3. InfluxDB, MQTT and Telegraf.

BIP Course, November, 2023

etsit escuela técnica superior de ingeniería de telecomunicación
Universidad Politécnica de Cartagena
MIEMBRO DE
EUROPEAN UNIVERSITY OF TECHNOLOGY

1 InfluxDB

InfluxDB is an open source time series platform. This includes APIs for storing and querying data, processing it in the background for ETL or monitoring and alerting purposes, user dashboards, and visualizing and exploring the data and more.

1.1 Introduction

InfluxDB is an open source time series database, highly scalable and a popular option for storing and querying time series data in a wide variety of application, specially IoT.

Key Characteristics of InfluxDB:

- **Time Series Storing** : InfluxDB is specially designed to store and query time series data.
- **Scalability** : InfluxDB is hihly scalable, capable of handling great volume of data.
- **Query Language** : InfluxDB uses a query language called InfluxQL, that allows efficient and flexible queries into time series data.
- **Data Retention** : InfluxDB offers data retention capabilities, allowing the customization of the data lifetime.
- **Integration With Other Services** : InfluxDB can work seamlessly with other services and tools, like Grafana or Telegraf.
- **Open Source** : InfluxDB is an open source project with MIT license.

1.2 Intallation and Configuration

For the purpose of this practice session, we are going to use Ubuntu as the operative system. Thus, in order to install and set up Influx we will go to `https://portal.influxdata.com/downloads/`, select Ubuntu&Debian platform and copy paste the command in the terminal.

To start the InfluxDB service then type in the terminal:

```
sudo service influxdb start
```

Then restart the system and verify the service is running:

```
sudo service influxdb status
```

Another way to start the service is by typing:

```
influxd
```

This start the daemon and allows you to use the UI in `http://localhost:8086/`.

The UI will guide you to create an initial user, you will have to:

1. Enter a Username for your initial user.
2. Enter a Password and Confirm Password for your user.
3. Enter your initial Organization Name.
4. Enter your initial Bucket Name.
5. Click continue
6. Copy the provided operator API token and store it for safe keeping.

1.3 Write and Query Data

InfluxDB provides many different options for ingesting or writing data, including the following:

- Influx user interface (UI)
- InfluxDB HTTP API
- influx CLI
- Telegraf
- InfluxDB client libraries

The data is written using Line Protocol, if using tools like Telegraf or InfluxDB client libraries, they will build the line protocol for you.

All data written to InfluxDB is written using line protocol, a text-based format that lets you provide the necessary information to write a data point to InfluxDB. Each line of line protocol contains the following elements, with the ones that are required in bold:

- **Measurement** : String that identifies the measurement to store the data in.
- Tag set: Comma-delimited list of key value pairs, each representing a tag. Tag keys and values are unquoted strings. Spaces, commas, and equal characters must be escaped.
- **Field set** : Comma-delimited list key value pairs, each representing a field. Field keys are unquoted strings. Spaces and commas must be escaped. Field values can be strings (quoted), floats, integers, unsigned integers, or booleans.
- Timestamp : Unix timestamp associated with the data. InfluxDB supports up to nanosecond precision.

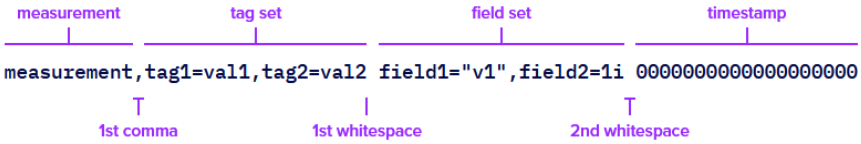

Figure 1: InfluxDB line protocol elements parsing

This is how the line protocol elements are parsed: Measurement is everything before the first unescaped comma before the first whitespace. The tag set are the key-value pairs between the first unescaped comma and the first unescaped whitespace. Field set are the key-value pairs between the first and second unescaped whitespaces and the timestamp is the integer value after the second unescaped whitespace. Bear in mind that lines are separated by the new line character (\n) and line protocol is whitespace sensitive.

Now we can try to write dummy data to influxDB. Consider the following case, we have a temperature and humidity sensor for each room in a house. To collect this data we follow the following schema:

- measurement : home.
  - tags
    - room : Kitchen, living room, bathroom, etc.
  - fields
    - temp : temperature in °C (float).
    - hum : percent humidity (float).
  - timestamp

The line protocol would look similar to this:

```
home,room=LivingRoom temp=21.1,hum=35.9 1699351200
home,room=Kitchen temp=21.0,hum=35.9 1699351200
home,room=LivingRoom temp=21.4,hum=35.9 1699351300
home,room=Kitchen temp=23.0,hum=36.2 1699351300
home,room=LivingRoom temp=21.8,hum=36.0 1699351400
home,room=Kitchen temp=22.7,hum=36.1 1699351400
home,room=LivingRoom temp=22.2,hum=36.0 1699351500
home,room=Kitchen temp=22.4,hum=36.0 1699351500
home,room=LivingRoom temp=22.2,hum=35.9 1699351600
home,room=Kitchen temp=22.5,hum=36.0 1699351600
home,room=LivingRoom temp=22.4,hum=36.0 1699351700
home,room=Kitchen temp=22.8,hum=36.5 1699351700
home,room=LivingRoom temp=22.3,hum=36.1 1699351800
home,room=Kitchen temp=22.8,hum=36.3 1641045600
home,room=LivingRoom temp=22.3,hum=36.1 1699351700
```

And to add this data to influxDB, once in `http://localhost:8086/`, navigate to **Load Data** > **Buckets** using the left bar. Click on **+ add data** on the bucket you want and select **line protocol**. Select **enter manually** and choose the precision used in the timestamp, seconds for the earlier example. Paste the line protocol above and click on **write data**.

Now that we have some dummy data on our bucket, we can try and build some queries to see if it works. As it happened with writing data, there are several tools that you can use to query from an influxDB database, but we are going to focus on doing so manually.

Older versions of influxDB used InfluxQL, a SQL-like language to query data, however, modern versions of influxDB use Flux.

Flux is a functional scripting language designed to query and process data from InfluxDB and other data sources. When querying InfluxDB with Flux, there are three primary functions you use:

- from(): Queries data from an InfluxDB bucket.
- range(): Filters data based on time bounds. Flux requires "bounded" queries—queries limited to a specific time range.
- filter(): Filters data based on column values. Each row is represented by r and each column is represented by a property of r. You can apply multiple subsequent filters.

Then Flux uses the pipe-forward operator (|>) to direct the output of one function as input to the next function. Now, we have all the building blocks to query our example data:

```
from(bucket: "name-of-your-bucket")
|> range(start: 2022-01-01T08:00:00Z, stop: 2022-01-01T20:00:01Z)
|> filter(fn: (r) => r._measurement == "home")
|> filter(fn: (r) => r._field == "hum" or r._field == "temp")
```

Now we just need to execute our query, in order to do that navigate to **Data Explorer** on the left bar. You will be presented with two options to query data:

- Query builder: Visual query builder that lets you select the time range, measurement, tags, and fields to query.
- Script Editor: In-browser code editor for composing and running Flux scripts.

Using query builder it would only be necessary to select the bucket and measurements of interest from the columns dropwdown menus. With the Script Editor, just click on it, write our query and click on submit.

2 MQTT

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.

2.1 Introduction

MQTT is a messaging protocol that uses a publish and suscribe model. This model makes it possible to send messages to multiple clients or none at all.

It is similar to a TV, a TV broadcaster broadcasts a TV program using a specific channel and a viewer tunes into this channel to view the broadcast. If you tune to this channel you will receive the message, if you don't then it will be "lost". There is no direct connection between broadcaster and viewer.

In MQTT a publisher publishes messages on a topic and a subscriber must subscribe to that topic to view the message. MQTT requires the use of a central broker to act as shown in the diagram below:

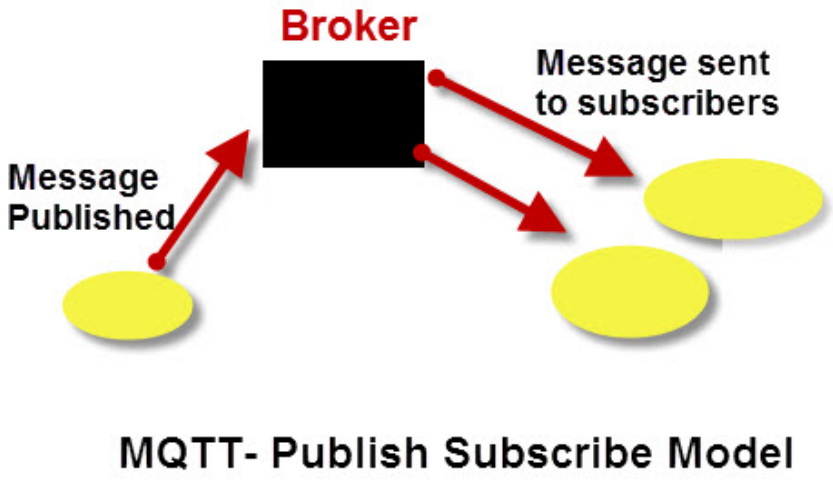

Figure 2: MQQT publish subscribe model

A client is free to publish on any topic it chooses. Currently there are no reserved topics. However brokers can restrict access to topics. A client cannot publish a message to another client directly and doesn't know if any clients receive that message.

A client can only publish messages to a single topic, and cannot publish to a group of topics. However a message can be received by a group of clients if they subscribe to the same topic.

2.1.1 Clients ID

All clients are required to have a client name or ID. The client name is used by the MQTT broker to track subscriptions and must also be unique. If you attempt to connect to an MQTT broker with the same name as an existing client then the existing client connection is dropped because most MQTT clients will attempt to reconnect following a disconnect and this can result in a loop of disconnect and connect.

2.1.2 Clean Sessions

MQTT clients by default establish a clean session with a broker. A clean session is one in which the broker is not expected to remember anything about the client when it disconnects.

With a non clean session the broker will remember client subscriptions and may hold undelivered messages for the client. However this depends on the Quality of service used when subscribing to topics, and the quality of service used when publishing to those topics.

2.1.3 Last Will Messages

MQTT also uses last will messages to notify a subscriber that the publisher is unavailable due to network outage. The last will message is set by the publishing client, and is set on a per topic basis which means that each topic can have its own last will message.

This means that each topic can have its own last will message associated with it. The message is stored on the broker and sent to any subscribing client (to that topic) if the connection to the publisher fails.

The last will message is included in the connection request message.

2.1.4 MQTT Topics

MQTT topics are a form of addressing that allows MQTT clients to share information. MQTT Topics are structured in a hierarchy similar to folders and files in a file system using the forward slash (/)as a delimiter.

Topics names need to be:

- Case sensitive.
- UTF-8 strings.
- At least one character long.

Except for the $SYS topic there is no default or standard topic structure. There are no topics created on a broker by default, except for the $SYS topic. All topics are created by a subscribing or publishing client, and they are not permanent. A topic only exists if a client has subscribed to it, or a broker has a retained or last will messages stored for that topic.

The $SYS topic is a reserved topic and used by most MQTT brokers to publish information about the broker. They are read-only topics for the MQTT clients. There is no standard for this topic structure but typically contains information such as:

- Broker version.
- Broker timestamp.
- Broker uptime.
- Total clients.
- Disconnected clients.
- Messages stored.
- Messages received.

A client can subscribe to individual or multiple topics. When subscribing to multiple topics two wildcard characters can be used, they are:

- # - Multilevel wildcard.
- + - Single level wildcard.

Wildcards can only be used to denote a level or multi-levels. For example, consider the following topic hierarchy:

/

/house

/house/room1

/house/room2

/house/room1/top-light

/house/room1/side-light

/house/room2/top-light

/house/room2/side-light

If a client subscribes to /house/# that means it will receive messages from topics:

/house

/house/room1

/house/room1/top-light

/house/room1/side-light

/house/room2/top-light

/house/room2/side-light

However, if a client subscribes to /house/+/top-light it will receive messages from topics:

/house/room1/top-light

/house/room2/top-light

However, wildcards characters are only valid when subscribing to topics, you can **NOT** publish in more than one topic, in order to do that you need to repeat the same message in a different topic.

Topics are created dynamically, when someone subscribes to a topic or when someone publish to a topic with the retained message set to true.

Topics are destroyed when the last client subscribed disconnects and clean session is true, or when a client connects and sets clean session to true.

2.1.5 MQTT Client-Broker Connections

MQTT uses TCP/IP to connect to the broker, TCP is a connection orientated protocol with error correction and guarantees that packets are received in order. Most MQTT clients will connect to the broker and remain connected even if they are not sending data.

Connections are acknowledged by the broker using a Connection acknowledgement message. MQTT is a command-response protocol and each command is acknowledged, you cannot publish or subscribe unless you are connected. MQTT clients publish a keepalive message at regular intervals (usually 60 seconds) which tells the broker that the client is still connected.

A typical MQTT message flow is show in the following figure:

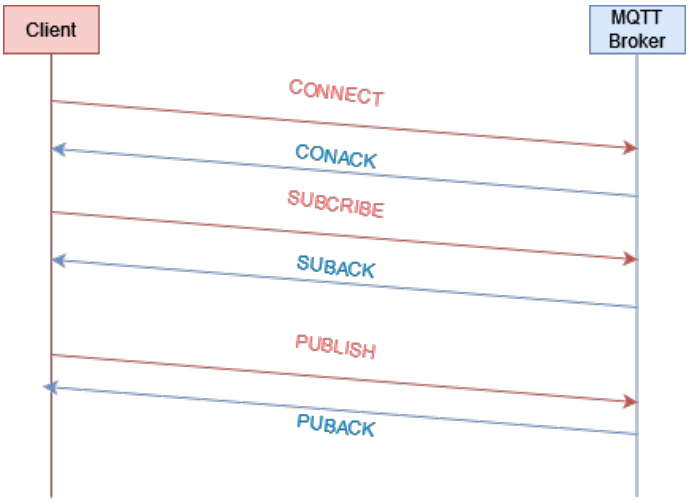

Figure 3: MQQT message flow

A client cannot publish a message to another client directly and does not know if any clients receive that message. A client can only publish messages to a single topic, and cannot publish to a group of topics. However a message can be received by a group of clients if they subscribe to the same topic.

MQTT supports 3 different levels of Quality of Service (QoS):

1. **QoS 0 - Default**: Does not guarantee message delivery.
2. **QoS 1**: Guarantees message delivery but can get duplicates.
3. **QoS 2**: Guarantees message delivery with no duplicates.

Messages with QoS 1 or QoS 2 are acknowledged by the broker which results in several messages being sent, they also have a message id that can be used to track them.

The MQTT message format is as follows:

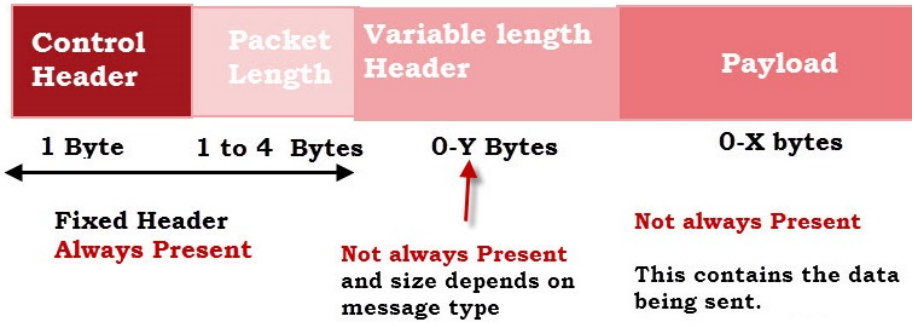

Figure 4: MQQT message format

The first byte is the control header, that includes the packet type and control flag like the message QoS and the retain flag. afterwards 1 to 4 bytes are used to declare the length of the packet. There is a variable length header that is optional and only applies to some messages and finally the payload, where the data is contained.

As a summary, some of the main characteristics of MQTT are:

- Clients do **NOT** have addresses and messages are not sent to clients.
- The messages are published to the broker on a topic.
- The MQTT broker is the one that filters these messages based on topic and delivers them to the subscribers.
- A client can receive these messages by subscribing to that topic on the same broker.

- There is **NO** direct connection between a publisher and a subscriber.
- All clients can publish (broadcast) and subscribe (receive).
- MQTT brokers do not normally store messages.

2.2 MQTT Broker Configuration

There are several implementations of MQTT brokers or server, however, in this practice class we are going to use Mosquitto. Mosquitto is an open source MQTT broker, available for several operative systems and highly used.

2.2.1 Mosquitto Installation

The installation depends of the operative systems, in our case, where we use a Debian based distribution, like Ubuntu, it can be installed with the following commands:

```
sudo apt-get update
sudo apt-get install mosquitto
```

Alternatively, to initialize the broker simply type the following command:

```
mosquitto
```

With this, the MQTT broker will be running in the default port, 1883. In order to publish a message we use *mosquitto_pub*, for example:

```
mosquitto_pub -t "my/topic" -m "Hello MQTT!"
```

In some cases, mosquitto does not install the client utilities, if the previous command does not work then type the following command and try again:

```
sudo apt install mosquitto-clients
```

Subscribing to a topic is similarly easy, the command is *mosquitto_sub*, to subscribe to the topic of the previous example:

```
mosquitto_sub -t "my/topic"
```

The MQTT broker configuration is displayed in the *mosquitto.conf* file, usually located in */etc/mosquitto/mosquitto.conf*. Further information about the mosquitto configuration file cane be found in `https://mosquitto.org/man/mosquitto-conf-5.html`.

Finally, to stop the Mosquitto broker type the command:

```
sudo systemctl stop mosquitto
```

3 Telegraf

Telegraf is a server-based agent for collecting and sending all metrics and events from databases, systems, and IoT sensors.

3.1 Introduction

Telegraf is a versatile and essential tool that empowers the collection of critical and stateful data from a wide array of devices and systems. It excels in gathering crucial information, such as pressure levels, temperature readings, and more. Telegraf achieves this through seamless integration with popular communication protocols like MQTT, ModBus, OPC-UA, and Kafka, making data collection highly efficient and accessible.

Beyond its proficiency in collecting device-specific data, Telegraf extends its capabilities to encompass a broader spectrum of metrics. It is proficient at retrieving metrics from various sources, including cloud platforms, containers, and orchestration services like GitHub, Kubernetes, CloudWatch, and Prometheus. This versatility ensures that Telegraf can adapt to the diverse technology stack of modern systems.

Telegraf doesn't stop at application and service metrics; it also excels at gathering system telemetry data. It can effortlessly retrieve information from sources such as iptables, Netstat, NGINX, and many more. This comprehensive approach provides a holistic view of your technology ecosystem, enabling effective monitoring and data-driven decision-making.

3.2 Installation and Configuration

Telegraf offers a wide variety of options for installing depending on your operative system and needs. As usual, it will be installed for Ubuntu and Debian distributions using the command that is displayed for $Ubuntu\&Debian$.

Once installed, it can be configured with the following command:

```
telegraf config > telegraf.conf
```

This will create a configuration file with default inputs and output plugins. You can specify where to store the configuration file, but it will usually be located in $/etc/telegraf/$.

Configuration files are very extensive spanning thousands of lines, each pertaining to specific parameters. From environmental variables to global tags, there are also plenty options to customize your configuration file.

This is also where your agent and plugins are configured, the default configuration will include every plugins but most will be commented out. Meaning they will not be in use. We are going to use a toy configuration, since this is just an approach to Telegraf and not a deep dive into its workings.

The agent configurations will be:

```
[agent]
  interval = "60s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  collection_jitter = "0s"
  flush_interval = "10s"
  flush_jitter = "0s"
  precision = ""
  logfile = ""
  logfile_rotation_interval = "1d"
  logfile_rotation_max_size = "100MB"
  hostname = "212.128.44.184:1883"
  omit_hostname = false
```

The inputs configuration will be:

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://212.128.44.184:1883"]
  topics = ["application/#"]
  username = "your_username"
  password = "your_password"
  data_format = "json"
```

This will attach to your MQTT broker, please swap the values with your own username, password and server direction.
The outputs configuration will be:

```
[[outputs.influxdb_v2]]
  urls = ["http://212.128.44.184:8086"]
  token = your-token"
  organization = "your-org"
  bucket = "your-bucket"
```

Likewise, the values for urls, token, organization and bucket should be filled with your own keys.

Further information about the configuration file can be found in `https://docs.influxdata.com/telegraf/v1/configuration/`.

Now, the Telegraf service needs ti be started and directed to the configuration file, that can be accomplished with the following command:

```
systemctl start telegraf
```

Or using the telegraf command:

```
telegraf --config path/to/your/config
```

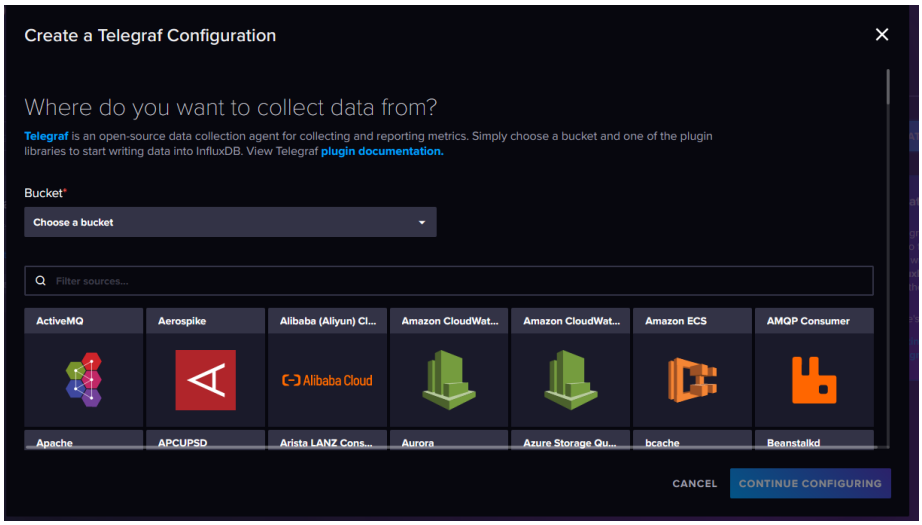

Figure 5: Bucket and Source selection

Antoher way to configure Telegraf, once installed, is to follow Influx web UI. Navigate to **Load Data** > **Telegraf** using the left bar. Then click on **Create Configuration**. You will be prompted to select a bucket and a source, see Figure 5.

Once you have selected your bucket and **MQTT Consumer** as source, you will be presented with this menu, Figure 6:

Where you can customize the configuration for your installation of Influx and your MQTT parameters. With that done, follow the set up instructions to get Telegraf running, see Figure 7.

Once completed, if you publish to a topic you have indicated in your Telegraf configuration, **mqtt_consumer** should appear in the **Data Explorer** section, where you can visualize the data you have just published.

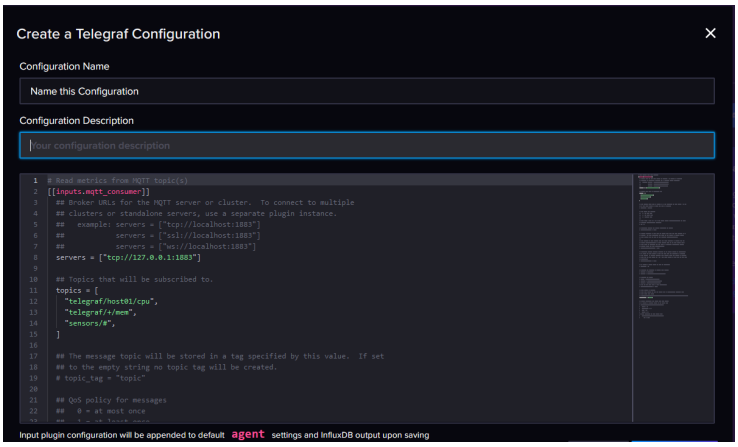

Figure 6: Telegraf configuration

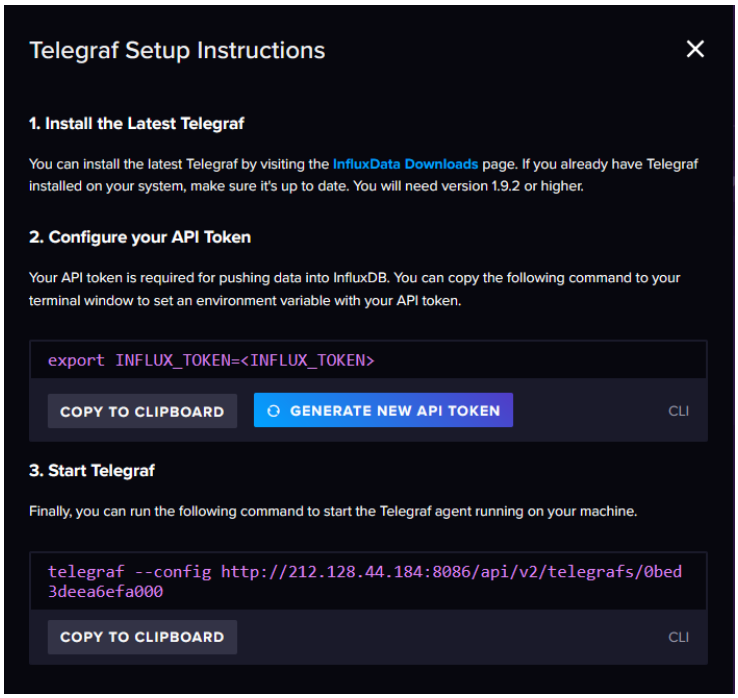

Figure 7: Set up instructions

4 Exercises

4.1 Influx

1 **Make a fresh installation of Influx in your own virtual machine.** Attach screenshots of every step taken to complete the installation.

2 **Access the influxDB UI and create a database, with at least one bucket.** Attach screenshots of the created database and a small explanation of the scheme you have created for your data.

3 **Write data into your database using line protocol**. Attach screenshot of the command used and your database populated.

4 **Query your own data. Go to Data Explorer in the influxDB UI and use both the Query Builder and the Script Editor to query datat.** Write the queries and attach screenshots of the results.

4.2 MQTT

1 **Make a fresh installation of Mosquitto in your own virtual machine.** Attach screenshots of every step taken to complete the installation.

2 **Use Mosquitto to create topics.** Write a small explanation about your topic scheme, remember topics are created dynamically.

3 **Use Mosquitto to subscribe to your topics. Experiment with wildcards!** Attach screenshots of the commands you use. Tip: use the command screen to organize your MQTT clients or several terminal windows.

4 **Now that we have topics and subscribers, publish some data to your topics and check if your subscribers get the messages!** Write the commands and attach screenshots of the results.

4.3 Telegraf

1 **Make a fresh installation of Telegraf in your own virtual machine.** Attach screenshost of every step taken to complete the installation.

2 **Make your own configuration file.** Create a default configuration and modify it or create your own from scratch.

3 **Check in the influxDB UI that Telegraf is running.** Make sure your configuration file shows up in Load Data and mqtt_consumer appears in the Query Builder.