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

InfluxDB Cloud

Application Server (Flask)

plantbuddy

Downsampled

Edge Replication Queue
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

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>POWERFUL</strong> API &amp; Toolset for real-time apps</td>
<td><strong>HIGH PERFORMANCE</strong> Time Series Engine for real-time data workloads</td>
<td><strong>MASSIVE</strong> Community &amp; Ecosystem of cloud &amp; open source developers</td>
</tr>
</tbody>
</table>
Telegraf for Ingestion

The open-source agent for collecting metrics

Driven by the community (600+ contributors)

Simple to configure, extremely flexible
# Client Libraries

**Back-end, front-end, and mobile applications**

<table>
<thead>
<tr>
<th>Arduino</th>
<th>C#</th>
<th>Dart</th>
<th>Go</th>
<th>Java</th>
<th>JavaScript/Node.js</th>
<th>Kotlin</th>
<th>PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="arduino-logo.png" alt="Arduino Logo" /></td>
<td><img src="csharp-logo.png" alt="C# Logo" /></td>
<td><img src="dart-logo.png" alt="Dart Logo" /></td>
<td><img src="go-logo.png" alt="Go Logo" /></td>
<td><img src="java-logo.png" alt="Java Logo" /></td>
<td><img src="node-logo.png" alt="JavaScript/Node.js Logo" /></td>
<td><img src="kotlin-logo.png" alt="Kotlin Logo" /></td>
<td><img src="php-logo.png" alt="PHP Logo" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Python</th>
<th>R</th>
<th>Ruby</th>
<th>Scala</th>
<th>Swift</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="python-logo.png" alt="Python Logo" /></td>
<td><img src="r-logo.png" alt="R Logo" /></td>
<td><img src="ruby-logo.png" alt="Ruby Logo" /></td>
<td><img src="scala-logo.png" alt="Scala Logo" /></td>
<td><img src="swift-logo.png" alt="Swift Logo" /></td>
</tr>
</tbody>
</table>
Flux Extension for VS code

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

Features:

- Flux language support

Categories
- Programming Languages
- Snippets

Resources
- Marketplace
- Repository
- License

Flash v0.6.6
InfluxData 5.404 ★★★☆☆ (3)

Flux language extension for VSCode

Disable  Uninstall

This extension is enabled globally.
Plotly for Graphing

- **Fundamentals**
  - The Figure Data Structure
  - Creating and Updating Figures
  - Displaying Figures
  - Plotly Express
  - Analytical Apps with Dash

- **Basic Charts**
  - Scatter Plots
  - Line Charts
  - Bar Charts
  - Pie Charts
  - Bubble Charts

- **Statistical Charts**
  - Error Bars
  - Box Plots
  - Histograms
  - Distplots
  - 2D Histograms

More Fundamentals »
More Basic Charts »
More Statistical Charts »
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

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

Data Systems
- APIs
- IoT Platforms
- CRMs

Telegraf
- HTTP
- Syslog
- Kubernetes
- Apache Kafka
- AWS Kinesis
- Azure Event Hubs
- GCP Pub/Sub

InfluxDB Platform

Client Libraries
- Python
- Arduino
- Node.js
- JavaScript
- Java
- .NET/C#
- PHP
- Ruby
- Go

Native Ecosystems
- JMeter
- NiFi
- Vector
- Fluentd

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

InfluxDB + Telegraf + Flux

Application Workflows

Infrastructure Insights

IoT Actions
Data Ingestion Setup
Connecting to the microcontroller

```
username@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

SOURCES  BUCKETS  TELEGRAF  API TOKENS

Search: Filter buckets...
Sort by Name (A → Z)

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.

downsampled
- Retention: 30 days
- Schema Type: Implicit
- ID: de5d92a7e4486d48
  
  + 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: 0fdca9988408a9b

_tasks
- System Bucket
- Retention: 3 days
- Schema Type: Implicit
- ID: 0fdca9988408a9b
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

```yaml
# 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]
```
Table example of the resulting data points

<table>
<thead>
<tr>
<th>_measurement</th>
<th>_field</th>
<th>_value</th>
<th>_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor_data</td>
<td>light</td>
<td>1337.47</td>
<td>2022-08-07T06:00:00.000Z</td>
</tr>
<tr>
<td>sensor_data</td>
<td>light</td>
<td>1281.8666666666666668</td>
<td>2022-08-07T06:10:00.000Z</td>
</tr>
<tr>
<td>sensor_data</td>
<td>soil_moisture</td>
<td>1372.0055555555555555</td>
<td>2022-08-08T17:40:00.000Z</td>
</tr>
<tr>
<td>sensor_data</td>
<td>soil_moisture</td>
<td>1322.74000000000002</td>
<td>2022-08-08T17:50:00.000Z</td>
</tr>
</tbody>
</table>
Flux
Introducing Flux

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

```plaintext
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(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))
})
```
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)
Flight SQL

APACHE ARROW

Parquet

DATA FUSION

influxdb™

plotly

ARUNDO

Apache Spark™

tableau

Grafana
Edge Data Replication
InfluxData Edge Data Replication

**Edge (InfluxDB OSS) Databases**

- Disk
- Bucket
- Queue
- Flux

**Cloud (InfluxDB Cloud) Database(s)**

- **Enables:**
  - Raw data replication
  - Downsampling
  - Eventual consistency

- **Build:**
  - Distributed databases
  - Hybrid apps
  - ML pipelines

---

### APIs and CLIs

<table>
<thead>
<tr>
<th>API</th>
<th>CLI</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/v2/remotes</td>
<td>influx remote [create, delete, list, update]</td>
</tr>
<tr>
<td>/api/v2/replications</td>
<td>influx replication [create, delete, list, update]</td>
</tr>
</tbody>
</table>
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-url https://us-east-1-1.aws.cloud2.influxdata.com
Data Request & Visualization
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

```python
# Wrapper function used to query InfluxDB
# Calls SQL script with parameters. 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

```python
@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 Dashboard

Welcome: Jay

Regenerate graphs

Click here to query InfluxDB for new data
Soil and Room Temperature

Plant Buddy Dashboard

Welcome: Jay

Click here to query InfluxDB for new data
Room Humidity and Soil Moisture

Plant Buddy Dashboard

Welcome: Jay

Regenerate graphs

Click here to query InfluxDB for new data
Plant Buddy Dashboard

Welcome: Jack

Regenerate graphs

Click here to query InfluxDB for new data
Alerting + Downsampling
Threshold Alert + Slack Notification

```javascript
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-323932326306512")
    |> 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

```flux
import "influxdata/influxdb/tasks"

option task = {name: "PB_downsample", every: 1h, offset: 10s}

from(bucket: "plantbuddy")
  |> range(start: tasks.lastSuccess(orTime: -task.every))
  |> filter(fn: (r) => r["_measurement"] == "sensor_data")
  |> aggregateWindow(every: 10m, fn: mean, createEmpty: false)
  |> yield(name: "mean")
  |> to(bucket: "downsampled")
```
Next steps...
Adding Grafana Visualization
Running downsample Tasks in AWS Lambda and Python

```python
from flightsql import FlightSQLClient
import pandas as pd
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

token = "-K1zxx4"
url = "https://us-east-1-1.aws.cloud2.influxdata.com/"
org = "28d1f2f956460b6c"
bucket = "anoeis-iow"

client = FlightSQLClient(host='us-east-1-1.aws.cloud2.influxdata.com',
token=token,
metadata={"bucket-name": 'anoeis-iow'},
features={"metadata-reflection": 'true'}))

# Execute a query against InfluxDB's Flight SQL endpoint
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_temp' FROM "anoeis-iow"."co" GROUP BY time(1m)"
reader = client.do_get(query.endpoints[0].ticket)

# Read all data into a pyarrow.Table
Table = reader.read_all()
print(Table)
```
Further Resources
Try it yourself

https://github.com/InfluxCommunity/plan_t_buddy

https://github.com/InfluxCommunity/plan_t_buddy_iox
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?