Part 41
-
InfluxDB

Version: 2020-02-20

Introduction

InfluxDB is a time series based database system. This means that each data point in the database will contain a timestamp. Being time series based makes InfluxDB one of the best databases for monitoring metrics and events. You can easily use InfluxDB to store information like the temperature in a room or the CPU usage of a system.

Collecting custom data, whether for a user-facing application or for an infrastructure requirement is probably going to require writing new code.

If you need to query and retrieve that data for your users, you'll probably want to take advantage of one of the InfluxDB libraries available in various languages to handle the interaction with InfluxDB within your application itself. There are a number of languages out there that already have InfluxDB libraries, many of them maintained by the community. We'll take a closer look at using the influxdb-python library, but if Python isn't your style, you can find a list of libraries on the InfluxDB API client libraries page

InfluxDB is the perfect database software to go alongside the popular visualization tool Grafana. Grafana has inbuilt support for displaying data from an InfluxDB database.

Installing InfluxDB to the Raspberry Pi

The first thing we need do before installing InfluxDB to the Raspberry Pi is making sure that all the currently installed packages are up to date.
We can upgrade all installed packages by running the following two commands.

```
sudo apt update
sudo apt upgrade
```

and reboot now.

```
sudo reboot
```

With everything now up to date, we can now proceed with installing InfluxDB to the Raspberry Pi. Our next step is to add the InfluxDB repository key to our Raspberry Pi.
Adding the key will allow the package manager on Raspbian to search the repository and verify the packages its installing.

We can add the InfluxDB key by running the following command.

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
```

This command will download the key using wget and pass it directly into the **`apt-key`** program by using a pipe "`|`".

Now that we have the InfluxDB repository key installed, we will need to go ahead and add its repository to the sources list. Enter the following command to do so. Make sure you pick the right command for the version of Raspbian that you are running. Most users running on new installations of Raspbian will likely be running Raspbian Buster.

```
echo "deb https://repos.influxdata.com/debian buster stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
```

With the repository added, we now need to go ahead and update the package list again.
We need to do this so that the apt package manager searches the repository that we just added for packages. The operating system does not automatically do this. Run the following command on your Raspberry Pi to update the package list.

```
sudo apt update
```

Now that we have set up the repository, we can now move on to installing the InfluxDB software.

To install InfluxDB to our Raspberry Pi, all we need to do is run the command below.

```
sudo apt -y install influxdb
```

With InfluxDB now installed, let's now get it to start at boot. We can do this by making use of the systemctl service manager to enable our InfluxDB service file. Run the following two commands to enable InfluxDB to start at boot.

```
sudo systemctl unmask influxdb
sudo systemctl enable influxdb
```

The first command we use unmasks the influxdb service file. Unmasking the service ensures that we can enable and start the service as a masked service is unable to be started.

Our second command enables the influxdb service. This command will tell the service manager to keep an eye on the "`influxdb.service`" file and setup the service based on its contents.
Now that everything has been set up, we can now proceed to start up InfluxDB.

To start up the InfluxDB server, we will need to run the following command. The service manager will then start up the service and begin monitoring it.

```
sudo systemctl start influxdb
```

Note: The way to install in according the the website of InfluxDB is

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-
1.7.10_linux_armhf.tar.gz
tar xvfz influxdb-1.7.10_linux_armhf.tar.gz
```

However,

- this make is version dependent. So you need to check at installation time for the right version
- You will have to do the installation yuorself = putting files to the right place, setting up the daemon, autostart etc.

Using InfluxDB

As we have now installed InfluxDB, we can now start talking with the database.
To do this, we will need to launch up Influx's command-line tool by running the command below.

```
influx
```

and wait until you see a "`>`" prompt
You don't have to worry about specifying an address to connect to as the tool will automatically detect the local installation of InfluxDB. By default, InfluxDB has no users setup. For now, we will quickly explore InfluxDB.

InfluxDB comes with no databases by default, so our first task will be to create one.
Creating a database is simple in InfluxDB and can be done by using "`CREATE DATABASE <DBNAME>`"

For our example, we will be creating a database called `roomtemperatures`,

```
CREATE DATABASE roomtemperatures
```

Before we can start modifying our new database, we must tell the CLI (Command Line Interface) to "`use`" it. Using a database is as simple as running the following command.

```
USE roomtemperatures
```

Our next step is to write some data to our newly created InfluxDB database.
To do this, we must first get a basic understanding of InfluxDB's datastore.
Data in InfluxDB are sorted by "`time series`". These "`time series`" can contain as many or as little data points as you need. Each of these data points represents a single sample of that metric. A data point consists of the `time`, a `measurement` name such as "temperature", and at least one `field`. You can also use `tags` which are indexed pieces of data that are a string only. Tags are essential for optimizing database lookups.
If you are familiar with the general layout of an SQL table, you can consider "`time`" to be the primary index, `measurement` as the table name, and the `tags` and `fields` as the column names. You do not need to specify the timestamp unless you want to specify a specific time and date for the data point.
Below we have included the basic format of an InfluxDB data point.

```
<measurement>[,<tag-key>=<tag-value>...] <field-key>=<field-value>[,<field2-
key>=<field2-value>...] [unix-nano-timestamp]
```

If you would like to learn more about the InfluxDB line syntax, then you can check out the InfluxDB official documentation.

Now that we have a basic understanding of data in InfluxDB, we can now proceed to add our very first data point to our database.
For this example database, we are going to be storing measurements of the "`temperature`" of various `locations` around a house. So for this, we will be inserting data points with a measurement name of "`temperature`" and a tag key of "`location`", and a field key of "`value`". For our first sample point, we will be saying the location is the "`living_room`", and the value is "`20`" .

```
INSERT temperature,location=living_room value=20
```

To make the data more interesting for showing off selecting data in InfluxDB, let's go ahead and add some more random data.
Enter the following few commands to enter some extra data into our database. These are just variations of the above "INSERT" command but with the value and location adjusted.

```
INSERT temperature,location=living_room value=10
INSERT temperature,location=bedroom value=34
INSERT temperature,location=bedroom value=23
```

Now that we have some sample data, we can now show you how to query this data using "SELECT". To start with, you can retrieve all data from a measurement by using a command like below. This command will grab all fields and keys from the specified measurement.

```
SELECT * FROM temperature
```

Using that command with our sample data you should get a result like we have below.

```
name: temperature
time                location    value
----                --------    -----
1574055049844513350 living_room 20
1574055196564029842 living_room 10
1574055196576516557 bedroom     34
1574055197188117724 bedroom     23
```

Let's say that you now only wanted to retrieve the temperature of the bedroom. You can do that by making use of the "WHERE" statement alongside a "SELECT" statement. We also specify the name of the tags/fields that we want to retrieve the values from.
When querying tag fields, you need to remember that all tags are considered to be strings. This means that we must wrap the value we are searching for in single quotes.

```
SELECT value FROM temperature WHERE location='bedroom'
```

With that command, you should receive the following data set, showing only the temperature value in the bedroom. Which in our example data's case, this should be 34 and 23.

```
name: temperature
time                value
----                -----
1574055196576516557 34
1574055197188117724 23
```

At this point, you should now have a basic understanding of InfluxDB and how its data works.

Adding Authentication to InfluxDB

The next step is to add extra authentication to our InfluxDB installation. Without authentication, anyone could interact with your database.
To get started, we need to first create a user to act as our admin. To create this user, we must first load up the InfluxDB CLI tool by running the following command.

```
Influx
```

Within this interface, we can create a user that will have full access to the database. This user will act as our admin account. To create this admin user, run the following command within InfluxDB's CLI tool. Make sure that you replace `<password>` with a secure password of your choice.

```
CREATE USER admin WITH PASSWORD '<password>' WITH ALL PRIVILEGES
```

This command will create a new user called "`admin`" with your chosen password and grant it all privileges. With that done, you can now exit out of InfluxDB by typing in "`exit`" and pressing `ENTER`.

Our next job is to modify the InfluxDB config file to enable authentication. We can begin editing the file by using the command below.

```
sudo nano /etc/influxdb/influxdb.conf
```

Find the `[HTTP]` section and add the following options underneath that section.

```
auth-enabled = true
pprof-enabled = true
pprof-auth-enabled = true
ping-auth-enabled = true
```

Once added, save the file by pressing CTRL + X, then Y, followed by ENTER.
Now, as we have made changes to InfluxDB's configuration, we will need to go ahead and restart the service by using the following command. Restarting the service will ensure that our configuration changes are read in.

```
sudo systemctl restart influxdb
```

As we have now turned InfluxDB's authentication on, we will need to enter our username and password before using the InfluxDB CLI tool.
You can use the "`auth`" command within the tool or pass the username and password in through the command line, as we have below.

```
influx -username admin -password <password>
```

Hopefully, at this point, you will now have successfully set up InfluxDB. You should now have a basic understanding of InfluxDB as well as have authentication mode enabled.
You can easily see how using this database model will make storing data from sensors and other sources very useful.

Installing and Using InfluxDB Python Library

Like many Python libraries, the easiest way to get up and running is to install the library `using pip`.
We're going to run `pip` using the `-m` argument to the Python command, in order to be certain which Python is the install target

```
python3 -m pip install influxdb
```

You should see some output indicating a successful installation.

We can now start writing a sample Python script, test-InfluxDB.py, that will handle the basics with InfluxDB.

Note: you could also use

```
sudo apt -y install python-influxdb
sudo apt -y install python3-influxdb
```

Importing the InfluxDB Client
The first thing to do is importing the InfluxDB Client from the python-influxdb library to make sure it was installed:

```
from influxdb import InfluxDBClient
```

Making a Connection
The next step will be to create a new instance of the InfluxDBClient, with information about the server that we want to access. We're running locally on the default port:

```
IDBclient = InfluxDBClient(host='localhost', port=8086, username='admin',
password='<password>')
```

Note: Replace <password> with the password you provided previously. If you need to access a InfluxDB host remotely, enter the host IP address or host name of the InfluxDB server. Same if InfluxDB service is running on a differten port

There are some additional parameters available to the InfluxDBClient constructor, including username and password, which database to connect to, whether or not to use SSL, timeout and UDP parameters.
If you wanted to connect to a remote host at mydomain.com on port 8086 with username myuser and password mypass and using SSL, you could use the following command instead, which enables SSL and SSL verification with two additional arguments, ssl=True and ssl_verify=True

```
IDBclient = InfluxDBClient(host='mydomain.com', port=8086, username='myuser',
password='mypass', ssl=True, verify_ssl=True)
```

Create Database
Now, let's create a new database called pyexample to store our data:

```
IDBclient.create_database('pyexample')
```

Note: before creating the database, we can check if the database is there by using the get_list_database() function of the client:

```
IDBclient.get_list_database()
[{'name': '_internal'}, {'name': 'pyexample'}]
```

Use Database
Finally, we'll set the client to use this database:

```
IDBclient.switch_database('pyexample')
```

Writing Data
Now that we have a database to write data to, and our client properly configured, it's time to insert some data. We're going to use our client's write() method using the 'line' protocol to do so.

Note: using JSON is also possible but internally JSON is converted to line, hence, it takes longer to write with JSON.
The data to be written is the same as above

```
temperature,location=living_room value=20
temperature,location=living_room value=10
temperature,location=bedroom value=34
temperature,location=bedroom value=23
```

So the Python code becomes

```
IDBclient.write(['temperature,location=living_room value=20'], {'db':'pyexample'},
204, 'line')
```

Note: 204 is the return code when write was succesful. See API documentation

```
IDBclient.write(['temperature,location=living_room value=10'], {'db':'pyexample'},
204, 'line')
IDBclient.write(['temperature,location=bedroom value=34'], {'db':'pyexample'},
204, 'line')
IDBclient.write(['temperature,location=bedroom value=23'], {'db':'pyexample'},
204, 'line')
```

Note: you can test a succesful write this way

```
if not IDBclient.write(['temperature,location=living_room value=10'],
{'db':'pyexample'}, 204, 'line'):
      print("write failed")
```

Querying Data
Now that we have some data in the database, let's try running some queries to get it back out. We'll use the same client object as we used to write data, except this time we'll execute a query on InfluxDB and get back the results using our client's query() function.

```
results = IDBclient.query('SELECT * FROM temperature')
```

The `query()` function returns a ResultSet object, which contains all the data of the result along with some convenience methods. Our query is requesting all the measurements in our pyexample database. You can use the `.raw` parameter to access the raw JSON response from InfluxDB:

```
print(results.raw)
```

In most cases you won't need to access the JSON directly, however. Instead, you can use the get_points() method of the ResultSet to get the measurements from the request, filtering by tag or field. If you wanted to iterate through all of Carol's brushing sessions; you could get all the points that are grouped under the tag "user" with the value "Carol", using this command:

```
points = results.get_points(tags={'location':'living_room'})
```

points in this case is a Python Generator, which is a function that works similarly to an Iterator; you can iterate over it using a for x in y loop, as follows:

```
for point in points:
      print("Time: %s, value: %i" % (point['time'], point['value']))
```

Depending on your application, you might iterate through these points to compute eg. the average.

Close Database
At the end, exit nicely by closing the connection

```
IDBclient.close()
```

The whole script

```
from influxdb import InfluxDBClient

# replace <password> with the password you assinged to InfluxDB
IDBclient = InfluxDBClient(host='localhost', port=8086, username='admin',
password='<password>')
IDBclient.create_database('pyexample')
IDBclient.switch_database('pyexample')

if not IDBclient.write(['temperature,location=living_room value=20'],
{'db':'pyexample'}, 204, 'line'):
    print("write failed")
if not IDBclient.write(['temperature,location=living_room value=10'],
{'db':'pyexample'}, 204, 'line'):
    print("write failed")
if not IDBclient.write(['temperature,location=bedroom value=34'],
{'db':'pyexample'}, 204, 'line'):
    print("write failed")
if not IDBclient.write(['temperature,location=bedroom value=23'],
{'db':'pyexample'}, 204, 'line'):
    print("write failed")

results = IDBclient.query('SELECT * FROM temperature')
print(results.raw)
points = results.get_points(tags={'location':'living_room'})

for point in points:
    print("Time: %s, value: %i" % (point['time'], point['value']))

IDBclient.close()
```

Running the above script would result in an output similar to this

```
python3 test-influxDB.py
{'statement_id': 0, 'series': [{'name': 'temperature', 'columns': ['time',
'location', 'value'], 'values': [['2020-02-18T17:18:32.664448994Z',
'living_room', 20], ['2020-02-18T17:18:32.995699334Z', 'living_room', 10],
['2020-02-18T17:18:33.016622684Z', 'bedroom', 34], ['2020-02-
18T17:18:33.046043157Z', 'bedroom', 23]]}]}
Time: 2020-02-18T17:18:32.664448994Z, value: 20
Time: 2020-02-18T17:18:32.995699334Z, value: 10
```

Additional Documentation and Functionality
The influx-python library contains a fair bit of additional functionality that we didn't cover in the article above. There is additional administrative functionality in the client like adding users, managing databases, and dropping measurements, as well as additional objects like SeriesHelper, which provides some convenience functionality for writing points in bulk.
If you're interested in using this library in your projects, it makes sense to spend some time with the API Documentation understanding the functionality that is provided.

https://influxdb-python.readthedocs.io/en/latest/api-documentation.html#influxdbclient

Up to Grafana?

If you plan to install Grafana hereafter, let this script run for a couple of hours so that the database is filled with data. First create this script `fillInfluxDB.py` using the command

```
sudo nano fillInfluxDB.py
```

Add these lines

```
import time
# generate random integer values
from random import seed
from random import randint

from influxdb import InfluxDBClient

# seed random number generator
seed(1)

# replace <password> with the password you assigned to InfluxDB
IDBclient = InfluxDBClient(host='localhost', port=8086, username='admin',
password='<password>')
IDBclient.create_database('grafanatest')
IDBclient.switch_database('grafanatest')

for num in range(0,200):

  print("Pass %i                              " % num)
  value = randint(0, 40)
  measurement = "temperature,location=living_room value=%i" % value
  if not IDBclient.write([measurement], {'db':'grafanatest'}, 204, 'line'):
      print("write failed")

  value = randint(0, 40)
  measurement = "temperature,location=bedroom value=%i" % value
  if not IDBclient.write([measurement], {'db':'grafanatest'}, 204, 'line'):
      print("write failed")

  if num % 10 == 0:
    results = IDBclient.query('SELECT * FROM temperature')
    points = results.get_points(tags={'location':'living_room'})
    for point in points:
        print("Time: %s, Living Room = %i" % (point['time'], point['value']))

    points = results.get_points(tags={'location':'bedroom'})
    for point in points:
        print("Time: %s, Bedroom     = %i" % (point['time'], point['value']))

  for tim in range(300,0,-10):
    print("sleeping %i    " % tim, end="\r")
    time.sleep(10)

IDBclient.close()
```

Run using

```
python3  fillInfluxDB.py
```