# Internet of Things (ECE:5550)
Spring 2024
Lab 04

**Start Date:** Monday, April 8.
**Due Date:** Sunday, April 14 by 11:59pm (ICON submission, TA check-off).
**Lab Teams:** Same as previous labs.

**Important Notes**

- These instructions assume you are working on your Raspberry Pi. For the few of you with Pi connectivity issues, this lab should be able to be completed on your own machine. In this document, **any reference to the Pi would instead be done on your machine. Where applicable, I have also attempted to include instructions for Windows and Mac machines.**
- These instructions contain several commands that need to be entered in your Raspberry Pi shell. If you use copy/paste to copy commands from this PDF document to your terminal window, **certain characters may not copy properly (e.g. double quotes, dashes, plus signs, etc.) and may need to be corrected on the terminal within your editor (e.g., Vim).**

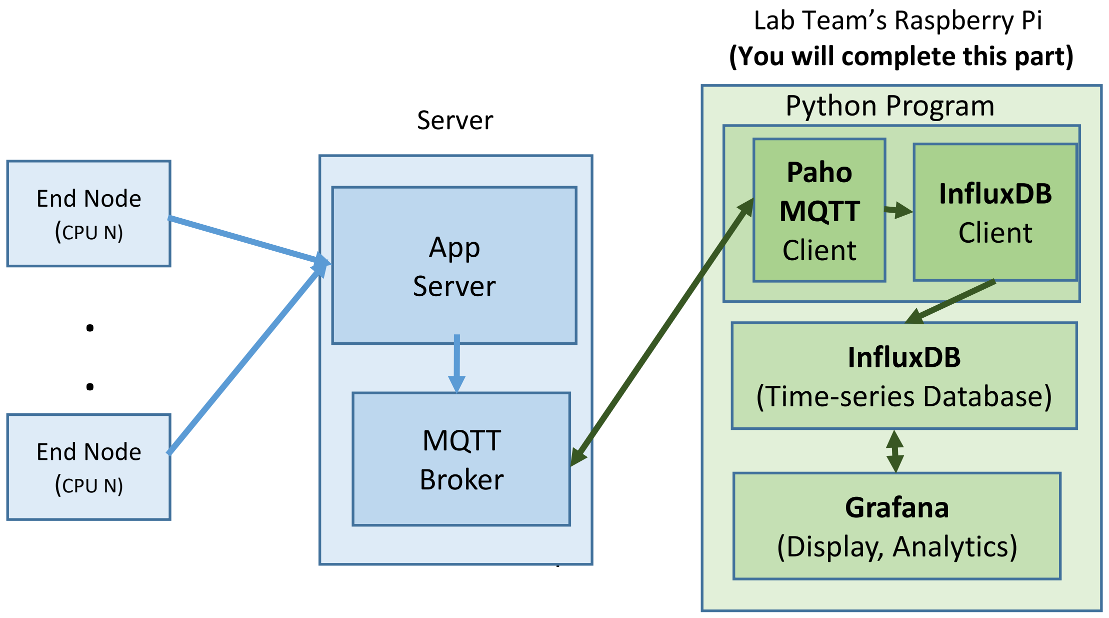
## Objectives

This lab has three main objectives:

1. Highlight *Message Queuing Telemetry Transport* (MQTT) for use in IoT systems.
2. Gain more familiarity with using JSON between inner parts of IoT application.
3. Illustrate use of time-series database for storing and querying periodically sampled data.

## Overview

Resource management for IoT devices can have an impact in terms of available computation, battery utilization, and bandwidth. This lab will utilize a Raspberry Pi to implement a monitoring system that collects, stores, and displays time-series data from a set of three remote CPUs.



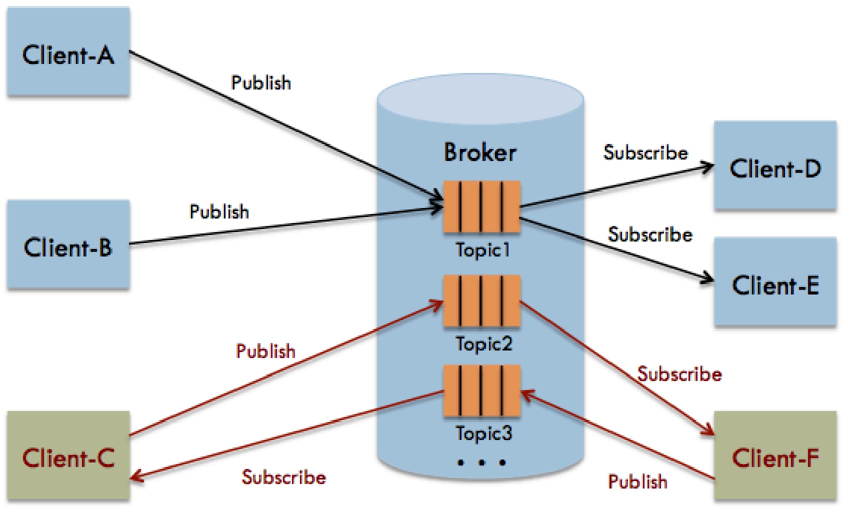
The blue portion of the above figure has already been implemented. **During this lab you will implement the green portion on your lab Raspberry Pi / PC / etc.**

The blue portion of the system consists of three simulated remote devices that are running their sensors, communications, and performing computations. To allow the core of the IoT system to monitor each device's health (and potentially reallocate tasks), the remote devices periodically check in and report their current CPU utilization (in terms of a percentage). To perform this check in, each device *publishes* their CPU usage percentage to a MQTT broker.

To implement the green portions of the figure, you will install InfluxDB (a time-series database) and Grafana (a visualization and analysis tool for time-series data) on your Raspberry Pi. You will then complete the implementation of a Python program that will bridge data coming from MQTT and store it within your database. The Python program will subscribe to *topics* on the MQTT broker to receive CPU usage data published and will forward that data into your InfluxDB database. You will then use Grafana to connect to your database to provide visualization and analysis.

## Part 1: MQTT and JSON

MQTT (Message Queueing Telemetry Transport) is a lightweight publisher-subscriber (pub-sub) protocol that runs on top of TCP/IP. The protocol was originally created for machine to machine (M2M) applications but has found widespread use in IoT systems. MQTT is much more than just a protocol as there are numerous implementations of MQTT functionality that can be directly integrated into IoT applications. The basic architecture of an MQTT-based application is shown below.



The architecture consists of a **broker** and one or more clients. A client can connect to the broker as a **publisher**, **subscriber**, or both. Data is organized into **topics**. A client that has connected as a publisher to a given topic can send updated topic data to the broker at any time. Clients that have connected as subscribers to a given topic will receive updated topic data from the broker whenever a publisher updates the topic. Topics have a hierarchical organization, akin to a directory structure, and are identified by slash-delimited strings, like file pathnames.

For instance, a topic might look something like:

```
app0/device1/sensordata/temperature
```

Clients can use *wildcards* to subscribe to multiple, related topics. The most common wildcard is '+.' For instance, the topic string:

```
app0/+/sensordata/temperature
```

could be used to subscribe to the temperature field from the sensordata of all devices that are associated with application `app0`. Another important wildcard is '#' which is used to wildcard the remainder of a topic string. For example:

```
app0/device1/#
```

refers to all topics that begin with `app0/device1`. In general, it's a good idea to keep topic names short as they are included in messages transferred between clients and the broker and thus contribute to communication overhead.

## JSON

Although the MQTT standard permits a variety of message data formats, the most used one is JSON (JavaScript Object Notation). You've already been using it in the previous labs when sending/receiving data from Firebase. JSON is a standard for "text-based" data interchange, that is based upon the structure of JavaScript objects (i.e., nested key-value pairs). JSON has largely supplanted XML as the preferred format for Internet data exchange for two important reasons: (1) it is typically more concise and results in less communication overhead; and (2) it is easily marshalled in/out of compatible key:value-based data structures in most modern programming languages (e.g. JavaScript objects, Python dictionaries, C++ and Java maps, Ruby hashes) thus reducing the need for specialized parsers.

## Mosquitto

There are several open-source implementations of MQTT browsers and clients. One of the of the most popular is Eclipse Mosquitto. We will use the Mosquitto command-line client to experiment with MQTT and examine the JSON format of the data published by the MQTT broker on our server. We will start by installing the `mosquitto-clients` package on the Pi by:

```
sudo apt update

sudo apt install mosquitto-clients
```

Installers and instructions for Mac and Windows can be found here: https://mosquitto.org/download/. On Windows, you may want to add the installation directory (e.g., C:\Program Files\mosquitto) to your system's PATH (tutorial). This will enable the *mosquitto_sub* executable to be found.

Now you test your connection to a MQTT broker. There is one publicly available for use in testing MQTT applications. Try subscribing to it to see what the results are via:

```
mosquitto_sub -h test.mosquitto.org -t "#" -v
```

This will (1) open the mosquitto subscriber command line client (man); (2) connect to the MQTT broker at the host *test.mosquitto.org*; and (3) subscribe to the wildcard topic "#" (all).

You should get *many* responses from this server as you've asked to receive anything published to the server. Use `ctrl+c` to kill the `mosquitto_sub` process. Based on some of the responses you get on the subscription to the wildcard #, try changing the -t parameter to something more restrictive (e.g., try listening to data coming from a specific source or device). From the responses on # there should be many topics to choose from. **Take screenshots of the responses you get when subscribed to "#" and another when subscribed to a topic of your choosing.**

Next, subscribe to the MQTT broker for our lab. You can do this by:

```
mosquitto_sub -h broker.hivemq.com -t "uiowa/iot/lab4/#" -v
```

This will connect to the MQTT broker on port 1883. Subscribing to the # on the uiowa/iot/lab4 topic will subscribe to all devices outputting data (our three simulated CPUs: cpu1, cpu2, cpu3). After you've received several responses from the server, please take another screenshot of your values.

**Part 1 Submission**

(1) Screenshot of responses from test.mosquitto.org on topic #.

(2) Screenshot of responses from test.mosquitto.org on a more specific topic of your choosing (please have the topic you chose visible).

(3) Screenshot of responses from this lab's MQTT broker.

## Part 2: Integrating MQTT with a Time-series Database

In this portion of the lab, we will extract sensor data obtained from the MQTT broker and store it into a time-series database called InfluxDB. As the name implies, a time-series database is optimized to store time-stamped streams of data samples. InfluxDB accepts data points in JSON format. To bridge the MQTT stream to InfluxDB we will need to create a Python program that acts both as an MQTT subscriber client and as a client for the InfluxDB API.

Let's begin by installing Influx database on the Pi [ref]. You can do this by entering the following:

```
$ curl -O https://dl.influxdata.com/influxdb/releases/influxdb2_2.7.5-1_arm64.deb
$ sudo dpkg -i influxdb2_2.7.5-1_arm64.deb
```

Now start the InfluxDB server by entering the command:

```
$ sudo service influxdb start
```

Now we need to get the command line interface (CLI) to test the server out:

```
$ wget https://dl.influxdata.com/influxdb/releases/influxdb2-client-2.7.3-linux-
arm64.tar.gz
$ tar xvzf ./influxdb2-client-2.7.3-linux-arm64.tar.gz
$ sudo cp ./influx /usr/local/bin/
```

*(For Mac and Windows, you can run the newest version of InfluxDB via Docker or you can install an older version of InfluxDB (1.X) natively. Find these options at https://portal.influxdata.com/downloads/.)*

You can learn a little about InfluxDB by playing with it through its command line interface. To open the InfluxDB CLI, enter the following:

```
influx
```

If set up properly, you will see all the influx run options printed. Now run:

```
influx setup
```

This will let you set up your Influx installation. Please take note of your username and password! For the organization name, you can select, "iot" / "iowa" / etc. For the initial bucket name, you can select "lab4" / "iot" / etc. Regardless of your selections, just ensure you remember them all (write them down).

Next, we'll need to create a token so that your database can be securely used by an Influx client.

```
influx auth create --all-access
```

You will see, among other info, a token is created. Record it as you will need it soon. Should be like:

```
SsuF9NBFlx-ieosHkdPcSFc2bbgG7_VeaDhrsOQE77qEazUEKgH1sCf9K_m_e9GhCJmjE9OZPv3Ie8eb4_dZ0Q==
```

Finally, here is an example of creating a new database "bucket" called lab4test2 and adding time-series data to it. A timestamp will automatically be added to the data as it is entered into the database.

```
pi@raspberrypi:~ $ influx bucket create lab4test2
...creates bucket / database called lab4test2...
pi@raspberrypi:~ $ influx v1 shell
InfluxQL Shell dev
Connected to InfluxDB OSS v2.7.5
> use lab4test2
> insert sensorstream1 temperature=53.2
> insert sensorstream1 temperature=65.3
> insert sensorstream2 temperature=47.9
> insert sensorstream2 temperature=39.8
> insert sensorstream1 temperature=61.7
> insert sensorstream2 temperature=56.9
> insert sensorstream1 temperature=71.3
> select * from sensorstream1
name: sensorstream1
time                           temperature
----                           -----------
2019-04-05T14:32:50.860180946Z 53.2
2019-04-05T14:33:07.971406885Z 65.3
2019-04-05T14:34:14.763457112Z 61.7
2019-04-05T14:34:56.973188402Z 71.3
> select * from sensorstream2
name: sensorstream2
time                           temperature
----                           -----------
2019-04-05T14:33:26.453956054Z 47.9
2019-04-05T14:33:49.416884042Z 39.8
2019-04-05T14:34:35.440866253Z 56.9
> quit
```

InfluxDB organizes data into time-series streams called **measurements**. In the above example, sensorstream1 and sensorstream2 are measurements. A measurement stores a series of timestamped **points**. In the above example, each of the insert operations adds a new timestamped point to one of the sensorstream measurements. Each point can contain one or more **fields**. Fields are key:value pairs that represent data. In the above example, temperature=53.2 is a field (temperature: 53.2). Influx points can also have additional descriptive fields, called tags, but we will not use them in this lab. Another important aspect of a time-series database that we will not explore in this lab is the retention policy, which defines how long data will be retained in the database. If interested, you can find information regarding tags and retention policies on the InfluxBD web site.

For the database that will store our MQTT sensor data, we will **utilize a separate measurement for each CPU (cpu1, cpu2, cpu3)**. Each point stored in a measurement will have one field called ***usage***.

To construct our Python program to integrate the MQTT feed with InfluxDB, let's begin by exploring how to configure a program to act as an MQTT subscriber client and how to convert JSON data published by the MQTT broker into a Python dictionary. Copy the example Python program from ICON into a file on your Pi. Make sure it has the correct file extension: '.py'. This program uses a Python client library for MQTT called **paho-mqtt**. This popular library is maintained by Eclipse, and there are also versions for many other programming languages (see here). If you need to install Python on your own machine you can find its installers here: https://www.python.org/downloads/. After you have Python, open a new terminal/command window.

To install the paho-mqtt module on your Pi, enter the command:

```
$ cd ~
$ mkdir lab4
$ cd lab4
$ python3 -m venv ~/lab4/venv
$ source ~/lab4/venv/bin/activate
$ pip3 install paho-mqtt
```

This creates for you a new directory for your lab 4 in ~/lab4, creates and activates a new virtual environment for python, and then installs the paho-mqtt package. Note: you will likely need to reactivate this virtual environment every time you come back here to work. You can do that by typing `cd ~/lab4` and then the `source …/activate` command each time.

Now, open the provided Python program with your favorite editor and review it. Observe how the program connects to the MQTT broker at *broker.hivemq.com* on port 1883 as a subscriber to topic: "uiowa/iot/lab4/#" and then listens for incoming JSON data from the broker. When a JSON message arrives, the program converts it to a Python dictionary (dict) object and then pretty-prints the dictionary. Note that the dictionary object contains the same information as we examined earlier with the mosquitto command-line client. Run the program with:

```
python3 lab4.py
```

Take a screenshot of the output printed by the initial Python program.

Now we are ready to integrate our sensor data stream with InfluxDB. To install the Python InfluxDB library, enter the command (again, you may not need the sudo):

```
pip3 install influxdb-client
```

Back in your Python code, you can now uncomment the include statements at the top of the file.

You can also then provide the proper values for the bucket, org, and token strings and uncomment out the two lines that initialize the InfluxDBClient.

Once done, this will connect to a specific database on the local InfluxDB service running on your machine, using the proper token you've provided.

**IMPORTANT: You need to 'setup' and create a bucket/database via the Influx command line interface before doing this in your Python program (see info on Page 5).** In order to connect, you also need to be sure you have the Influx server started/running (see end of Page 4).

Now that your Python program has access to data published by the MQTT broker and the database, it's time to perform the bridging. You can write a **point** to your database with the statement:

```
influx_write_api.write(bucket=bucket, record=[pointData])
```

where point_data is a *dictionary* containing the necessary information for the point. Hint: you will most likely construct a new dictionary and use the *write* function within each call of *on_message*. Note that the square brackets around point_data above are necessary as *write* expects a list of dictionary objects. The structure of the dictionary must be as follows:

```
{
      "measurement": measurement name (e.g., cpu1, cpu2, cpu3),
      "fields":
      {
            Your fields go here, as key:value pairs
            (e.g., usage:value from MQTT)
      }
}
```

Recall that we want the *measurement* name to be the device name (cpu1, cpu2, or cpu3), and the fields to hold data values (as key:value pairs) for the field named *usage*. You will need to construct the *point_data* dictionary using information from the *msg_payload* dictionary. This StackOverflow answer may help you dynamically construct an object to use in *write*.

While running your Python program, you can check that points are being written into your database by using the Influx command line interface (CLI) from another command / terminal window:

```
$ influx v1 shell
>use lab4db (or whatever your database name is)
>select * from cpu1 (or another measurement)
```

Take a screenshot of the Influx CLI showing the display of a queried measurement.

**Part 2 Submission**

(1) A screenshot showing the initial output printed by the lab4.py program.

(2) A screenshot of the Influx CLI showing results of a query on one of your measurements.

## Part 3: Integrating InfluxDB with Grafana

In this final portion of the lab, you will connect the time-series database to a visualization and analysis tool called Grafana. Make sure that your Python program from Part 2 is still running during this step so that your InfluxDB database is continuously being populated with data.

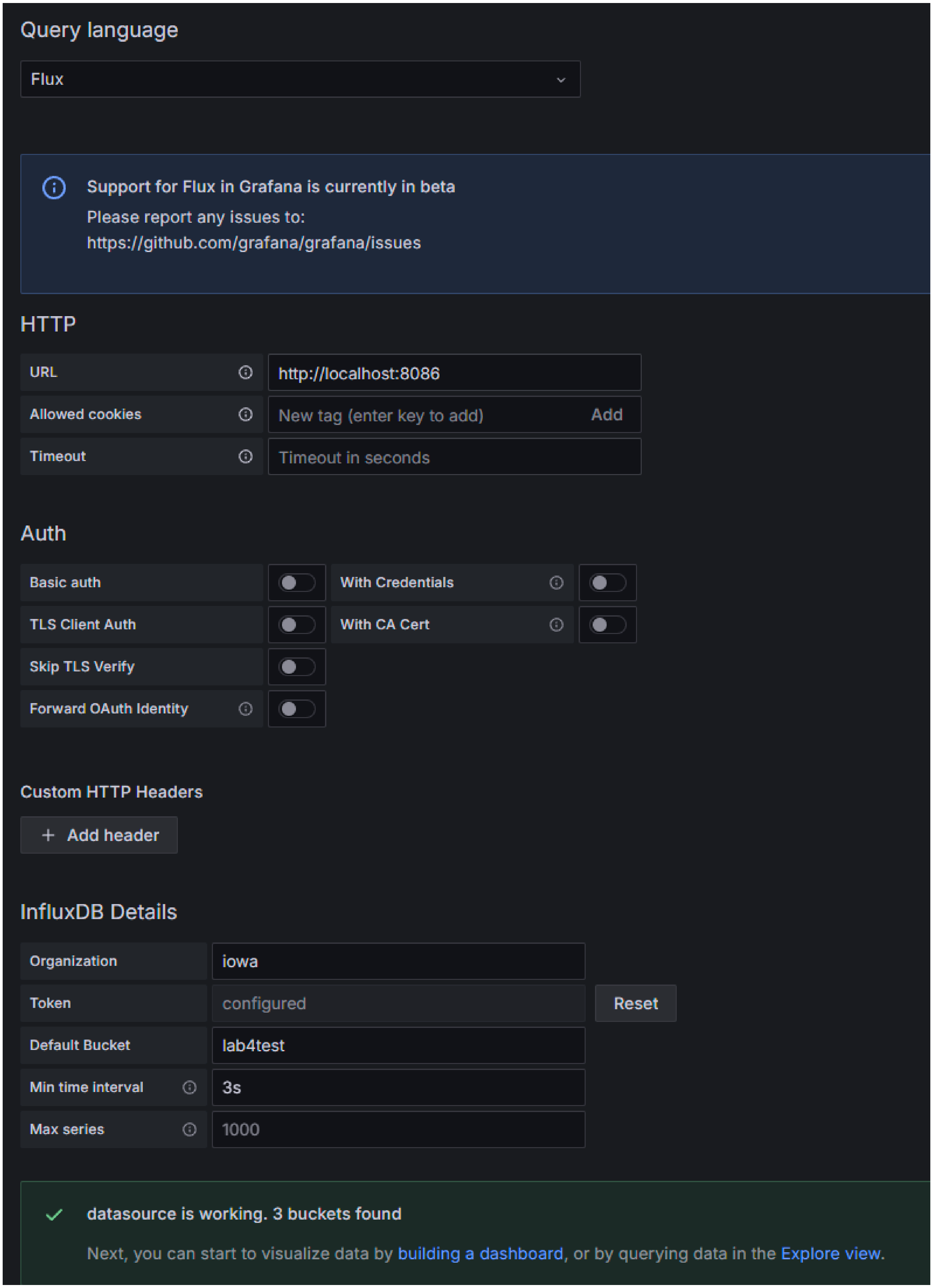
Begin by (1) installing and then (2) starting Grafana on your Pi following these instructions: https://grafana.com/tutorials/install-grafana-on-raspberry-pi/ at the **Install Grafana** section.

For Mac and Windows, you can find the downloads at https://grafana.com/grafana/download.

- If on Windows:
  - Windows install guide: https://grafana.com/docs/grafana/latest/installation/windows/
  - After installing, Grafana should start running in the background
- If on Mac:
  - Mac install guide: https://grafana.com/docs/grafana/latest/installation/mac/
  - After, start the Grafana service with ‘brew services start grafana’

After installing and starting Grafana on your Pi, point your PC’s browser at **http**://<address of your Pi; whatever you use for SSH>:3000 (e.g., http://raspberrypi.local:3000). Notice how it’s http, not https. If you are working on your own machine, point your browser to localhost:3000. This should take you to the Grafana login page where you can login with the username: *admin* and password: *admin.* You may then be asked to provide a new password.

Once you are logged in to the Grafana dashboard, click on “Add your first data source,” select InfluxDB, and then select “Flux” as your query language. Substitute your own org, token, and bucket name under InfluxDB details. For URL: http://localhost:8086 is default but this should be your Pi’s IP address:8086 or whatever else you used above when connecting to http://...:3000 above. For min time interval (e.g., 3s). Then click “Save and Test”. You should see a green box indicating that the data source is working. See the screenshot below for more info.



Query language

Flux

Support for Flux in Grafana is currently in beta
Please report any issues to:
https://github.com/grafana/grafana/issues

HTTP

| | |
|---|---|
| URL | http://localhost:8086 |
| Allowed cookies | New tag (enter key to add) Add |
| Timeout | Timeout in seconds |

Auth

| | | | |
|---|---|---|---|
| Basic auth | off | With Credentials | off |
| TLS Client Auth | off | With CA Cert | off |
| Skip TLS Verify | off | | |
| Forward OAuth Identity | off | | |

Custom HTTP Headers

+ Add header

InfluxDB Details

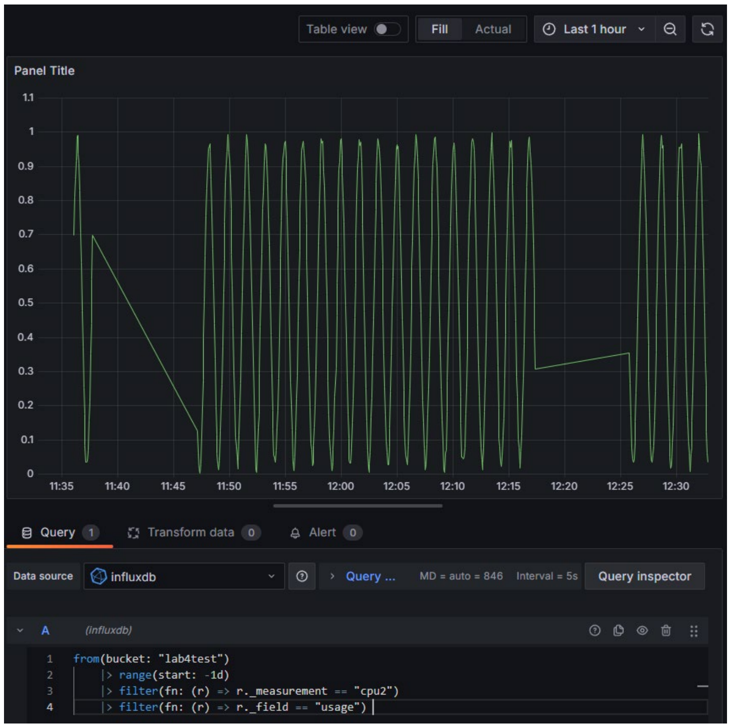
| | |
|---|---|
| Organization | iowa |
| Token | configured (Reset) |
| Default Bucket | lab4test |
| Min time interval | 3s |
| Max series | 1000 |

✓ datasource is working. 3 buckets found

Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

Now do the following steps:

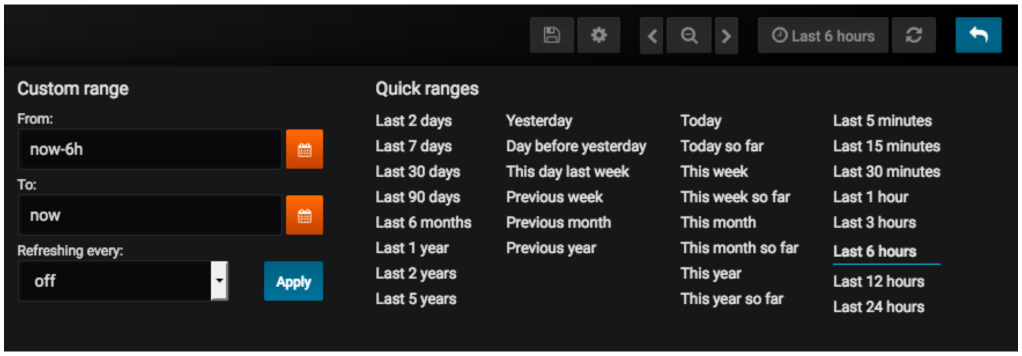
- Click on ‘+’ in the menu along the left side of the page
- Select “Create Dashboard” and click “Add a new panel” or “Add visualization”
- This will bring up a blank graph panel.
- This will cause a query editor to open below the panel.
- Use the query editor to adjust the query until it looks something like the picture below, <u>substituting your data source and measurement from the available options</u>.



Data source: influxdb
MD = auto = 846  Interval = 5s

```
from(bucket: "lab4test")
    |> range(start: -1d)
    |> filter(fn: (r) => r._measurement == "cpu2")
    |> filter(fn: (r) => r._field == "usage")
```

You can adjust the time range of data displayed on the graph via the tab in the upper right portion of the panel that will initially be labeled “Last 6 hours.” If you click on this tab, a menu will be displayed with a variety of range options, as shown below:



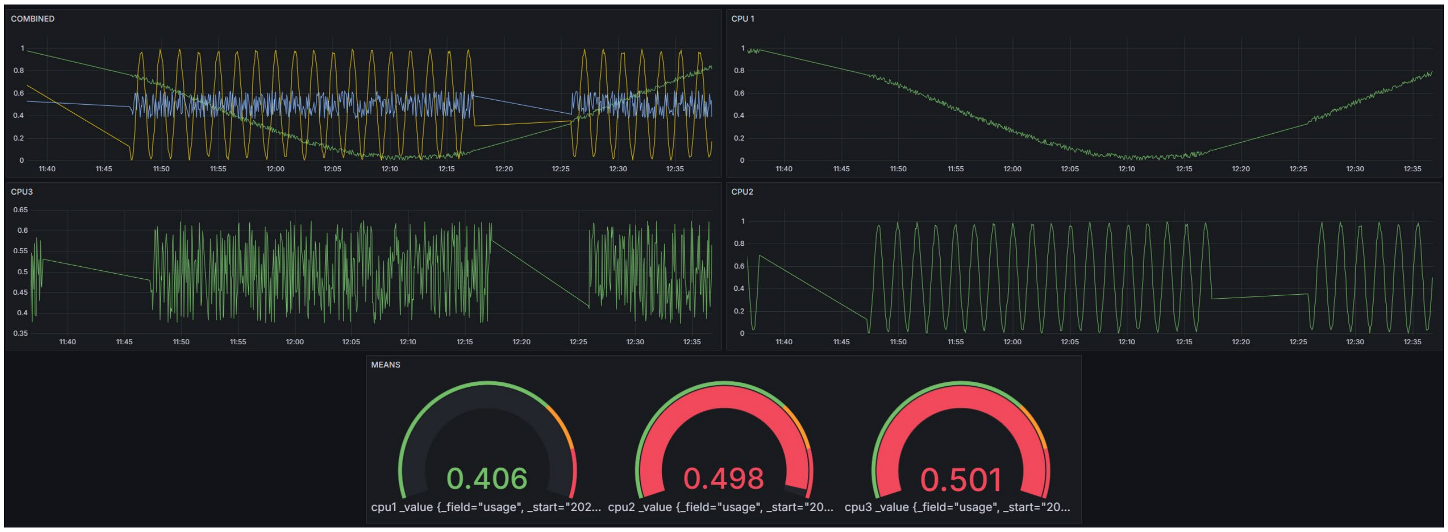
From: now-6h
To: now
Refreshing every: off

Create additional queries and visualizations to display more sensor data and stats in the panel and add additional panels to your dashboard (bar-graph icon with a ‘+’ sign in upper-right menu). Your goal should be displaying the sensor data collected by the system in a meaningful and informative manner. You can also try alternative query structures and visualizations from the wide variety that Grafana offers. You can consult Grafana documentation for more details regarding query structure.

**Part 3 Submission**

(1) Screenshot(s) of your Grafana dashboard, showing the display of collected data. You should at least achieve the first 4 visualizations on the following screenshot (cpu1, cpu2, cpu3, combined). You can also attempt to create an aggregate visualization. This is shown in the gauge example, and is produced by piping query results into a mean() function (see here).

(2) Screenshot(s) of your Grafana queries.


COMBINED
CPU 1
CPU3
CPU2
MEANS
0.406
0.498
0.501
cpu1 _value {_field="usage", _start="202...
cpu2 _value {_field="usage", _start="20...
cpu3 _value {_field="usage", _start="20...

## Submission & Check-off (one submission per lab team)

Submit to the ICON dropbox all the items under the **Part N Submission** headers for each of this lab's three parts. Also, **please submit the source code for your Python program.**

**In addition, please submit a short 1-2 page lab report describing an overview of your completed lab, code, teammate contribution(s), and how everything runs and is integrated together from an IoT perspective.** Please also detail the various steps you took towards completing the lab, any issues you ran into along the way, how you overcame them, and any other observations you had. Here you can include additional screenshots, code snippets, and insights you gained.

**The submission deadline is Sunday, April 14 by 11:59pm (ICON submission).**

As before, you will need to demonstrate your completed lab to a TA and receive their check-off. The TAs will have a sign-up sheet available in the Teams Checkoff channel's files to allow teams to reserve slots for this check-off. If you would like to meet with a TA outside of these times, please make these arrangements with him via email.

*If you have any issues, **particularly with installing any of the libraries or dependencies**, please contact our TAs or Prof. Bell ASAP so we can help get you on the right track.*