](#)

z



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: ecfcd9988408a9b

_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

```
# 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",  
    ]  
]
```

Line Protocol

ALL time series data is written to InfluxDB using Line Protocol, and uses the following format:

```
<measurement>[, <tag-key>=<tag-value>] [<field-key>=<field-value>]  
[unix-nano-timestamp]
```

Measurement	Tag Set	Field Set	Timestamp
cpu_load	hostname=server02, us_west=az	temp=24.5, volts=7	1234567890000000

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

Introducing Flux

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



```
1 import "math"
2
3 bicycles3 = from(bucket: "smartcity")
4   |> range(start:2021-03-01T00:00:00Z, stop: 2021-04-01T00:00:00Z)
5   |> filter(fn: (r) => r._measurement == "city_IoT")
6   |> filter(fn: (r) => r._field == "counter")
7   |> filter(fn: (r) => r.source == "bicycle")
8   |> filter(fn: (r) => r.neighborhood_id == "3")
9   |> aggregateWindow(every: 1h, fn: mean, createEmpty:false)
10
11 bicycles4 = from(bucket: "smartcity")
12   |> range(start:2021-03-01T00:00:00Z, stop: 2021-04-01T00:00:00Z)
13   |> filter(fn: (r) => r._measurement == "city_IoT")
14   |> filter(fn: (r) => r._field == "counter")
15   |> filter(fn: (r) => r.source == "bicycle")
16   |> filter(fn: (r) => r.neighborhood_id == "4")
17   |> aggregateWindow(every: 1h, fn: mean, createEmpty:false)
18
19 join(tables: {neighborhood_3: bicycles3, neighborhood_4: bicycles4}, on: ["_time"], method: "inner")
20   |> keep(columns: ["_time", "_value_neighborhood_3", "_value_neighborhood_4"])
21   |> map(fn: (r) => ({
22     r with
23     difference_value: math.abs(x: (r._value_neighborhood_3 - r._value_neighborhood_4))
24   })))
```

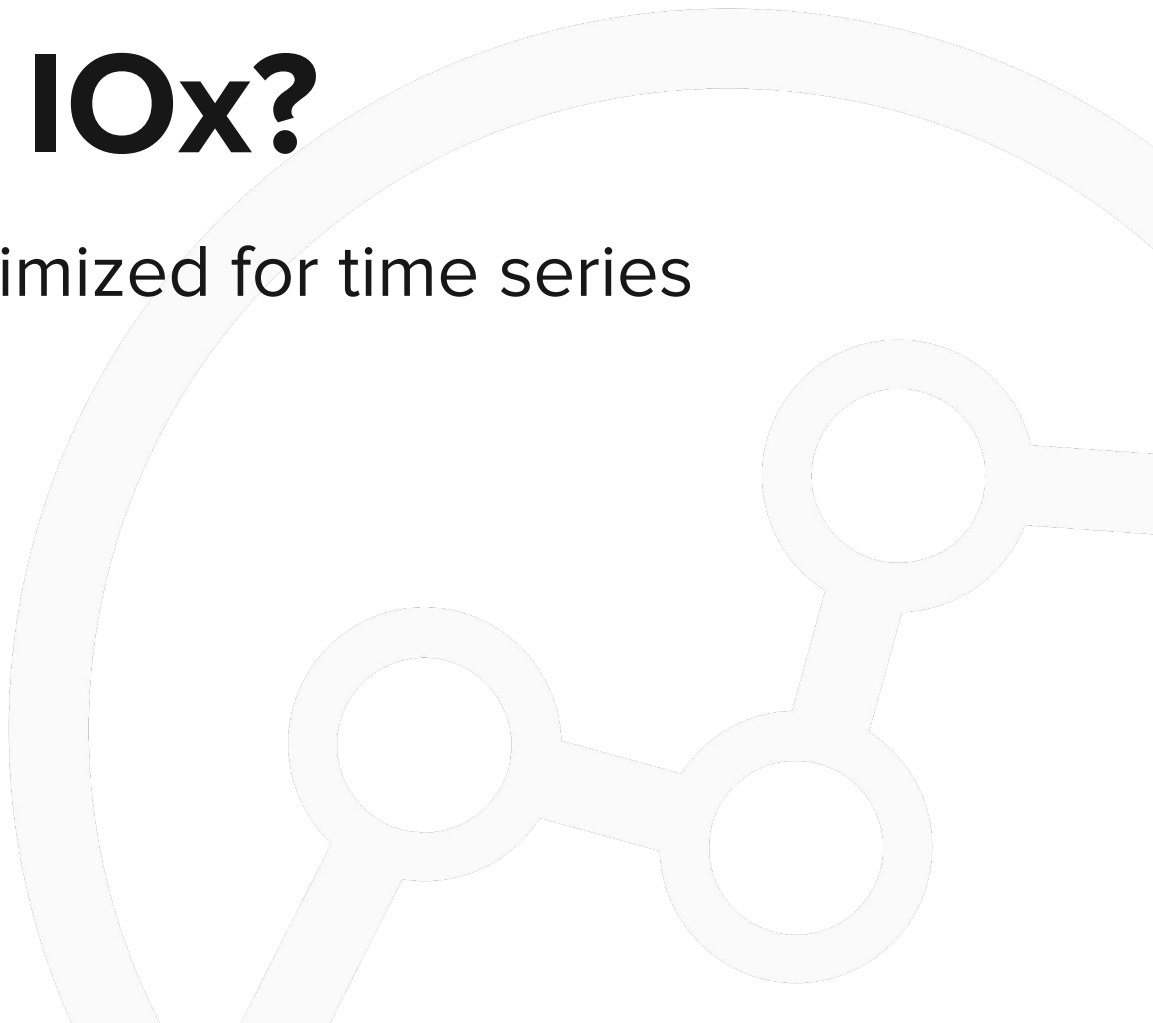
Flux Query

```
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!



What is InfluxDB IOx?

- Cloud columnar database optimized for time series
 - Schema-on-write
 - Object store persistence
 - SQL native
 - InfluxQL (coming soon)
- 

InfluxDB IOx enables

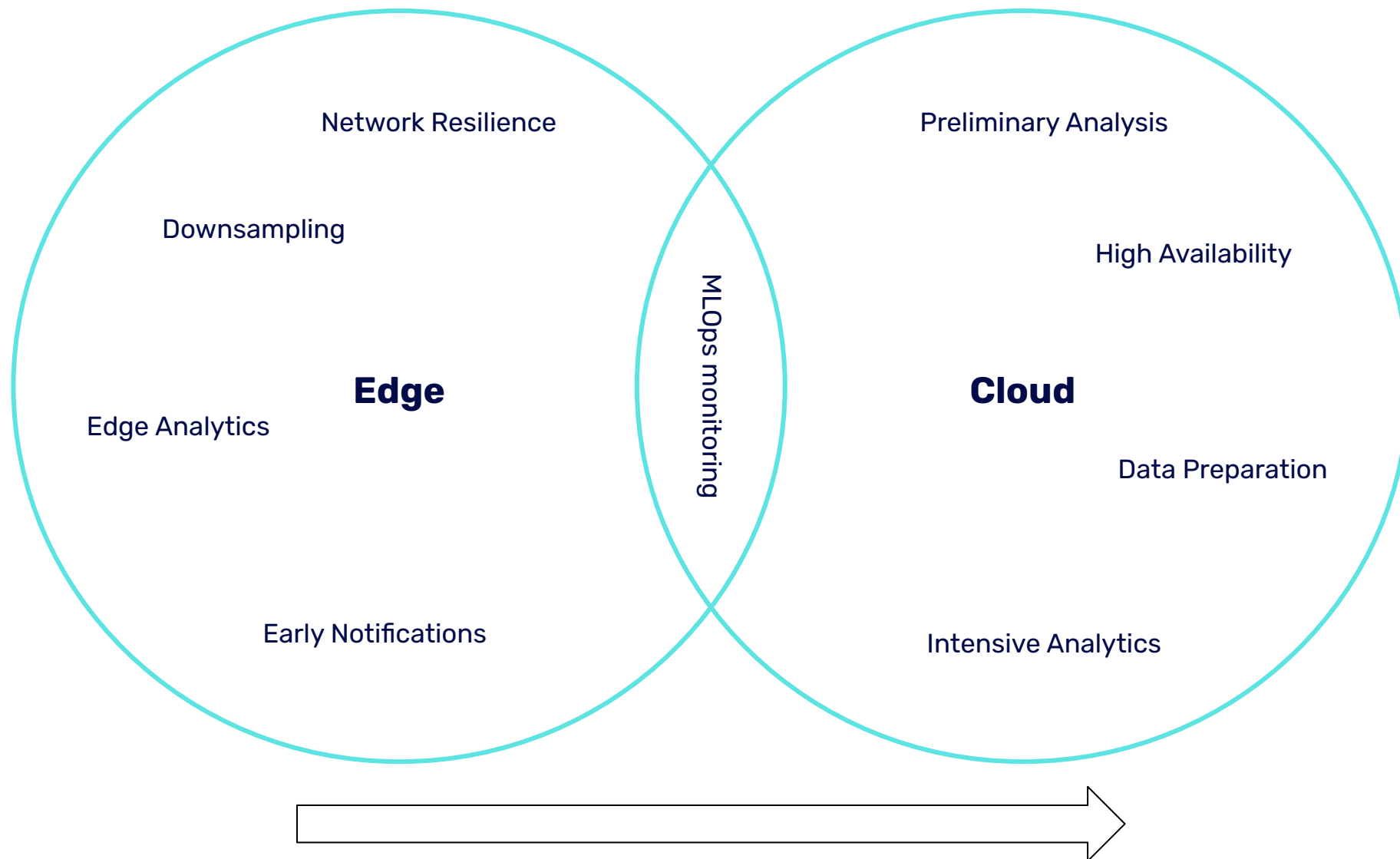
- Unlimited cardinality
- Separate storage from compute
- Separate ingest from query
- Separate query workloads
- Bulk data import & export (coming soon)

Integrations

- FlightSQL python library
- FlightSQL in other languages
- Grafana Flight SQL Plugin
- Superset FlightSQL Plugin
- JDBC driver (coming soon)
- ODBC driver (coming soon)

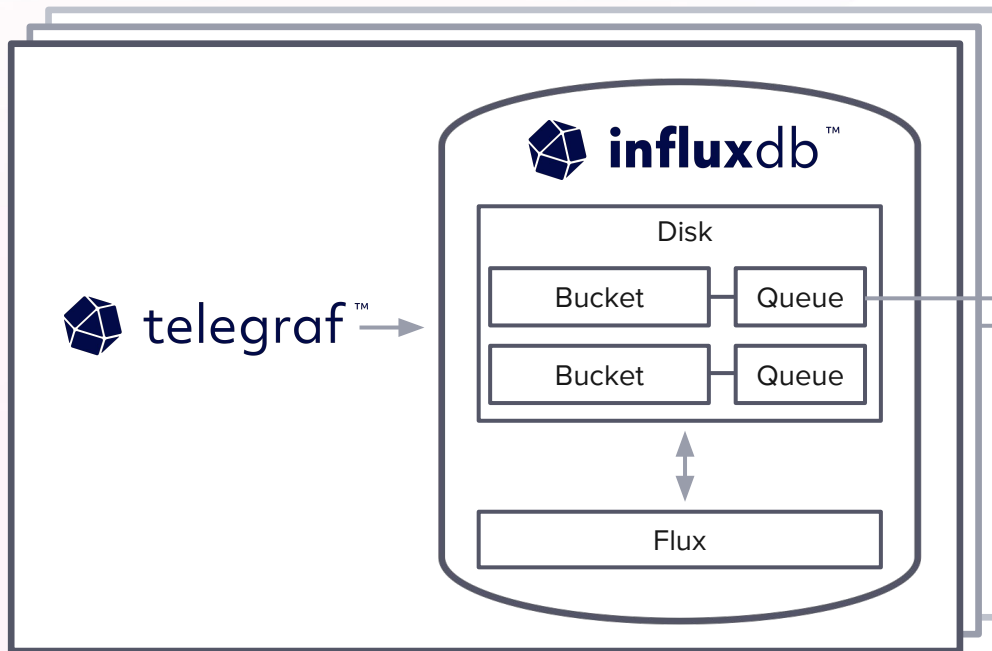


Edge Data Replication

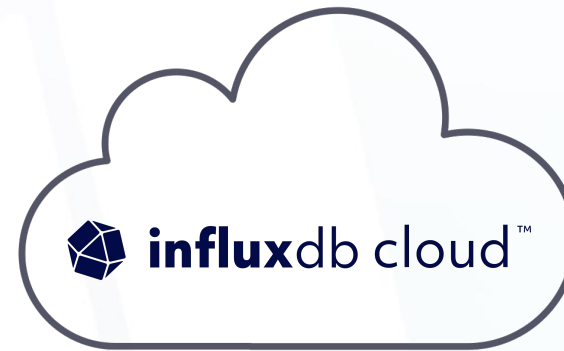


InfluxData Edge Data Replication

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
<code>/api/v2/remotes</code>	<code>influx remote [create,delete,list,update]</code>
<code>/api/v2/replications</code>	<code>influx replication [create,delete,list,update]</code>

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.com
```

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

```
influx replication create --local-bucket-id 1f158076adc417f5 --remote-bucket-id 621a1bf27327b2fc --remote-connection plant-buddy-cloud
```

Data Request & Visualization

Query data from Influx

```
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



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



Welcome:Jay

Regenerate graphs

[Click here to query InfluxDB for new data](#)

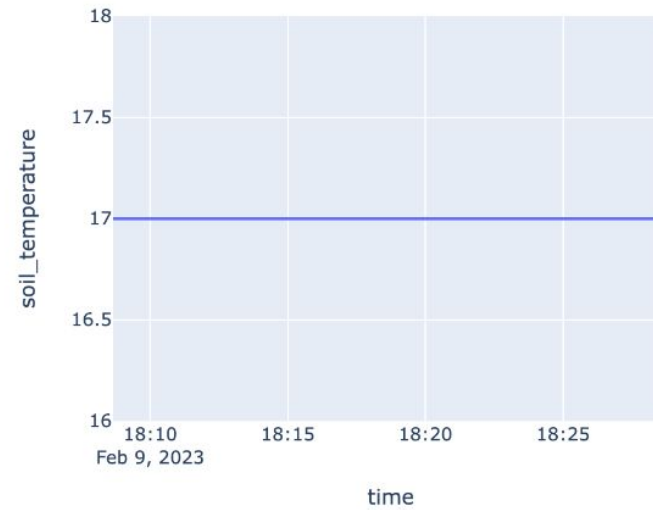
Plant Buddy Dashboard

Overall Light

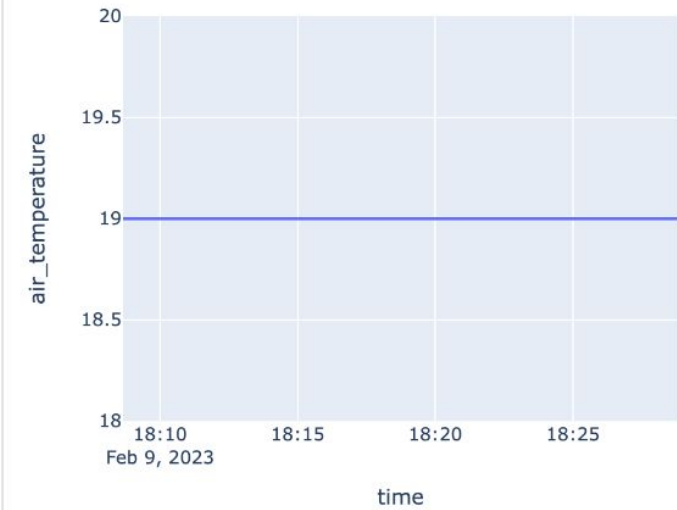
Soil and Room Temperature

Room Humidity and Soil Moisture

Soil Temperature



Air Temperature



Room Humidity and Soil Moisture



Welcome: Jay

Regenerate graphs

[Click here to query InfluxDB for new data](#)

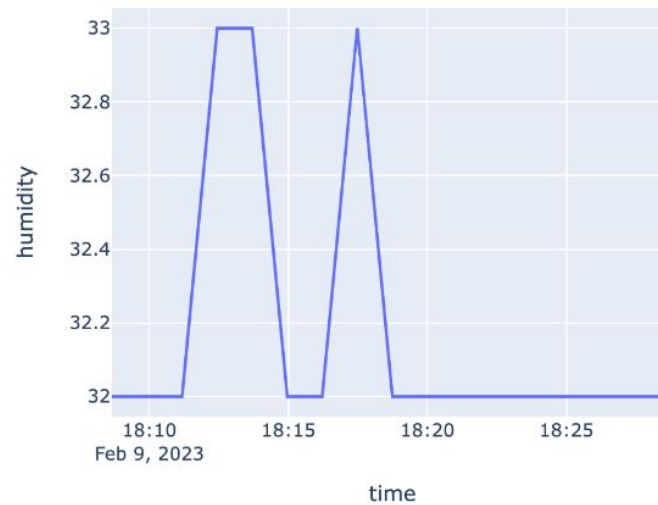
Plant Buddy Dashboard

Overall Light

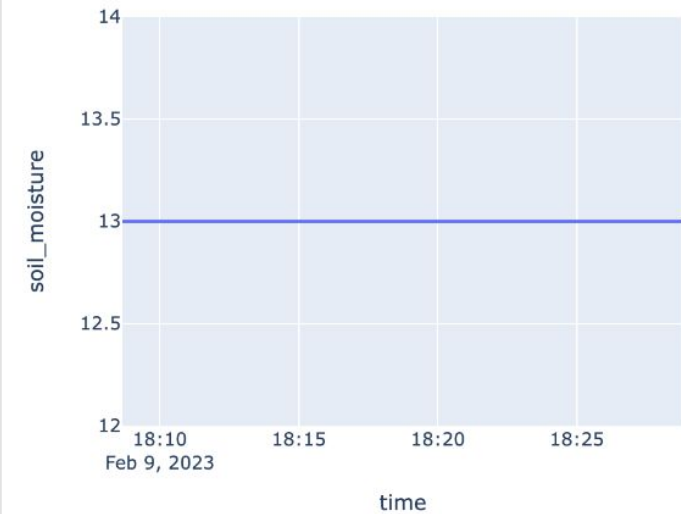
Soil and Room Temperature

Room Humidity and Soil Moisture

humidity



Soil Moisture





Welcome: Jay

Regenerate graphs

[Click here to query InfluxDB for new data](#)

Plant Buddy Dashboard

Data Explorer

Soil and Room Temperature

Room Humidity and Light

Fields

Select...

air_temperature

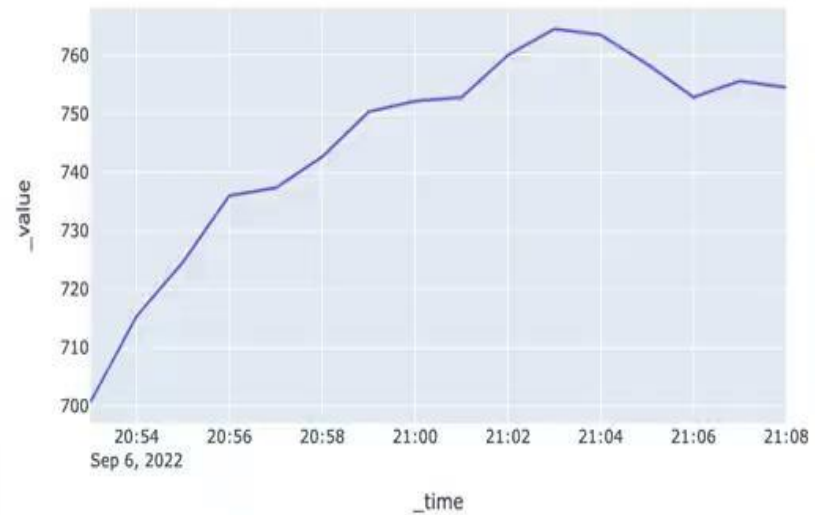
humidity

light

soil_moisture

soil_temperature

soil_moisture



Alerting + Downsampling

Threshold Alert + Slack Notification

```
option task = {name: "PB_Moisture_Test", every: 1h, offset: 5s}

check = {
  _check_id: "local_427a3c8c-6444-478f-848a-f8767eb5e48d",
  _check_name: "PB_Moisture_Test",
  _type: "custom",
  tags: {},
}

notification = {
  _notification_rule_id: "local_427a3c8c-6444-478f-848a-f8767eb5e48d",
  _notification_rule_name: "PB_Moisture_Test Rule",
  _notification_endpoint_id: "local_427a3c8c-6444-478f-848a-f8767eb5e48d",
  _notification_endpoint_name: "PB_Moisture_Test Endpoint",
}

task_data =
  from(bucket: "plantbuddy")
    |> range(start: -1h)
    |> filter(fn: (r) => r["_measurement"] == "sensor_data")
    |> filter(fn: (r) => r["_field"] == "soil_moisture")
    |> filter(fn: (r) => r["device_id"] == "eui-323932326d306512")
    |> last()

trigger = (r) => r["soil_moisture"] < 50
messageFn = (r) =>
  " ${time(v: r._source_timestamp)} Your plant is getting thirsty. Moisture Level is at: ${r.soil_moisture}% ! "

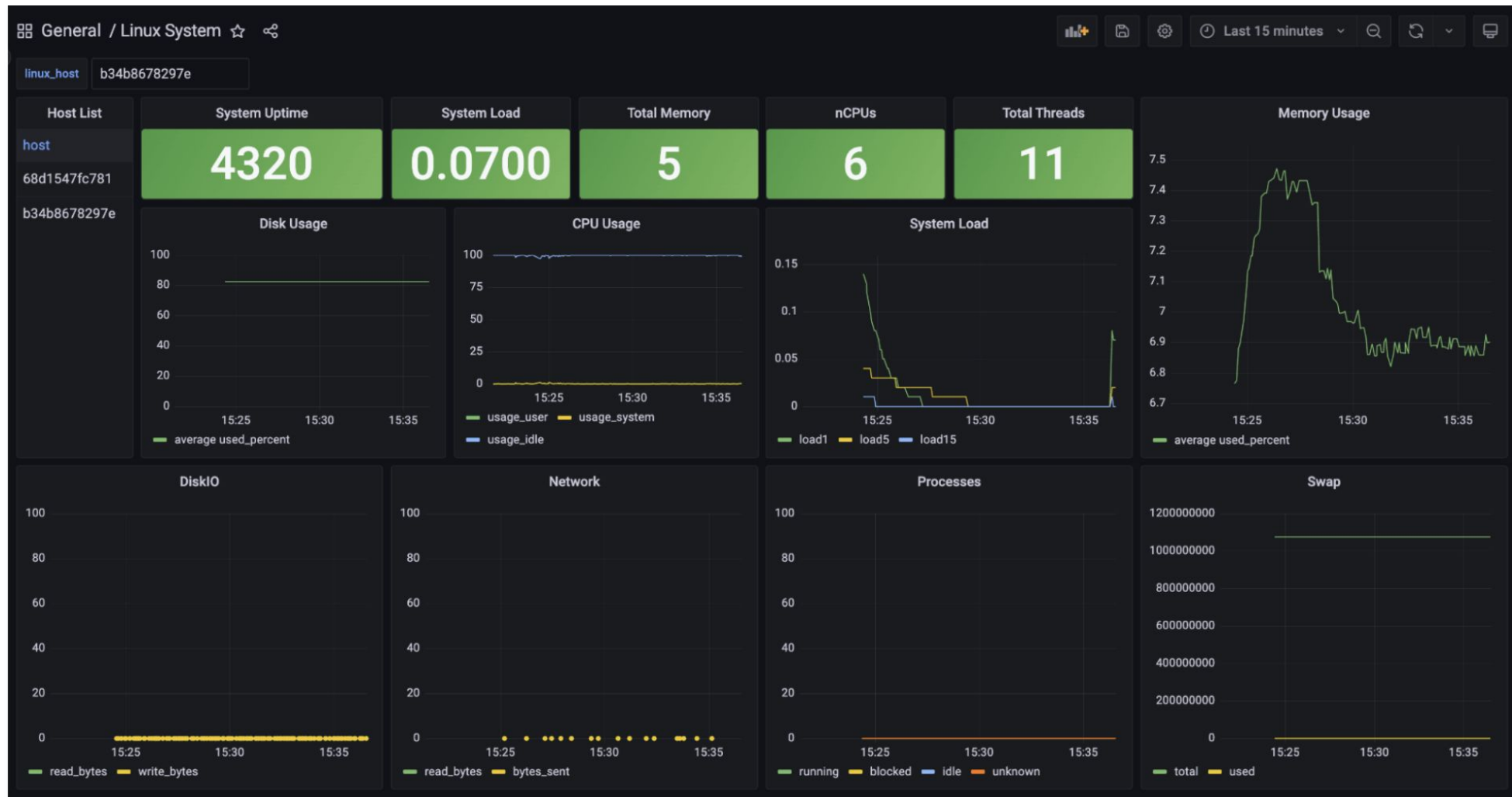
task_data
  |> schema["fieldsAsCols"]()
  |> monitor["check"](data: check, messageFn: messageFn, crit: trigger)
  |> monitor["notify"](
    data: notification
```


Downsampling with Flux

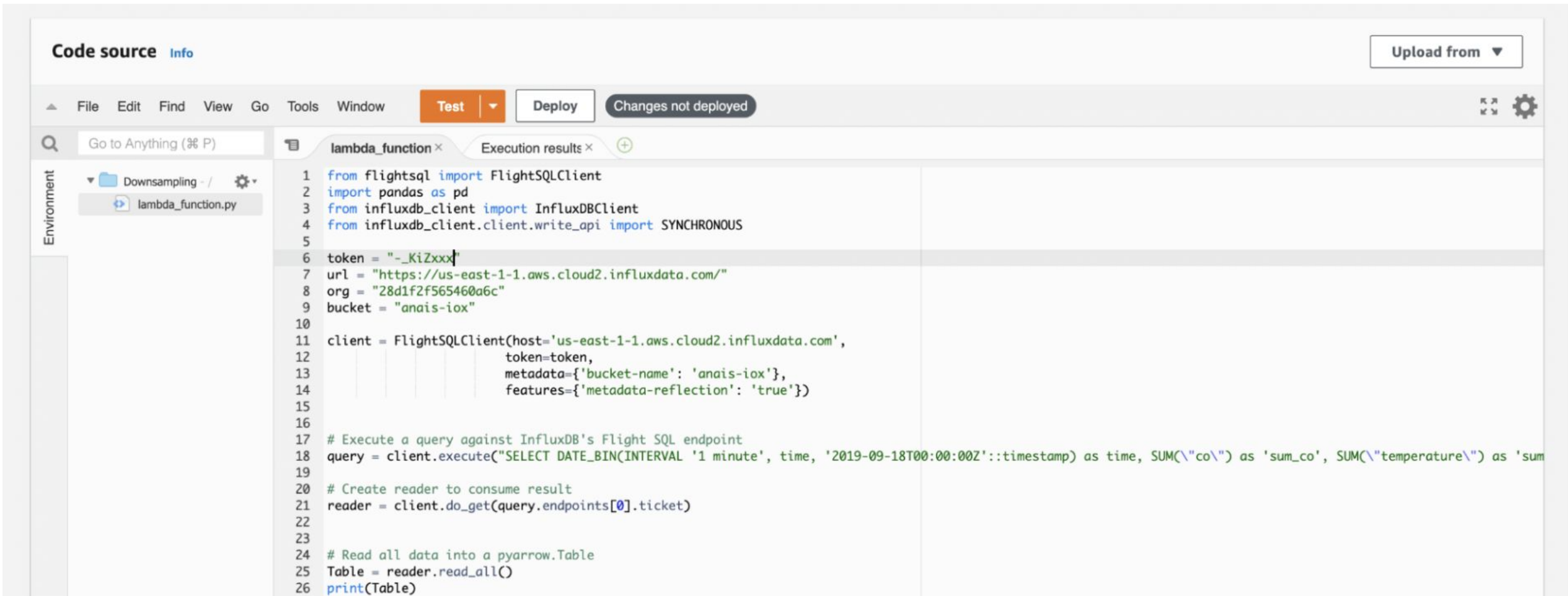
```
1  import "influxdata/influxdb/tasks"
2
3  option task = {name: "PB_downsample", every: 1h, offset: 10s}
4
5  from(bucket: "plantbuddy")
6    |> range(start: tasks.lastSuccess(orTime: -task.every))
7    |> filter(fn: (r) => r["_measurement"] == "sensor_data")
8    |> aggregateWindow(every: 10m, fn: mean, createEmpty: false)
9    |> yield(name: "mean")
10   |> to(bucket: "downsampled")
```

Next steps...

Adding Grafana Visualization



Running downsample Tasks in AWS Lambda and Python



The screenshot displays the AWS Lambda console's 'Code source' tab for a function named 'lambda_function'. The interface includes a top navigation bar with 'Code source' and 'Info' tabs, and a right-hand 'Upload from' button. Below this is a menu bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', and 'Window' menus, alongside 'Test' and 'Deploy' buttons, and a 'Changes not deployed' status indicator. The left sidebar shows the 'Environment' section with a file explorer for 'Downsampling - /' containing 'lambda_function.py'. The main editor area shows the Python code for the lambda function, which imports FlightSQLClient, pandas, and InfluxDBClient, sets up credentials, connects to InfluxDB, and executes a query to retrieve downsampled data.

```
1 from flightsql import FlightSQLClient
2 import pandas as pd
3 from influxdb_client import InfluxDBClient
4 from influxdb_client.client.write_api import SYNCHRONOUS
5
6 token = "-_KiZxxx"
7 url = "https://us-east-1-1.aws.cloud2.influxdata.com/"
8 org = "28d1f2f565460a6c"
9 bucket = "anais-io"
10
11 client = FlightSQLClient(host='us-east-1-1.aws.cloud2.influxdata.com',
12                          token=token,
13                          metadata={'bucket-name': 'anais-io'},
14                          features={'metadata-reflection': 'true'})
15
16
17 # Execute a query against InfluxDB's Flight SQL endpoint
18 query = client.execute("SELECT DATE_BIN(INTERVAL '1 minute', time, '2019-09-18T00:00:00Z'::timestamp) as time, SUM(\"co\") as 'sum_co', SUM(\"temperature\") as 'sum")
19
20 # Create reader to consume result
21 reader = client.do_get(query.endpoints[0].ticket)
22
23
24 # Read all data into a pyarrow.Table
25 Table = reader.read_all()
26 print(Table)
```

Further Resources

Try it yourself

The screenshot displays the GitHub repository for `InfluxCommunity/plant_buddy`. The repository is public and has 5 branches and 0 tags. The commit history shows a recent commit by Jayclifford345 adding Influx templates. The README content describes the project as a demonstration of using InfluxDB as a storage backend for a Flask server, with a BPMN diagram illustrating the data flow.

Repository Structure:

File/Folder	Commit Message	Time Ago
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 requirments README not finished	5 months ago

README.MD Content:

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 retriive your IoT data and then use Dash plotly to visualise. The below BPMN outlines the overall architecture and data flow:

```
graph LR
    InfluxDB[InfluxDB] --> CollectData[Collect data from sensors]
    CollectData --> VisualizeData[Visualize data]
    VisualizeData --> SendData[Send data via serial (USB)]
```



https://github.com/InfluxCommunity/plant_buddy

https://github.com/InfluxCommunity/plant_buddy_iox

InfluxDB Community Slack workspace

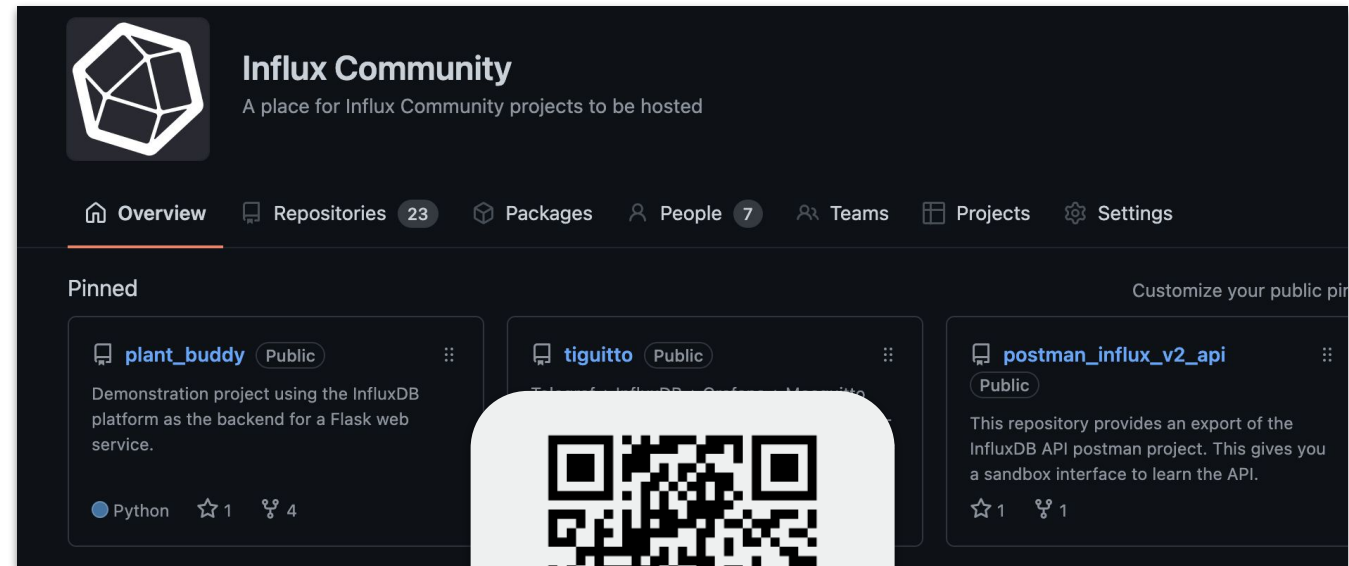
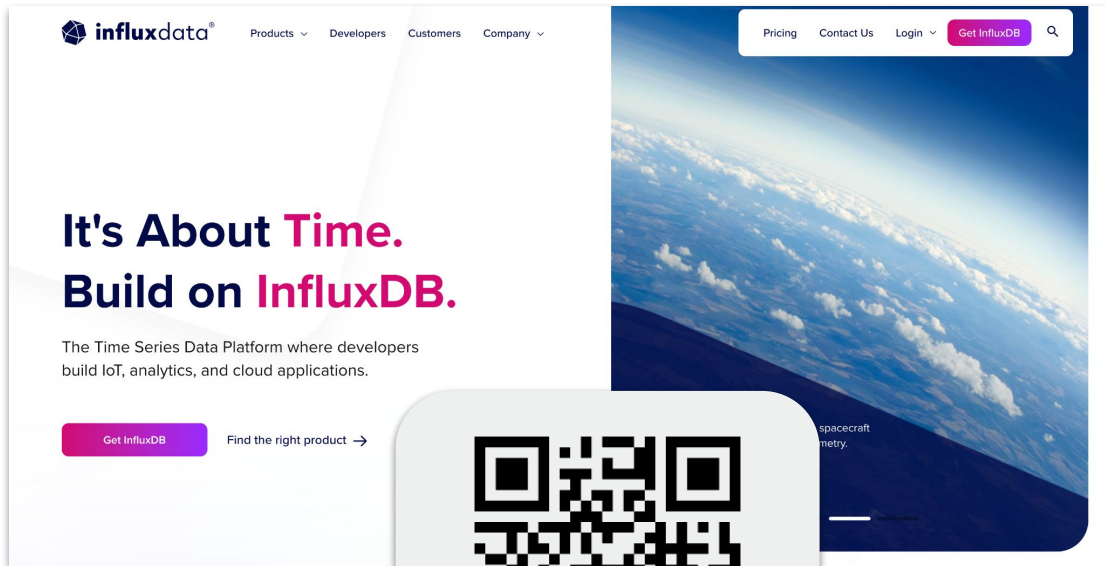


Please join us in the InfluxDB
Community Slack at
www.influxdata.com/slack.

To participate in conversations,
join the `#influxdb_iox` channel.

Try it yourself

Get Started



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?