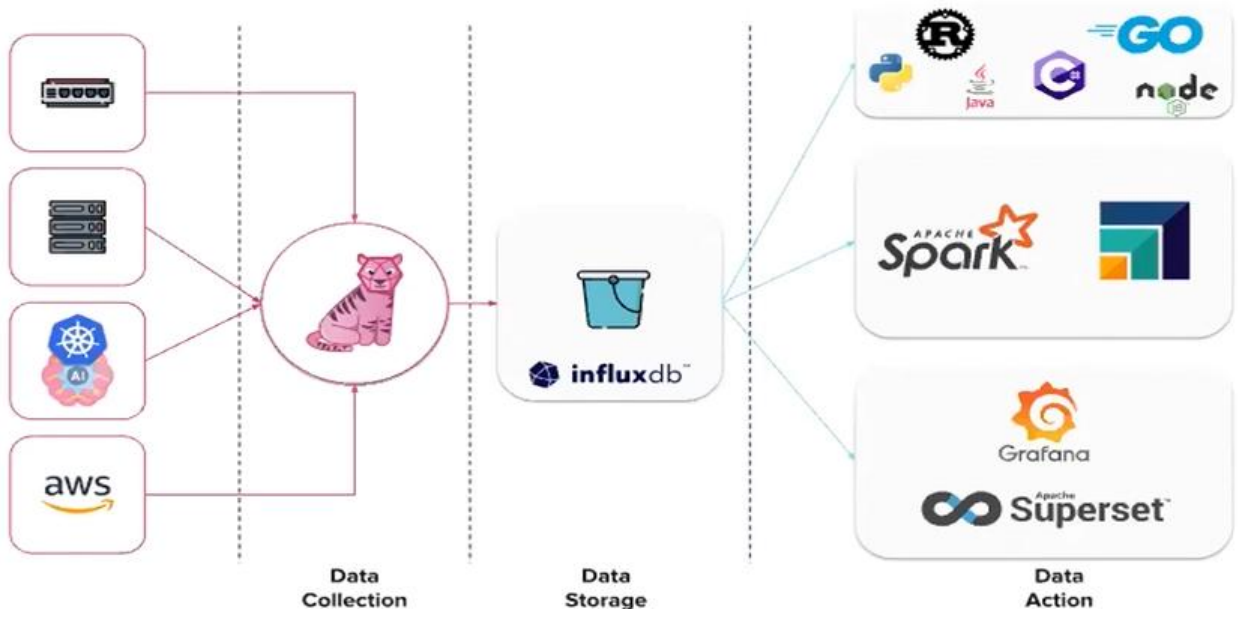
Author: Oscar Muñoz Cueva (omunozcu@cisco.com)
Date: 9th December 2024

TIG (Telegraf, InfluxDB, Grafana) Stack for Infrastructure Monitoring

Contents

Introduction

This is the type of solution we want to build for infrastructure management:

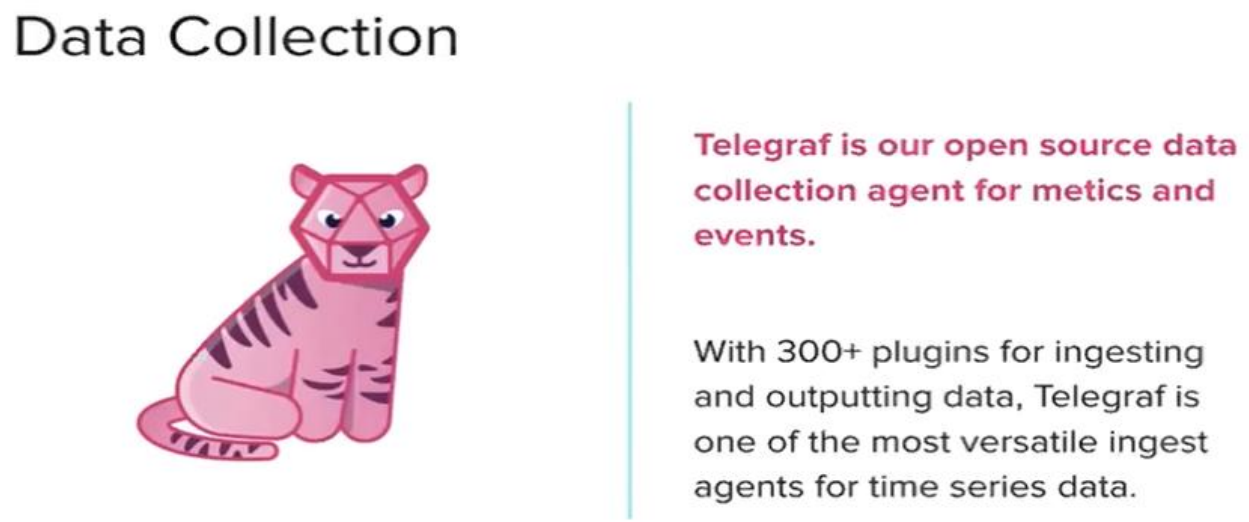


As we see, there's three layers we need to take care of: collection, storage and action on our data. We'll use the TIG stack for that.

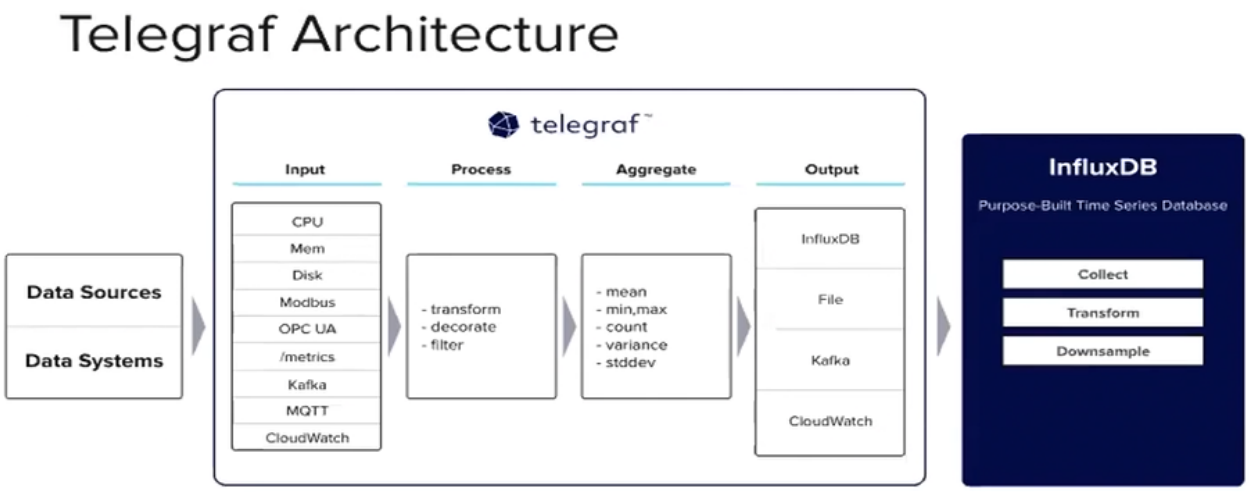
We need to collect relevant data from our infrastructure, whether it's routers, switches, firewalls, servers, VMs, microservices, etc. We will use Telegraf as a collection agent. Telegraf also gives us the option to aggregate or transform the data. Then we need to store the collected (and optionally, transformed) data in a time-series database, which will be achieved by InfluxDB. Last, we want to be able to visualize that data in the form of graphs and eventually, generate alarms based on that data. This will be taken care of by Grafana. Any of the applications used for each of these tasks can be replaced by other software solutions. Both Telegraf and InfluxDB are solutions maintained by the company InfluxData.

Telegraf

Telegraf is an open-source data-collection server agent that collects and reports various system and application metrics. It supports a wide range of input plugins that can gather data from sources like system resources (CPU, memory, disk), network devices, databases, and more. Telegraf then processes this data and sends it to various output destinations. Telegraf is written in Go, but no coding is required to use Telegraf.



Telegraf supports four categories of plugins: input, output, aggregator, and processor.

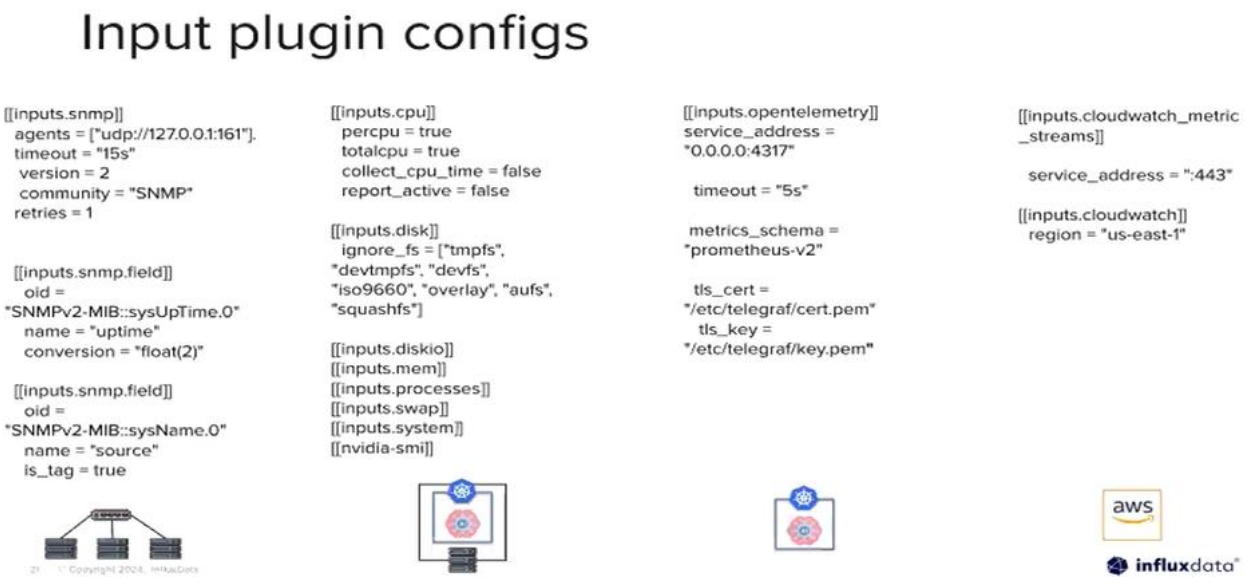


Input plugins are used to collect data. Processor plugins transform, decorate, and filter metrics. Aggregator plugins create aggregate metrics (for example, mean, min, max, quantiles, etc). Output plugins write metrics to various destinations.

Examples of plugins are: gnmi, snmp, iptables, kafka_consumer, Kubernetes, etc.

What we do is: we find each plugin and its parameters in the configuration file. Input collection can occur either with push or pull model, depending on the plugin. Then we have processor plugins to transform, decorate or filter our data. We also have aggregator plugins to pre-aggregate our metrics before sending them to an output. And the output plugins help us send the data to a destination of our choice. If the plugin we need doesn't exist, it's possible to create our own plugins as well. In the case we're studying in this document, i.e. the TIG stack, the destination will be InfluxDB.

All parameters of Telegraf get configured through configuration files.



Let's see an example where our intention is for Telegraf to get local host metrics (input), and then send them to an InfluxDB database (output). In that case, we could uncomment the following section in the Telegraf configuration file for the input:

```
# Read metrics about cpu usage
[[inputs.cpu]]
  ## Whether to report per-cpu stats or not
  percpu = true
  ## Whether to report total system cpu stats or not
  totalcpu = true
  ## If true, collect raw CPU time metrics
  collect_cpu_time = false
  ## If true, compute and report the sum of all non-idle CPU states
  report_active = false

# Read metrics about disk usage by mount point
[[inputs.disk]]
  ## By default stats will be gathered for all mount points.
  ## Set mount_points will restrict the stats to only the specified mount
  ## points.
  # mount_points = ["/"]

  ## Ignore mount points by filesystem type.
  ignore_fs = ["tmpfs", "devtmpfs", "devfs", "iso9660", "overlay", "aufs",
"squashfs"]

# Read metrics about disk IO by device
[[inputs.diskio]]

# Get kernel statistics from /proc/stat
[[inputs.kernel]]
  # no configuration

# Read metrics about memory usage
[[inputs.mem]]
  # no configuration

# Get the number of processes and group them by status
[[inputs.processes]]
  # no configuration

# Read metrics about swap memory usage
[[inputs.swap]]
  # no configuration

# Read metrics about system load & uptime
[[inputs.system]]
```

That was the input. For the output, we want Grafana to send data to InfluxDB, so we have the `[[outputs.influxdb]]` section where we define the settings towards the InfluxDB service. An example (using environment variables) would be:

```
[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
```

```
  ## urls will be written to each interval.
  ##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["http://
"${DOCKER_INFLUXDB_INIT_HOST}:${DOCKER_INFLUXDB_INIT_PORT}"]

  ## Token for authentication.
  token = "$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"

  ## Organization is the name of the organization you wish to write to; must
exist.
  organization = "$DOCKER_INFLUXDB_INIT_ORG"

  ## Destination bucket to write into.
  bucket = "$DOCKER_INFLUXDB_INIT_BUCKET"
  ## Use TLS but skip chain & host verification
  insecure_skip_verify = false
```

Telegraf Plugins for Cisco MDT

Here we can find plugins for Cisco MDT: https://github.com/influxdata/telegraf/blob/master/plugins/inputs/cisco_telemetry_mdt/README.md

InfluxData offers some documentation about it too: https://www.influxdata.com/integration/cisco-model-driven-telemetry/

The Cisco Model-Driven Telemetry Telegraf Plugin will allow you to consume this data in InfluxDB. Once in InfluxDB, you can visualize the data in InfluxDB, Grafana, or your own custom dashboards. This plugin works for from Cisco IOS XR, IOS XE and NX-OS platforms. It supports TCP & GRPC dialout transports. GRPC-based transport can utilize TLS for authentication and encryption. Telemetry data is expected to be GPB-KV (self-describing-gpb) encoded.

A configuration example for a switch can be found here: https://www.cisco.com/c/en/us/support/docs/wireless/catalyst-9800-series-wireless-controllers/222054-configure-advanced-grpc-workflow-with-te.html

And for IOS XR here: https://xrdocs.io/programmability/blogs/Dial-out-MDT-with-TIG/

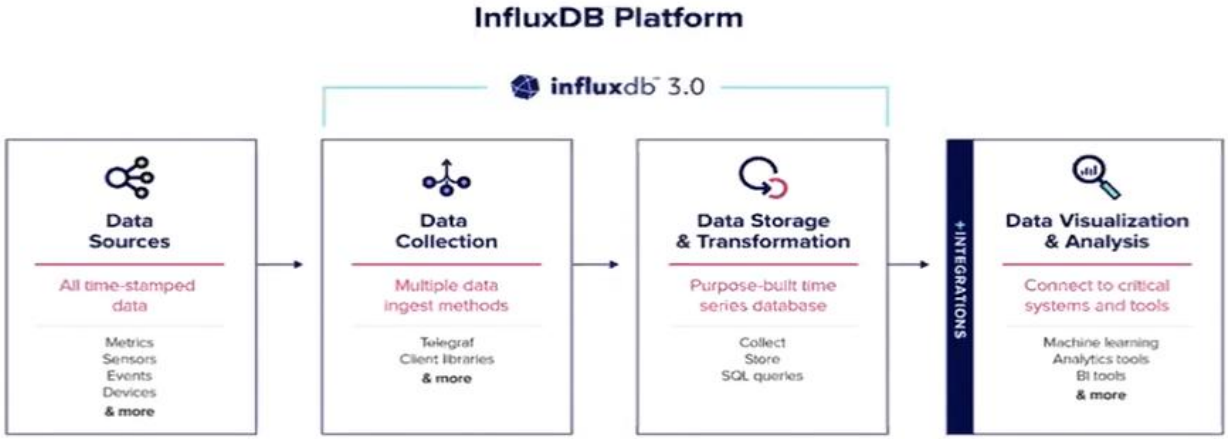
In order to configure the plugin, we need to uncomment and adapt the values of the corresponding section of the `telegraf.conf` configuration file. For example:

```
# Cisco model-drive telemetry (MDT) input plugin for IOS XR, IOS XE and NX-OS
# platforms
[[inputs.cisco_telemetry_mdt]]
# ## Telemetry transport can be “tecp” or “grpc”. TLS is only supported when
# ## using the grpc transport.
transport = "grpc"
#
# ## Address and port to host telemetry listener
```

```
service_address = ":57100"
#
# ## Enable TLS; grpc transport only.
# # tls_cert = "/etc/telegraf/cert.pem"
# # tls_key = "/etc/telegraf/key.pem"
#
# ## Enable TLS client authentication and define allowed CA certificates:
# ## grpc transport only
# # tls_allowed_cacerts = ["/etc/telegraf/clientca.pem"]
#
# ## Define (for certain nested telemetry measurements with embedded tags)
# ## which fields are tagged
# # embedded_tags = ["Cisco-IOS-XR-qos-ma-oper:qos/interface-
table/interface/input/service-policy-names/service-policy-
instance/statistics/class-stas/class-name"]
#
# ## Define aliases to map telemetry encoding paths to simple measurement
# ## names
# [inputs.cisco_telemetry_mdt.aliases]
#   ifstats = "ietf-interfaces:interfaces-state/interface/statistics"
# ## Define Property Xformation, please refer README
# [inputs.cisco_telemetry_mdt.dmes]
#   ModTs = "ignore"
#   CreateTs = "ignore"
```

InfluxDB

InfluxDB is a database purpose-built for handling time series data at massive scale for real-time analytics. It can be used for ingesting, storing and analyzing metrics, events and traces. It supports queries in SQL and InfluxQL, as well as APIs for retrieving and analyzing time-series data.



Concepts: Data Model

Bucket
- All InfluxDB data is stored in a bucket. A bucket combines the concept of a database and a retention period (the duration of time that each data point persists).

Measurement
- A name to a group of data at a high level (Table)

Tag set
- A set of key-value pairs to group data at a low level (values are strings)

Field set
- A set of key-value pairs to represent data (values are numerical & strings)

Timestamp
- Time of the data with nanosecond precision

Series
- A unique combination of measure+tags

A bucket is, in essence, a database in SQL with a corresponding retention policy or period.

Data Storage

- Writing points to InfluxDB uses Line Protocol, which takes the following format:

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

Measurement	Tag Set	Field Set	Timestamp
server	,hostname=server02,us_west=az	cpu=24.5,mem=12.4	1234567890000000

The default port InfluxDB runs on is 8086.

Grafana

Grafana acts as the visualization and dashboarding component of the TIG stack. It connects to InfluxDB (or other data sources) to pull the time-series data and allows users to create rich and interactive dashboards.

Data Action

Grafana is an open-source data visualization and monitoring platform.

Allows users to create interactive dashboards for real-time data analysis and tracking of metrics across various data sources.

The default port where Grafana runs on is 3000.

There are two flavors of Grafana: open source and cloud. You can choose from two plugins: FlightSQL or InfluxQL.

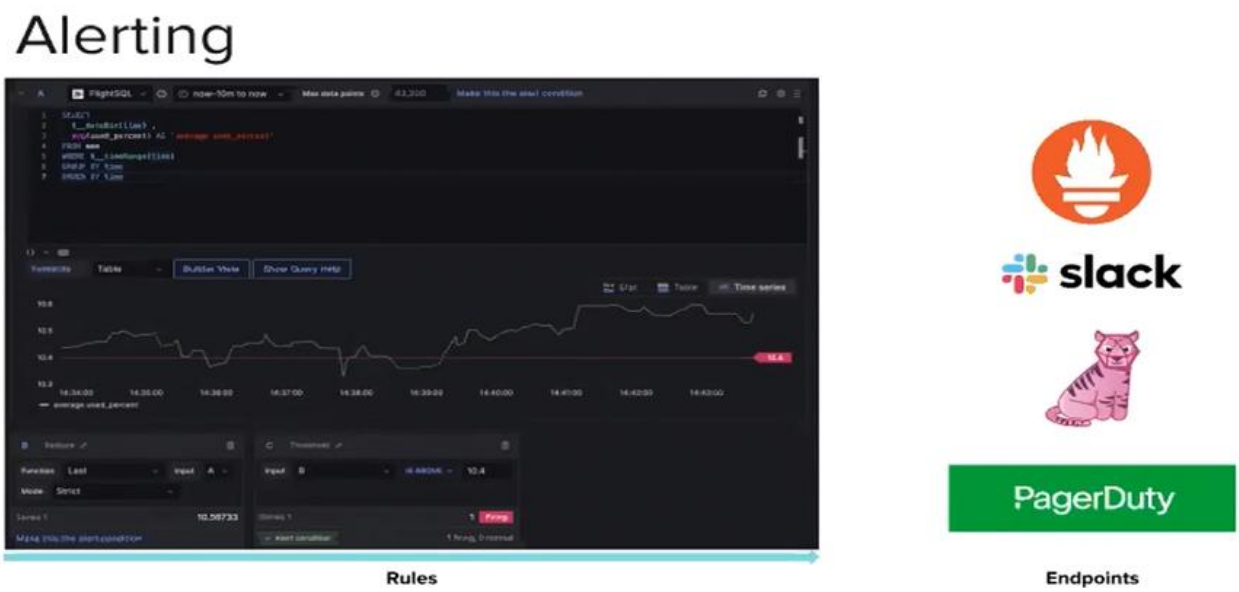
Useful Queries

SQL Command

```
SELECT
  $__dateBin(time) ,
  avg(usage_user) AS 'usage_user',
  avg(usage_system) AS 'usage_system',
  avg(usage_idle) AS 'usage_idle'
FROM cpu
WHERE cpu='cpu-total' AND  $__timeRange(time)
GROUP BY 1
ORDER BY time
```


```
SELECT
  selector_last(total, time)['time'] AS time,
  selector_last(total, time)['value'] / 1024 / 1024 / 1024 As total
FROM mem
WHERE host='${linux_host}' AND $__timeRange(time)
ORDER BY time
```

We have selector functions and aggregation functions.



You can define alerting thresholds for your metrics and notify endpoints.

Installation of the TIG Stack

You can install each of the components of the TIG stack on a different server (better performance and streamlined monitoring) or combine them all in one. A prerequisite is that you have root privileges on each of the servers. The possibility of deploying the stack in the form of Docker containers is also possible.

The installation process doesn't differ much from any other Linux application. You download, the software, install it and then start the corresponding service. Then we need to apply some basic configurations (create admin users, create user accounts for the other pieces of the solution, configure authentication, etc).

In order for Telegraf and Grafana to communicate with InfluxDB, you need to generate an admin API token.

Setting up a TIG Stack with Docker Compose

We're going to use the following versions of the images:

```
$ docker image ls
REPOSITORY                  TAG           IMAGE ID       CREATED          SIZE
influxdb                    2.1.1         0d8d6b98c43f   2 years ago      346MB
telegraf                    1.19          58d08adb2f70   2 years ago      342MB
grafana/grafana-oss         8.4.3         8d0b77430ee9   2 years ago      278MB
```

There is a number of parameters we have to configure as environment variables. We can add them individually in the `docker-compose.yml` file, or we can add them all in a specific file, which we'll do here:

```
$ cat .env
DOCKER_INFLUXDB_INIT_MODE=setup

## Environment variables used during the setup and operation of the stack
#

# Primary InfluxDB admin/superuser credentials
#
DOCKER_INFLUXDB_INIT_USERNAME=cisco       # Used to log in the UI
DOCKER_INFLUXDB_INIT_PASSWORD=cisco123    # Used to log in the UI
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=89fa7cc4671363ed7600167e64c051fd96baa44e3b43
3075f4fe91049df79826                      # See explanation below

# Primary InfluxDB organization & bucket definitions
#
DOCKER_INFLUXDB_INIT_ORG=omunozcu
DOCKER_INFLUXDB_INIT_BUCKET=cisco_mdt_bucket    # DB name to be used

# Primary InfluxDB bucket retention period
#
# NOTE: Valid units are nanoseconds (ns), microseconds(us), milliseconds (ms)
# seconds (s), minutes (m), hours (h), days (d), and weeks (w).
DOCKER_INFLUXDB_INIT_RETENTION=4d               # Adapt as necessary

# InfluxDB port & hostname definitions
#
DOCKER_INFLUXDB_INIT_PORT=8086            # Default port, we could change it
DOCKER_INFLUXDB_INIT_HOST=influxdb        # Used in Grafana configuration

# Telegraf configuration file
#
# Will be mounted to container and used as telegraf configuration
TELEGRAF_CFG_PATH=./telegraf/telegraf.conf      # Location in host

# Grafana port definition
GRAFANA_PORT=3000                         # Default port, we could change it
```

Regarding the variable `DOCKER_INFLUXDB_INIT_ADMIN_TOKEN`, it's the API token needed for configuration between Telegraf and InfluxDB. One possibility is to generate a random 32-hexadecimal characters strings in the following way:

```
$ openssl rand -hex 32
f129ba23bdb90b05428cd45b75433dd99567635b4d0107f6c70ae4c2374376fa
```

Notice we had to specify username and password for InfluxDB. We'll use that one to log in the WebUI. Regarding Grafana, the default username and password is admin/admin. You will be asked to change the password the first time.

Let's look at the `docker-compose.yml` file now:

```
networks:               # optional part
```

```
  mgmt:
    driver_opts:
      com.docker.network.container_iface_prefix: xr-3
    ipam:
      config:
      - subnet: 172.3.0.0/24
services:
  influxdb:                          # InfluxDB container
    image: influxdb:2.1.1
    volumes:
      - influxdb-storage:/var/lib/influxdb2:rw
    env_file:
      - .env                   # Environment variable file created before
    entrypoint: ["./entrypoint.sh"]       # Script (explanation below)
    restart: on-failure:10
    ports:
      - ${DOCKER_INFLUXDB_INIT_PORT}:8086 # Port mapping at host
    networks:                  # optional part
      mgmt:
        ipv4_address: 172.3.0.202
  telegraf:                          # Telegraf container
    image: telegraf:1.19
    volumes:
      - ${TELEGRAF_CFG_PATH}:/etc/telegraf/telegraf.conf:rw # Conf file
    env_file:
      - .env                   # Environment variable file created before
    depends_on:                # Telegraf shouldn't exist without InfluxDB
      - influxdb
    networks:                  # optional part
      mgmt:
        ipv4_address: 172.3.0.201
  grafana:                           # Grafana container
    image: grafana/grafana-oss:8.4.3
    volumes:
      - grafana-storage:/var/lib/grafana:rw
    depends_on:
      - influxdb               # Grafana shouldn't exist without InfluxDB
    ports:
      - ${GRAFANA_PORT}:3000               # Port mapping at host
    networks:                  # optional part
      mgmt:
        ipv4_address: 172.3.0.203

version: '2.4'
volumes:
  grafana-storage:
  influxdb-storage:
```

Regarding the following line under `influxdb`:

```
entrypoint: ["./entrypoint.sh"]
```

It contains the commands that we will execute to initialize the influxdb service:

```
$ cat entrypoint.sh
#!/bin/bash

# Protects script from continuing with an error
set -eu -o pipefail

# Ensures environment variables are set
export DOCKER_INFLUXDB_INIT_MODE=$DOCKER_INFLUXDB_INIT_MODE
export DOCKER_INFLUXDB_INIT_USERNAME=$DOCKER_INFLUXDB_INIT_USERNAME
export DOCKER_INFLUXDB_INIT_PASSWORD=$DOCKER_INFLUXDB_INIT_PASSWORD
export DOCKER_INFLUXDB_INIT_ORG=$DOCKER_INFLUXDB_INIT_ORG
export DOCKER_INFLUXDB_INIT_BUCKET=$DOCKER_INFLUXDB_INIT_BUCKET
export DOCKER_INFLUXDB_INIT_RETENTION=$DOCKER_INFLUXDB_INIT_RETENTION
export DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN
export DOCKER_INFLUXDB_INIT_PORT=$DOCKER_INFLUXDB_INIT_PORT
export DOCKER_INFLUXDB_INIT_HOST=$DOCKER_INFLUXDB_INIT_HOST

# Conducts initial InfluxDB using the CLI
influx setup --skip-verify --bucket ${DOCKER_INFLUXDB_INIT_BUCKET} --
retention ${DOCKER_INFLUXDB_INIT_RETENTION} --token
${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN} --org ${DOCKER_INFLUXDB_INIT_ORG} --
username ${DOCKER_INFLUXDB_INIT_USERNAME} --password
${DOCKER_INFLUXDB_INIT_PASSWORD} --host
http://${DOCKER_INFLUXDB_INIT_HOST}:8086 --force
```

We could omit the execution of that script as part of our `docker-compose` file, but then the first thing we'd need to do is to do a docker exec to the `influxdb` container and run those commands manually.

Alright, with all that information, we could spin up our project:

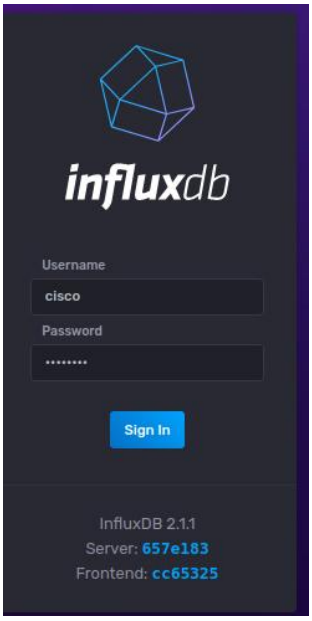
```
$ docker-compose up -d
$ docker ps
CONTAINER ID  IMAGE                         COMMAND                   CREATED      STATUS       PORTS
NAMES
4c01100e87e8  telegraf:1.19                 "/entrypoint.sh tele…"  3 hours ago  Up 3 hours  8092/udp, 8125/udp,
8094/tcp                      telegraf_1
1267b5ace154  grafana/grafana-oss:8.4.3  "/run.sh"                 3 hours ago  Up 3 hours  0.0.0.0:3000-
>3000/tcp, :::3000->3000/tcp  grafana_1
d5a5e2703eca  influxdb:2.1.1                "./entrypoint.sh"         3 hours ago  Up 3 hours  0.0.0.0:8086-
>8086/tcp, :::8086->8086/tcp  influxdb_1
```

You can use the following URLs in a browser to log into the UIs:

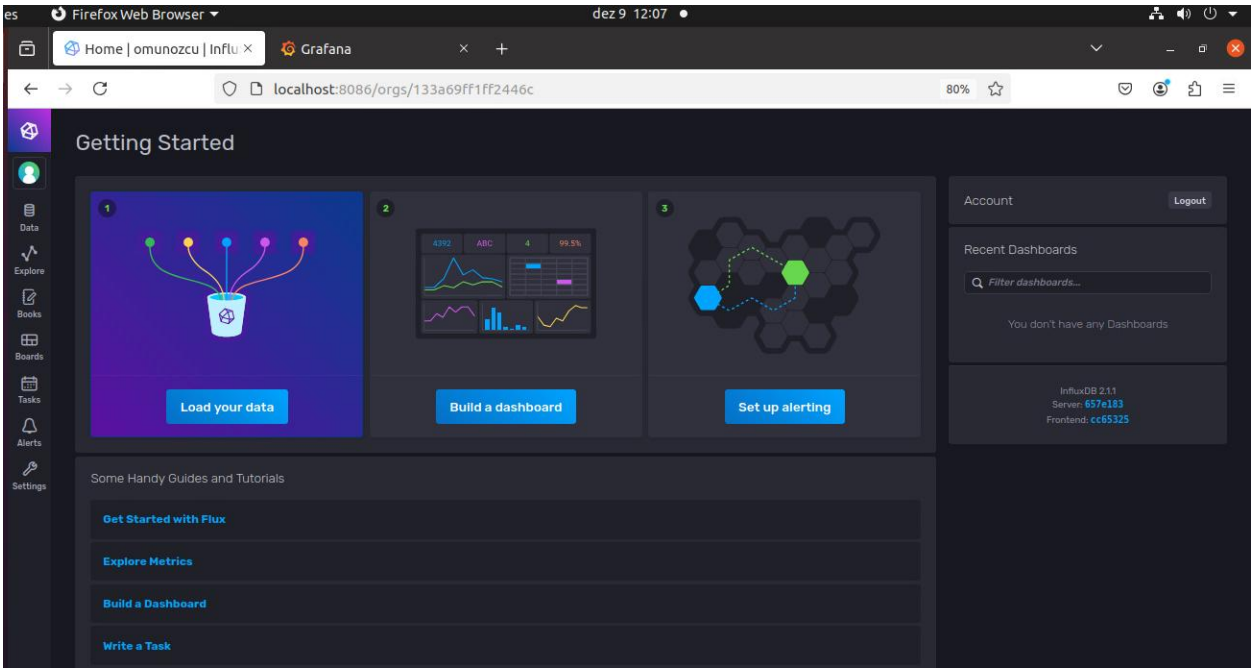
- InfluxDB: http://<host_ip_address>:DOCKER_INFLUXDB_INIT_PORT
- Grafana: http://<host_ip_address>:GRAFANA_PORT

Notice the port numbers are the environment variables we defined in our `docker-compose.yml` file or environment variables file. If you're trying it from the Docker host, you can use locahost instead of the IP address.

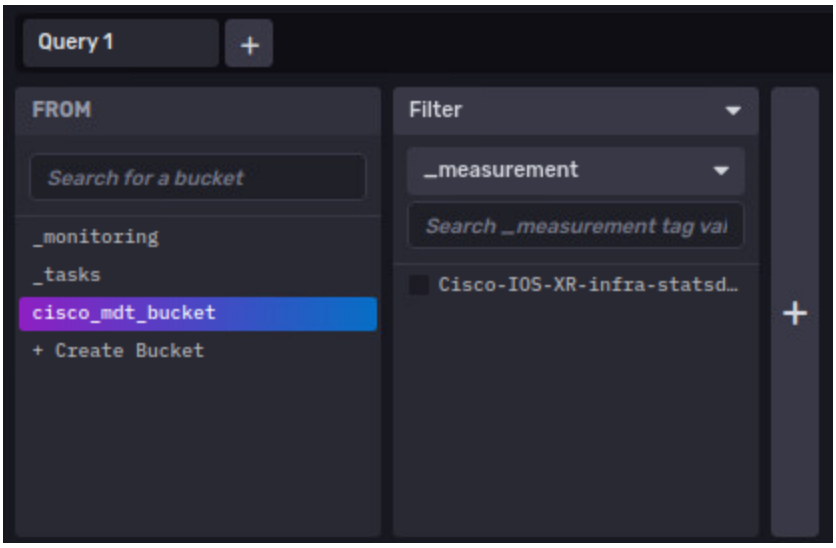
We'll start by configuring InfluxDB. And please remember, we already accomplished a part of the configuration with the help of a script referenced in the `docker-compose.yml` file. Now we'll use the UI through https://localhost:8086:



After log in, we'll land here:



If we click on "Explore" at the left hand side menu, we'll see the following:



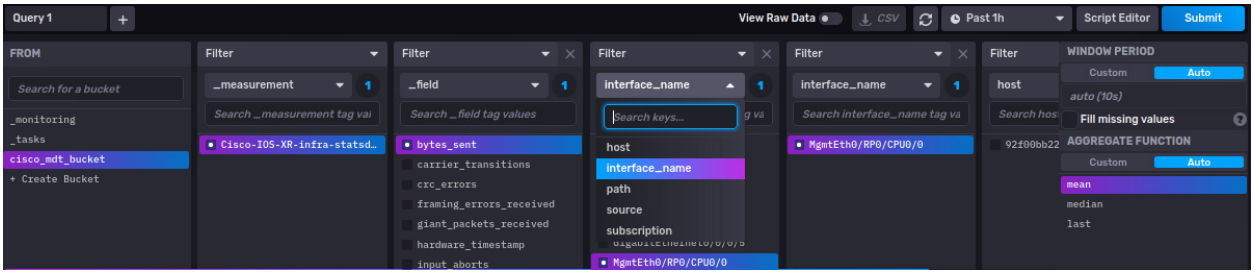
As we can see, the `cisco_mdt_bucket` we defined is there, and there seems to be sensor under the `_measurement` menu. The bucket was defined as an environment variable in the `.env` file:

```
DOCKER_INFLUXDB_INIT_BUCKET=cisco_mdt_bucket
```

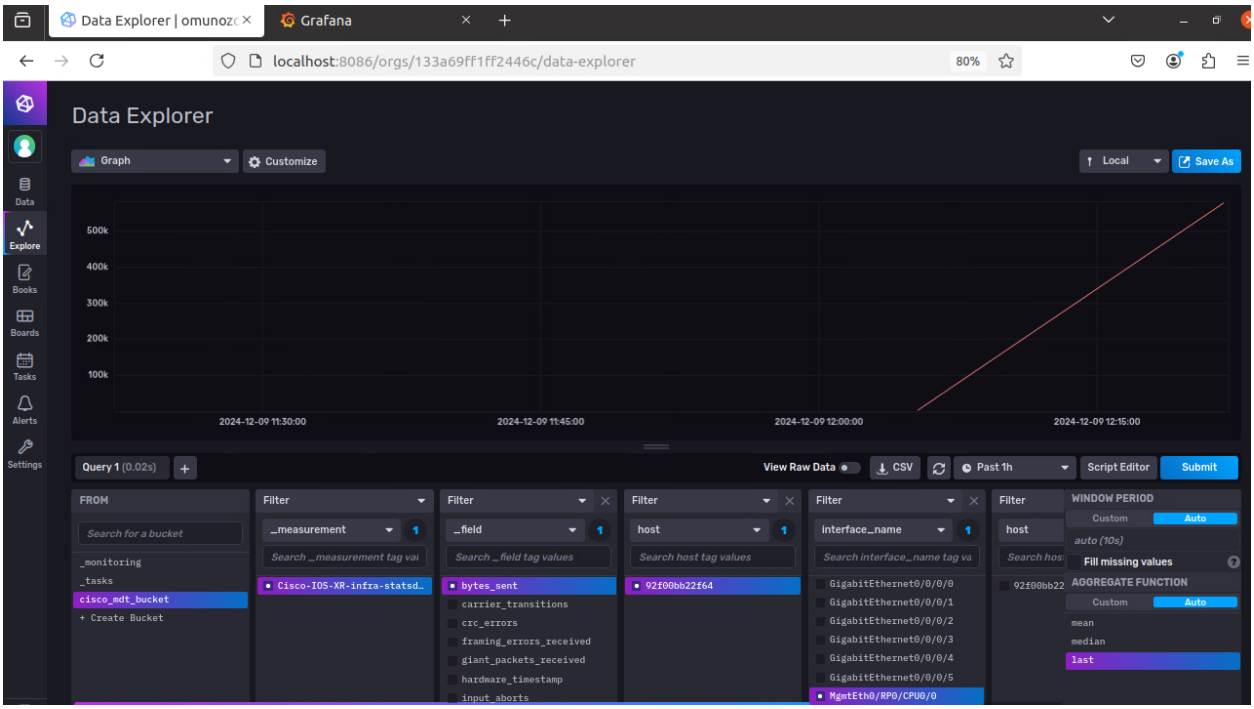
Here is where the MDT information sent from the devices will land. Let's see the router configuration we have applied as an example:

```
telemetry model-driven
 destination-group TELEGRAF
  address-family ipv4 172.3.0.201 port 57100     # Telegraf container IP addr
   encoding self-describing-gpb
   protocol grpc no-tls
  !
 !
 sensor-group SENTEST1
  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-counters
 !
 subscription SUBTEST1
  sensor-group-id SENTEST1 sample-interval 30000
  destination-id TELEGRAF
 !
!
```

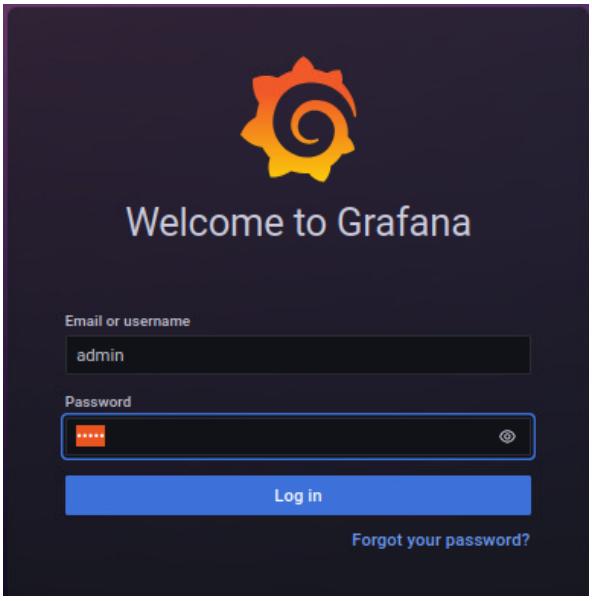
The router is sending telemetry related to a specific sensor path in the `Cisco-IOS-XR-infra-statsd-oper` YANG model to the Telegraf container every 30 seconds. Let's see if this information is making its way to InfluxDB:



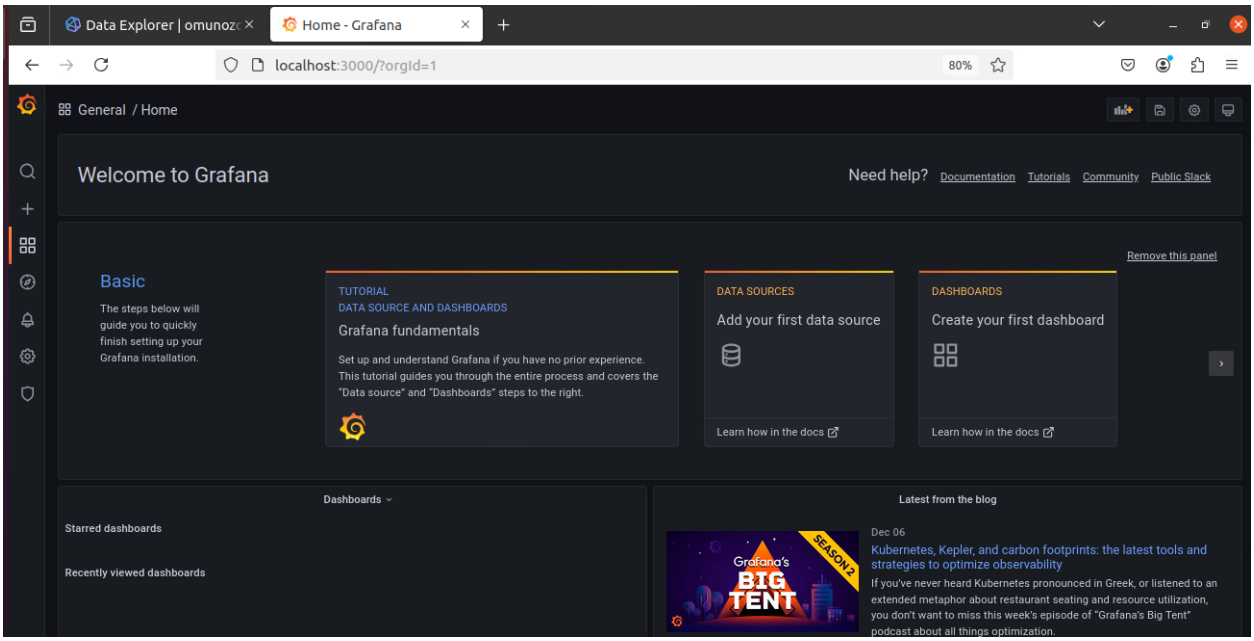
We can drill down in the information sent by the device and select the field(s) we're interested in. When we're ready, we can click on "Submit". We will be able to visualize the information then:



This is good news, the information is in InfluxDB. We'll configure Grafana next:



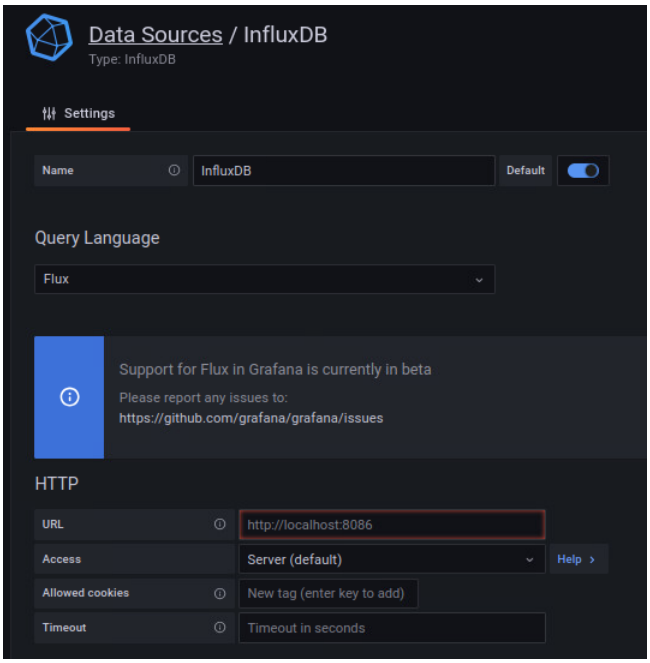
The first time we log into Grafana, we have to use the default credentials admin/admin, and the system will ask us to set a new password. After doing that, the landing page looks like this:

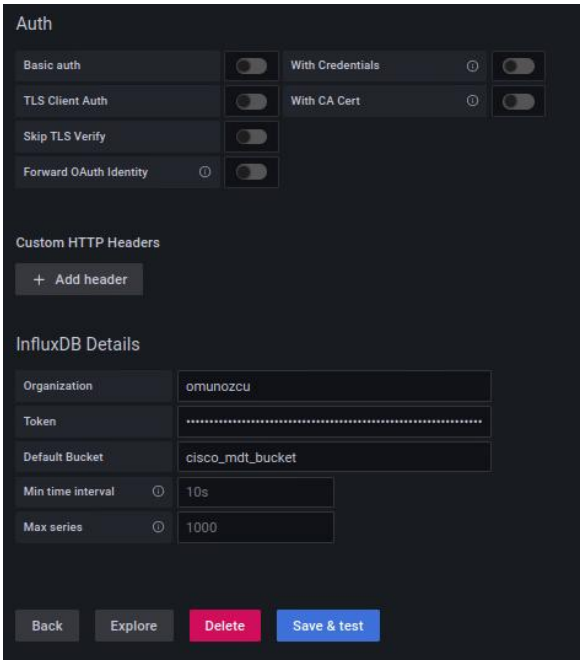


What we have to do now, is to use InfluxDB as a data source. So, we click on the “DATA SOURCES – Add your first data source” box.

A menu will open and we’ll select InfluxDB as data source type. We’ll get to a menu with different fields to fill. In the version I’m using at the moment, the two supported query languages are InfluxQL and Flux. I’ll choose Flux and show in a moment that we don’t actually need to know this language.

We’ll fill the information like this:

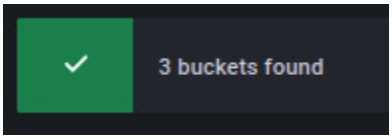




The URL is not filled yet, but we can't use localhost this time. If we're using Docker containers running on the same host machine, like we're doing now, we have to type either the InfluxDB container's name (found out with the `docker ps` command), or the InfluxDB container's IP address (found out via `docker container inspect` command followed by the InfluxDB container name), followed by the port number InfluxDB is running on.

I have disabled basic auth, which was enabled by default. I've also added the token value, which we defined as `$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN`, the value of `$DOCKER_INFLUXDB_INIT_ORG` as organization and the value of `$DOCKER_INFLUXDB_INIT_BUCKET` as default bucket.

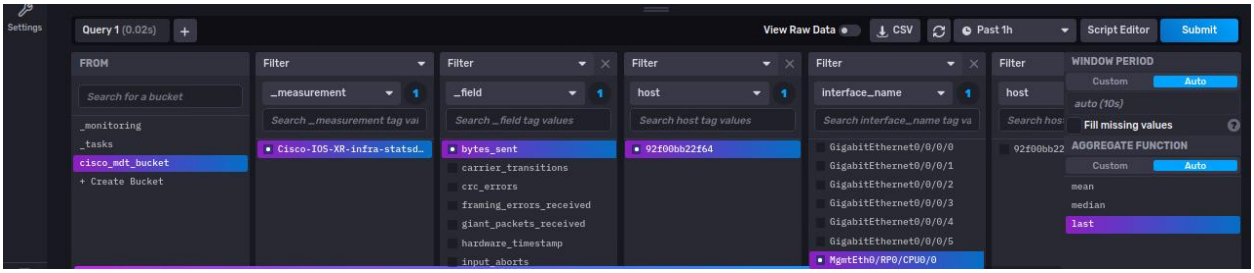
Once I have filled all the information, I click "Save & Test" and things should go well:



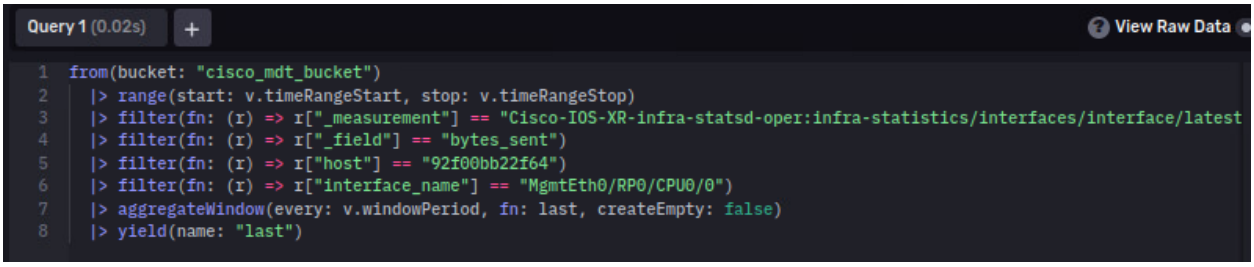
The three buckets are due to the fact that I have configured 3 routers to send MDT to Telegraf.

We have set up the bucket(s) we had in our InfluxDB container as data source in Grafana, but we need to choose which information from that bucket to visualize. Remember we chose Flux as query language, so we would need to write a Flux query ourselves…

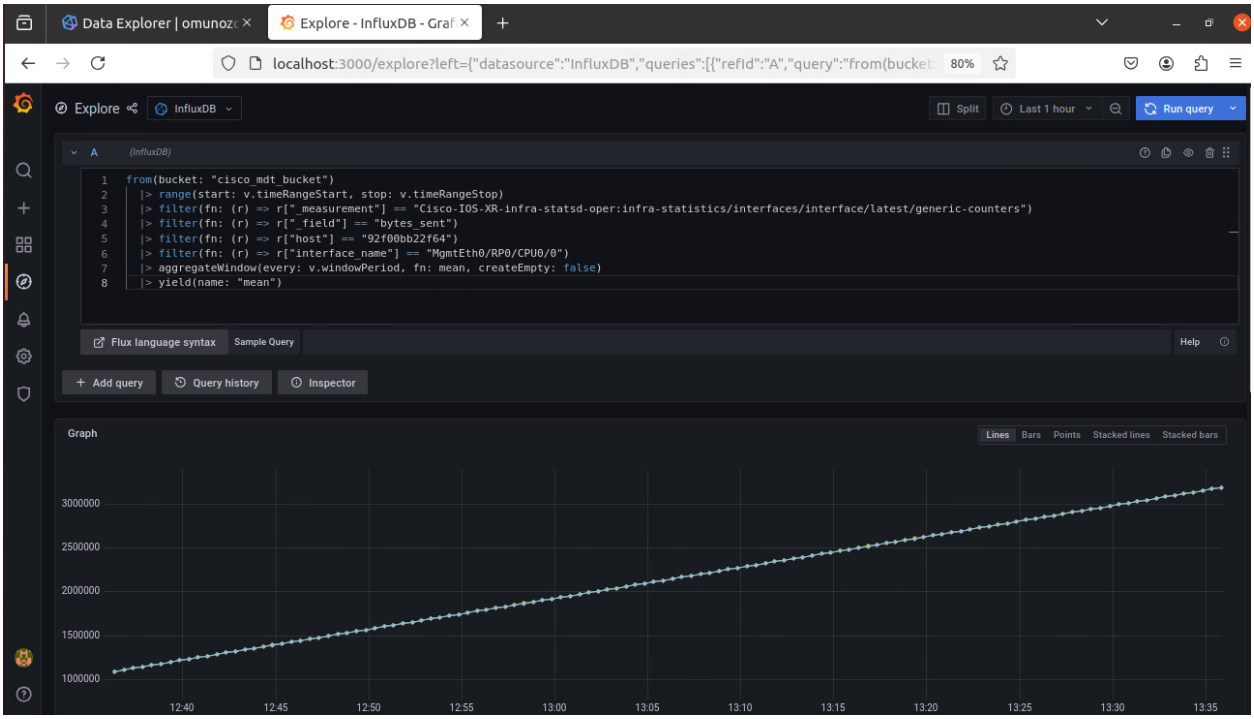
Luckily, we can get the query directly from InfluxDB by using the "Script Editor" functionality (next to the "Submit" button):



We just have to choose which data we want Grafana to query from all the fields contained in the sensor. After doing that, if we click “Script Editor”, we’ll see the query for the data we had selected:

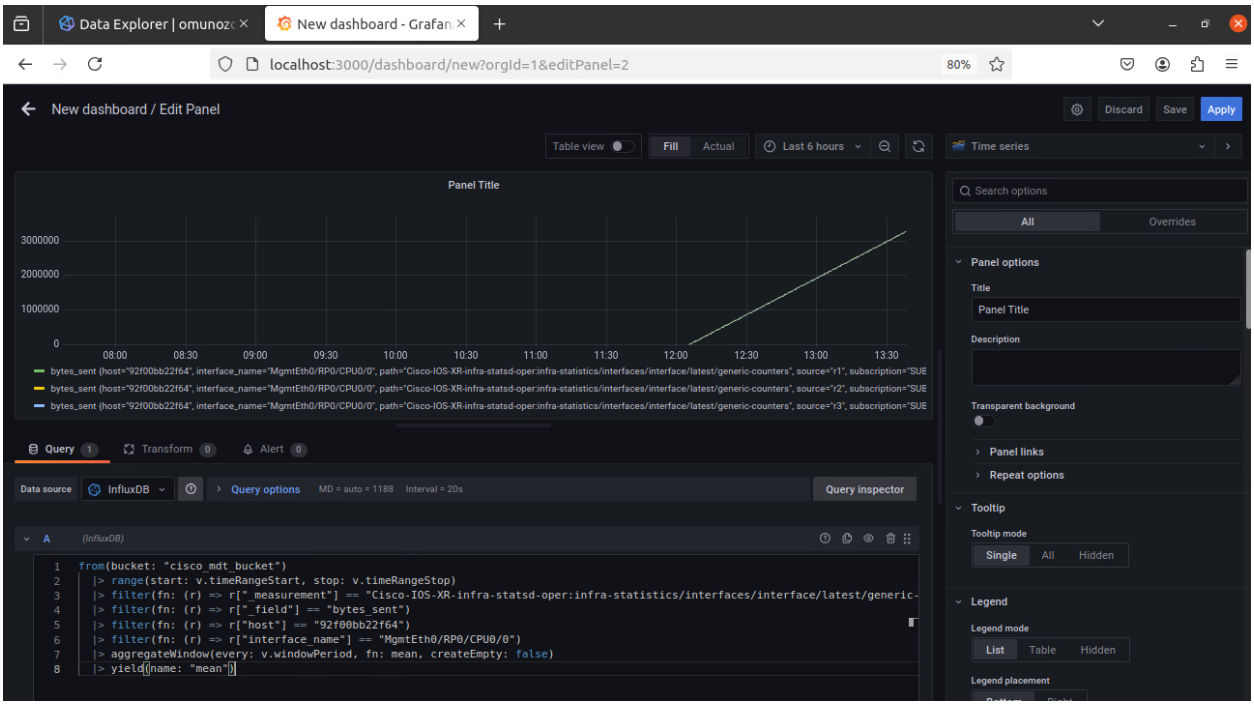


Now we can copy that text and paste it in Grafana, under :



We can verify that the data is accessible to Grafana.

The next step would be to create dashboards with the data we want to display, and customize them making use of all the available settings. We can click on the “plus” sign at the left-hand side menu and then on “Create dashboard”, “Add panel”, etc.



At the right hand side you have a lot of customization options for the visual representation of the data. Obviously, you can rename your dashboards, panels, etc too.