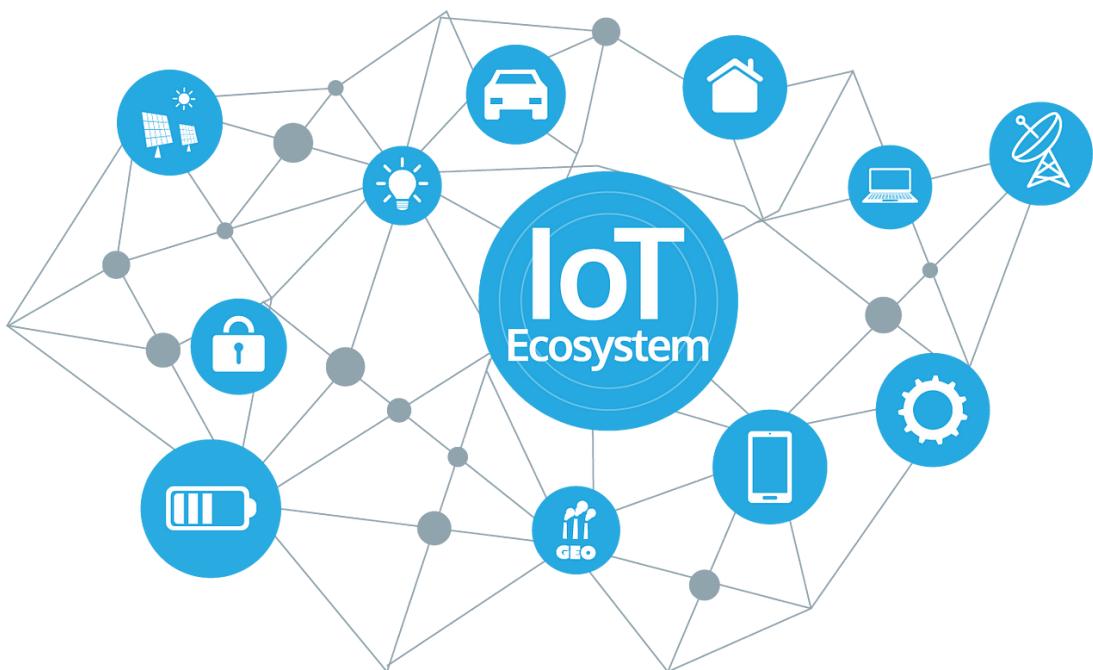


# Despliegue de arquitectura de red IoT e integración con bases de datos de serie de tiempo.

Un enfoque para la persistencia de Datos en Redes IoT



**Alumno:** Quintana Guillermo Germán.

**Profesor:** Lic. Pautsch, Jesús Germán Andrés.

**Cátedra:** Bases de datos.



# Introducción:

En la actualidad, el Internet de las cosas (IoT) ha experimentado un auge significativo, impulsando la necesidad de soluciones eficientes para recopilar, almacenar y analizar grandes volúmenes de datos generados por sensores distribuidos en diferentes entornos. En este trabajo de investigación, se propone la implementación de un sensor IoT que enviará sus mediciones a través de tecnologías de bases de datos optimizadas para aplicaciones de IoT. El flujo de datos se gestionará mediante una instancia de Telegraph que publicará las mediciones de nuestro sensor en un broker MQTT utilizando EMQX. Posteriormente, otra instancia de Telegraph suscrita al broker escribirá los datos en una base de datos de serie de tiempo InfluxDB. Por último, los datos se visualizarán utilizando Grafana, y todo el sistema se desplegará y orquestará en un entorno Docker Compose.

# Bases de Datos Optimizadas para Aplicaciones de IoT:

## ¿Qué es una TSDB?

En el contexto de las aplicaciones de IoT, es fundamental contar con bases de datos optimizadas para manejar grandes volúmenes de datos en tiempo real y facilitar su análisis. Una base de datos de serie de tiempo (TSDB, por sus siglas en inglés) es una base de datos optimizada para datos con marca de tiempo o series de tiempo. Los datos de series de tiempo son simplemente mediciones o eventos que se rastrean, monitorean, muestrean y agregan a lo largo del tiempo. Esto puede incluir métricas de servidores, monitoreo del rendimiento de aplicaciones, datos de redes, datos de sensores, eventos, clics, transacciones en un mercado y muchos otros tipos de datos analíticos.

Una base de datos de serie de tiempo está diseñada específicamente para manejar métricas, eventos o mediciones con marca de tiempo. Una TSDB está optimizada para medir el cambio a lo largo del tiempo. Las propiedades que hacen que los datos de series de tiempo sean muy diferentes de otras cargas de trabajo de datos son la gestión del ciclo de vida de los datos, la summarización y el escaneo de rangos amplios de muchos registros.

## TSDBs en la actualidad.

Las bases de datos de serie de tiempo no son nuevas, pero las bases de datos de serie de tiempo de primera generación se enfocaban principalmente en datos financieros, la volatilidad del comercio de acciones y sistemas construidos para resolver operaciones de trading. Sin embargo, los datos financieros son apenas una de las muchas aplicaciones de los datos de series de tiempo en la actualidad; de hecho, es solo una entre numerosas aplicaciones en diversas industrias. Las condiciones fundamentales de la informática han cambiado drásticamente en la última década. Todo se ha vuelto modular. Los mainframes monolíticos han desaparecido, reemplazados por servidores sin servidor, microservidores y contenedores.

Hoy en día, todo lo que puede ser un componente es un componente. Además, estamos presenciando la instrumentación de todas las superficies disponibles en el mundo material: calles, automóviles, fábricas, redes eléctricas, casquitos polares, satélites, ropa, teléfonos, microondas, recipientes de leche, planetas, cuerpos humanos. Todo tiene, o tendrá, un sensor. Por lo tanto, ahora todo dentro y fuera de estas empresas s emite un flujo incesante de métricas, eventos o datos de series de tiempo.

Esto significa que las plataformas subyacentes deben evolucionar para admitir estas nuevas cargas de trabajo: más puntos de datos, más fuentes de datos, más monitoreo, más controles. Lo que estamos presenciando y lo que demandan los tiempos es un cambio paradigmático en cómo abordamos nuestra infraestructura de datos y cómo construimos, monitoreamos, controlamos y administramos sistemas. Lo que necesitamos es una base de datos de serie de tiempo con rendimiento, escalabilidad y construcción específica para su propósito.

## ¿Qué distingue la carga de trabajo de las series de tiempo?

Las bases de datos de series de tiempo tienen propiedades de diseño arquitectónico clave que las hacen muy diferentes de otras bases de datos. Estas incluyen almacenamiento y compresión de datos con marca de tiempo, gestión del ciclo de vida de los datos, summarización de datos, capacidad para manejar grandes escaneos dependientes de series de tiempo de muchos registros y consultas conscientes de las series de tiempo.

Por ejemplo, con una base de datos de serie de tiempo, es común solicitar un resumen de datos en un largo período de tiempo. Esto requiere recorrer un rango de puntos de datos para realizar algún cálculo, como un aumento porcentual de una métrica este mes en comparación con el mismo período en los últimos seis meses, resumido por mes. Este tipo de carga de trabajo es muy difícil de optimizar con una base de datos distribuida de clave-valor. Las TSDB están optimizadas específicamente para este caso de uso, brindando tiempos de consulta de milisegundos sobre meses de datos. Otro ejemplo: con las bases de datos de series de tiempo, es común mantener datos de alta precisión durante un corto período de tiempo. Estos datos se agregan y muestran a datos de tendencias a largo plazo. Esto significa que cada punto de datos que ingresa a la base de datos debe eliminarse una vez que haya pasado su período de tiempo. Esta gestión del ciclo de vida de los datos es difícil de implementar para los desarrolladores de aplicaciones sobre bases de datos regulares. Deben idear esquemas para eliminar de manera económica grandes conjuntos de datos y resumir constantemente esos datos a gran escala. Con una base de datos de serie de tiempo, esta funcionalidad se proporciona de manera nativa.

Como se mencionó con anterioridad otra característica distintiva de las bases de datos de serie de tiempo es la gestión del ciclo de vida de los datos. Es común mantener datos de alta precisión solo durante un corto período de tiempo. Estos datos se agregan y resumen en datos de tendencias a largo plazo. Esto significa que cada punto de datos tiene una vida útil después de la cual debe eliminarse. La gestión eficiente de este ciclo de vida de los datos es difícil de implementar en bases de datos regulares y requiere soluciones especializadas.



## Decisión de utilizar InfluxDB

En este trabajo, se utilizará InfluxDB como la base de datos principal, ya que está diseñada específicamente para aplicaciones de serie de tiempo y ofrece características como el almacenamiento eficiente de datos de alta velocidad y una consulta ágil basada en marcas temporales. Estas características hacen que InfluxDB sea una opción idónea para el almacenamiento y análisis de datos provenientes de sensores ya que fue diseñada desde cero como una base de datos de serie de tiempo específica, en lugar de ser adaptada para este propósito. El tiempo fue incorporado en su diseño desde el principio. InfluxDB es parte de una plataforma completa que incluye la recolección, almacenamiento, monitoreo, visualización y alertas de datos de serie de tiempo. Va más allá de ser solo una base de datos de serie de tiempo.

Las etiquetas y el nombre de la medición se almacenan en un índice invertido que permite búsquedas rápidas de series de tiempo específicas, también utiliza una compresión variable según la precisión requerida por el usuario, lo que permite un equilibrio entre el rendimiento y el almacenamiento.

Todas estas cuestiones se suman al excelente soporte por parte de servicios de terceros lo que facilita su implementación, configuración, e integración con las demás tecnologías necesarias para montar una infraestructura robusta.

# Integración de Tecnologías:

**Nodos IoT:** Se implementarán 3 nodos IoT emulados capaz de generar y enviar mediciones en tiempo real. Este "sensor" estará conectado a través de la red para comunicarse con el broker.

**EMQX:** Se empleará como el broker MQTT para facilitar la comunicación bidireccional entre el sensor IoT y la instancia de Telegraph encargada de escribir los datos en InfluxDB. El broker MQTT garantiza la entrega confiable de mensajes en un entorno de IoT distribuido. Una de las ventajas de su utilización es que múltiples dispositivos pueden publicar o leer de los tópicos permitiendo una comunicación eficiente utilizando un patrón de diseño publicador / suscriptor en nuestra arquitectura.

<https://www.emqx.io/>

**Telegraf:** Telegraf actuará como el intermediario para la transferencia de datos entre el broker y la base de datos.

<https://www.influxdata.com/time-series-platform/telegraf/>

**InfluxDB:** La base de datos InfluxDB se utilizará para almacenar y gestionar las mediciones provenientes del sensor IoT. Gracias a su estructura optimizada para datos de serie de tiempo, InfluxDB permite un almacenamiento eficiente y consultas ágiles sobre los datos capturados por el sensor.

<https://www.influxdata.com/>

**Grafana:** Se emplea Grafana como herramienta de visualización para representar los datos almacenados en InfluxDB. Grafana permite crear paneles de control personalizados, gráficos interactivos y alertas basadas en los datos provenientes del sensor IoT.

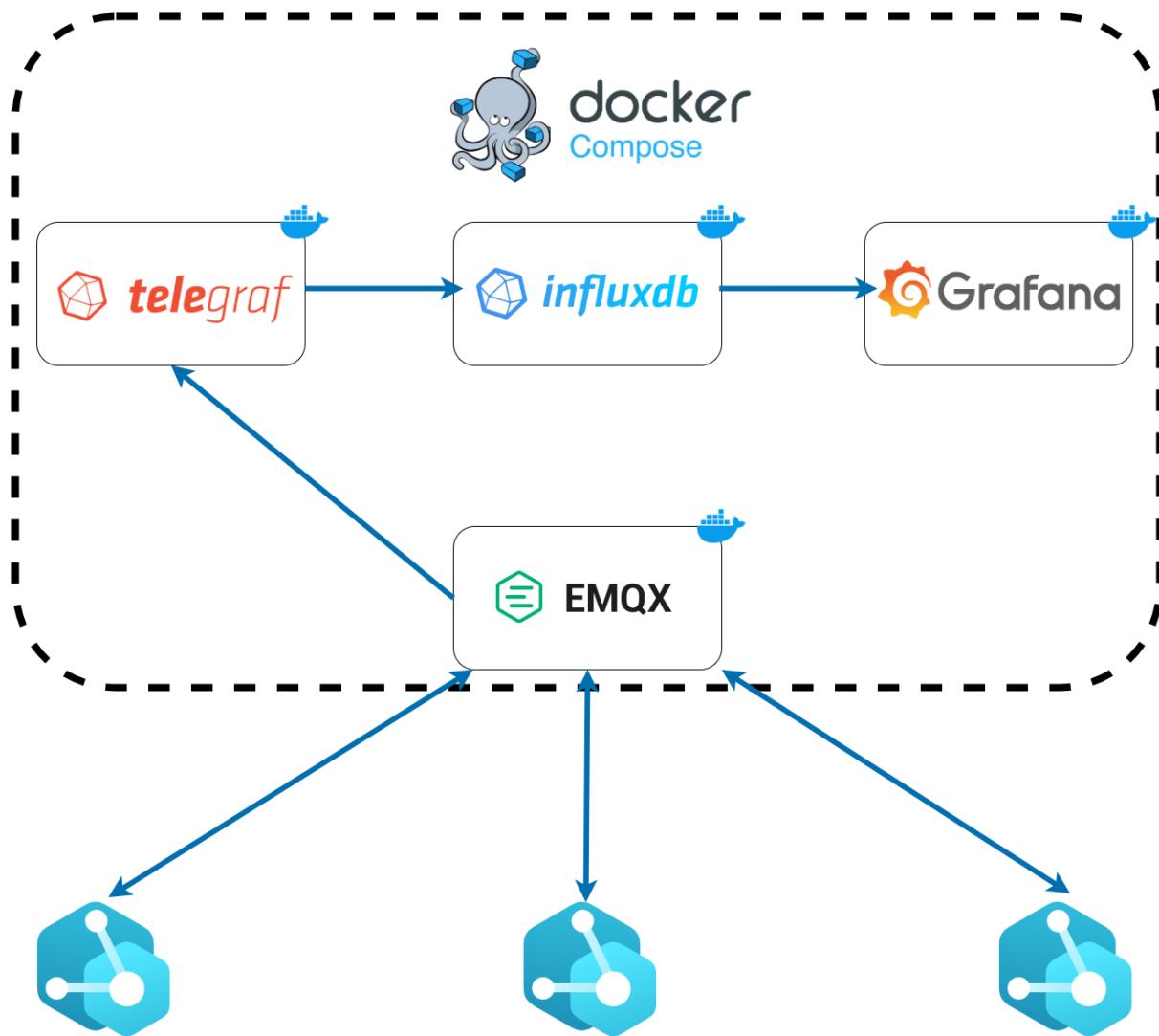
<https://grafana.com/>

**Docker Compose:** El entorno de Docker Compose se utilizará para orquestar y desplegar todas las tecnologías y componentes del sistema propuesto como servicios integrados dentro de una misma imagen. La utilización de contenedores facilita la replicación y escalabilidad del sistema en entornos de producción.

<https://docs.docker.com/compose/>

# Arquitectura

La arquitectura propuesta establece una infraestructura IoT con el broker MQTT EMQX como punto central de comunicación, una instancia de Telegraf como consumidor de mensajes, una base de datos InfluxDB para almacenar los datos procesados junto a una instancia de Grafana para visualizar los datos de manera efectiva. Todo esto se despliega y se gestiona en un entorno Docker Compose.



## Docker compose

En esta sección, se presentará una introducción a la configuración de la infraestructura de IoT utilizando Docker Compose. Se explicará que el archivo Docker Compose utiliza una sintaxis YAML para definir los servicios y componentes necesarios para la infraestructura. Además, se destacará que el archivo proporciona una forma conveniente y reproducible de gestionar y desplegar los diferentes servicios.

Docker Compose es una herramienta que permite definir y gestionar aplicaciones Docker multi-contenedor. Utiliza un archivo de configuración llamado "docker-compose.yml" para describir los servicios, redes y volúmenes necesarios para ejecutar una aplicación.

La sintaxis utilizada en el archivo Docker Compose es YAML (acrónimo recursivo para "YAML Ain't Markup Language"). YAML es un formato legible por humanos y fácil de entender que se utiliza para estructurar datos. Proporciona una manera clara y concisa de definir la configuración y las relaciones entre los componentes de la infraestructura.

En el archivo Docker Compose presentado, se definen varios servicios que componen la infraestructura de IoT. Cada servicio está definido como un bloque en el archivo, y se especifican los detalles como la imagen de Docker a utilizar, los puertos que se deben mapear, los volúmenes a montar y las variables de entorno a configurar.

Además de los servicios, el archivo Docker Compose también puede incluir la definición de redes y volúmenes, que permiten la comunicación entre los contenedores y el almacenamiento persistente de datos.

Docker Compose proporciona varias ventajas al configurar una infraestructura de IoT:

**Reproducibilidad:** Al definir la infraestructura en un archivo YAML, se asegura que todos los componentes se desplieguen de manera consistente en diferentes entornos, lo que facilita la replicación y la gestión del sistema.

**Gestión simplificada:** Docker Compose permite iniciar, detener y escalar los servicios de la infraestructura con un solo comando, lo que simplifica enormemente la gestión de los diferentes componentes.

**Comunicación entre servicios:** Docker Compose facilita la creación de redes virtuales para que los diferentes servicios puedan comunicarse entre sí de forma segura y eficiente.

**Configuración flexible:** La sintaxis YAML de Docker Compose permite configurar fácilmente los diferentes componentes, incluyendo la definición de variables de entorno, volúmenes y puertos.

**La versión de Docker Compose utilizada es la "3.9".**

version: "3.9"

**El archivo define los servicios a desplegar en la infraestructura.**

services:

**El primer servicio es "emqx", que es un broker MQTT basado en EMQX.**

1. Se especifica que el usuario del contenedor será "root".
2. Se utiliza la imagen "emqx:4.4.3".
3. Se mapean los siguientes puertos del contenedor al host: 18083, 1883 y 8883.
4. Se montan los directorios locales "./data/emqx/data" y "./data/emqx/log" dentro del contenedor en los directorios "/opt/emqx/data" y "/opt/emqx/log", respectivamente.
5. Se configuran las variables de entorno "EMQX\_DASHBOARD\_\_DEFAULT\_USER\_\_LOGIN" y "EMQX\_DASHBOARD\_\_DEFAULT\_USER\_\_PASSWORD" con los valores "admin" y "admin", respectivamente.

```
# Broker MQTT EMQX
emqx:
  user: root
  image: "emqx:4.4.3" # Utiliza la imagen de EMQX versión 4.4.3
  ports:
    - "18083:18083" # Mapea el puerto 18083 del contenedor al puerto 18083 del host
    - "1883:1883" # Mapea el puerto 1883 del contenedor al puerto 1883 del host
    - "8883:8883" # Mapea el puerto 8883 del contenedor al puerto 8883 del host
  volumes:
    - ./data/emqx/data:/opt/emqx/data # Monta el directorio local './data/emqx/data' en '/opt/emqx/data' dentro del contenedor
    - ./data/emqx/log:/opt/emqx/log # Monta el directorio local './data/emqx/log' en '/opt/emqx/log' dentro del contenedor
  environment:
    - EMQX_DASHBOARD__DEFAULT_USER__LOGIN=admin # Configura la variable de entorno EMQX_DASHBOARD__DEFAULT_USER__LOGIN con el valor 'admin'
    - EMQX_DASHBOARD__DEFAULT_USER__PASSWORD=admin # Configura la variable de entorno EMQX_DASHBOARD__DEFAULT_USER__PASSWORD con el valor 'admin'
```

**El segundo servicio es "influxdb".**

1. Se utiliza la imagen "influxdb:2.2.0-alpine".
2. Se mapea el puerto 8086 del contenedor al puerto 8086 del host.
3. Se montan los directorios locales "./data/influxdb/data" y "./data/influxdb/config" dentro del contenedor en los directorios "/var/lib/influxdb2" y "/etc/influxdb2", respectivamente.
4. Se configuran varias variables de entorno relacionadas con la inicialización de InfluxDB, como el modo de inicialización, el nombre de usuario y contraseña, la organización, el bucket, la retención de datos y el token de autenticación del administrador.

```
# Base de datos InfluxDB

influxdb:

image: "influxdb:2.2.0-alpine" # Utiliza la imagen de InfluxDB versión
2.2.0-alpine

ports:
- "8086:8086" # Mapea el puerto 8086 del contenedor al puerto 8086 del host

volumes:
- ./data/influxdb/data:/var/lib/influxdb2 # Monta el directorio local
'./data/influxdb/data' en '/var/lib/influxdb2' dentro del contenedor
- ./data/influxdb/config:/etc/influxdb2 # Monta el directorio local
'./data/influxdb/config' en '/etc/influxdb2' dentro del contenedor

environment:
- DOCKER_INFLUXDB_INIT_MODE=setup # Configura la variable de entorno
DOCKER_INFLUXDB_INIT_MODE con el valor 'setup'
- DOCKER_INFLUXDB_INIT_USERNAME=administrator # Configura la variable de
entorno DOCKER_INFLUXDB_INIT_USERNAME con el valor 'admin'
- DOCKER_INFLUXDB_INIT_PASSWORD=Administrator1 # Configura la variable de
entorno DOCKER_INFLUXDB_INIT_PASSWORD con el valor 'admin'
- DOCKER_INFLUXDB_INIT_ORG=GuillermoQuintana # Configura la variable de
entorno DOCKER_INFLUXDB_INIT_ORG con el valor 'GuillermoQuintana'
- DOCKER_INFLUXDB_INIT_BUCKET=iotdata # Configura la variable de entorno
DOCKER_INFLUXDB_INIT_BUCKET con el valor 'iotdata'
- DOCKER_INFLUXDB_INIT_RETENTION=1w # Configura la variable de entorno
DOCKER_INFLUXDB_INIT_RETENTION con el valor '1w'
- DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=GuillermoQuintana-iotdata-auth-token # Configura la variable de entorno DOCKER_INFLUXDB_INIT_ADMIN_TOKEN con el valor 'GuillermoQuintana-iotdata-auth-token'
```

**El tercer servicio es "telegraf", que es el agente de recolección y envío de datos.**

1. Se utiliza la imagen "telegraf:1.22.4".
2. Se configura una política de reinicio en caso de falla, con un retraso de reinicio de 10 segundos y un máximo de 20 intentos.
3. Se monta el archivo de configuración local "./data/telegrafoutput/telegraf.conf" dentro del contenedor en el directorio "/etc/telegraf/telegraf.conf".

```
# Telegraf
telegrafoutput:
image: "telegraf:1.22.4" # Utiliza la imagen de Telegraf versión 1.22.4
deploy:
restart_policy:
condition: on-failure
delay: 10s
max_attempts: 20
volumes:
- ./data/telegrafoutput/telegraf.conf:/etc/telegraf/telegraf.conf # Monta el
archivo de configuración local './data/telegrafoutput/telegraf.conf' en
'/etc/telegraf/telegraf.conf' dentro del contenedor
```

**El cuarto servicio es "grafana", que es la plataforma de visualización de datos.**

1. Se especifica que el usuario del contenedor será "root".
2. Se utiliza la imagen "grafana/grafana:8.5.3".
3. Se mapea el puerto 3000 del contenedor al puerto 3000 del host.
4. Se configura la variable de entorno "GF\_SECURITY\_ADMIN\_PASSWORD\_\_FILE" con el valor "/run/secrets/admin\_password".
5. Se monta el directorio local "./data/grafana" dentro del contenedor en el directorio "/var/lib/grafana".
6. Se utiliza un secreto llamado "grafana\_admin\_password" para configurar la contraseña del administrador de Grafana. El secreto se define en la sección "secrets" más abajo y se especifica el archivo que contiene la contraseña en "./secrets/grafana\_admin\_password".

```
# Grafana
grafana:
  user: root
  image: "grafana/grafana:8.5.3" # Utiliza la imagen de Grafana versión 8.5.3
  ports:
    - "3000:3000" # Mapea el puerto 3000 del contenedor al puerto 3000 del host
  environment:
    - GF_SECURITY_ADMIN_PASSWORD__FILE=/run/secrets/admin_password # Configura la
      variable de entorno GF_SECURITY_ADMIN_PASSWORD__FILE con el valor
      '/run/secrets/admin_password'
  volumes:
    - ./data/grafana:/var/lib/grafana # Monta el directorio local
      './data/grafana' en '/var/lib/grafana' dentro del contenedor
  secrets:
    - source: grafana_admin_password # Utiliza el secreto
      'grafana_admin_password' definido más abajo
      target: /run/secrets/admin_password

  secrets:
    grafana_admin_password:
      file: ./secrets/grafana_admin_password # Define el secreto
      'grafana_admin_password' utilizando el archivo
      './secrets/grafana_admin_password'
```

## Nodos IoT emulados

En este proyecto, se implementan dos nodos en python para simplicidad: El nodo del sensor (publicador) y el nodo del cliente (suscriptor). Cada uno desempeña un papel crucial en la medición y transmisión de la temperatura a través del protocolo MQTT.

### Nodo sensor

El nodo del sensor es responsable de generar mediciones aleatorias de temperatura y publicarlas en un tópico MQTT. Utiliza una función para generar una temperatura aleatoria y actualiza un diccionario de datos del sensor con la nueva medición. Luego, la medición se publica en formato JSON en el tópico "telegraf/telegrafinput/Temperature". El nodo del sensor también muestra las mediciones generadas en la consola para fines de visualización.

```
import time
import json
import random
import paho.mqtt.client as mqtt

def generate_random_temperature(sensor_data):
    """
    Función que genera una temperatura aleatoria y actualiza el diccionario
    sensor_data con la nueva medición.
    """
    sensor_data["timestamp"] = time.time()
    # Se agrega la marca de tiempo actual al diccionario sensor_data
    # Se obtiene la temperatura actual del diccionario
    current_temp = sensor_data["temperature"]

    # Se genera una diferencia aleatoria entre 0 y 0.5
    difference = random.uniform(0, 5)

    # Se decide aleatoriamente si se suma o resta la diferencia a la temperatura
    # actual
    add_or_subtract = random.randint(0, 1)

    if add_or_subtract and current_temp < 100: # Si add_or_subtract es 1 y la
        # temperatura actual es menor a 35
        # Se suma la diferencia a la temperatura actual
        sensor_data["temperature"] += difference
    elif current_temp > 25: # Si la temperatura actual es mayor a 10
```

```
# Se resta la diferencia a la temperatura actual
sensor_data["temperature"] -= difference

def on_connect(client, userdata, flags, rc):
    print("Conectado al Broker MQTT")

def main():
    """
    Función principal que genera mediciones aleatorias de temperatura y las
    publica en un tópico MQTT.
    """

    # Configuración del cliente MQTT
    client = mqtt.Client()
    client.on_connect = on_connect
    client.connect("localhost", 1883, 60)

    # Se crea un diccionario con los datos del sensor
    sensor_data = {
        "device_id": "e2e78334",
        "client_id": "c03d5155",
        "sensor_type": "Temperature",
        "temperature": 25,
        "timestamp": time.time()
    }

    while True:
        # Genera una nueva temperatura aleatoria
        generate_random_temperature(sensor_data)

        # Imprime la medición en la consola
        print("Medición generada: " + str(sensor_data))

        # Se publica la medición en formato JSON en el tópico MQTT
        client.publish("telegraf/telegrafinput/Temperature",
                      json.dumps(sensor_data))

        # Se imprime la medición en la consola
        print("Medición enviada: " + str(sensor_data))

        # Espera 0.5 segundos antes de generar la siguiente medición
        time.sleep(0.5)
```

## Nodo suscriptor

El nodo del cliente MQTT se conecta al broker MQTT y se suscribe al tópico "telegraf/telegrafinput/#" para recibir los mensajes publicados. Cuando se recibe un mensaje en el tópico, la función de manejo de mensajes se activa. El nodo del cliente MQTT verifica si el mensaje está relacionado con la temperatura y extrae la temperatura actual del mensaje en formato JSON. Luego, muestra la temperatura actual en la consola y verifica si excede los 60°C. Si la temperatura es superior a este umbral, se muestra un mensaje de alerta junto con la temperatura.

```
import paho.mqtt.client as mqtt
import json

def on_connect(client, userdata, flags, rc):
    print("Conectado al Broker MQTT")
    client.subscribe("telegraf/telegrafinput/#")

def on_message(client, userdata, msg):
    # Limpiar la pantalla
    print("\033c")
    # Obtener la temperatura actual del mensaje recibido
    if "Temperature" in msg.topic:
        payload = json.loads(msg.payload.decode("utf-8"))
        temperatura = payload["temperature"]

    # Redondear la temperatura a dos decimales
    temperatura = round(temperatura, 2)

    # Mostrar la temperatura actual
    print("Temperatura actual: {:.2f}".format(temperatura))

    # Verificar si la temperatura excede los 60°C
    if temperatura > 60:
        print(
            ";Alerta! La temperatura ha excedido los 60°C: {:.2f}".format(
                temperatura)
        )
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        client.connect("localhost", 1883, 60)
        client.loop_forever()
```

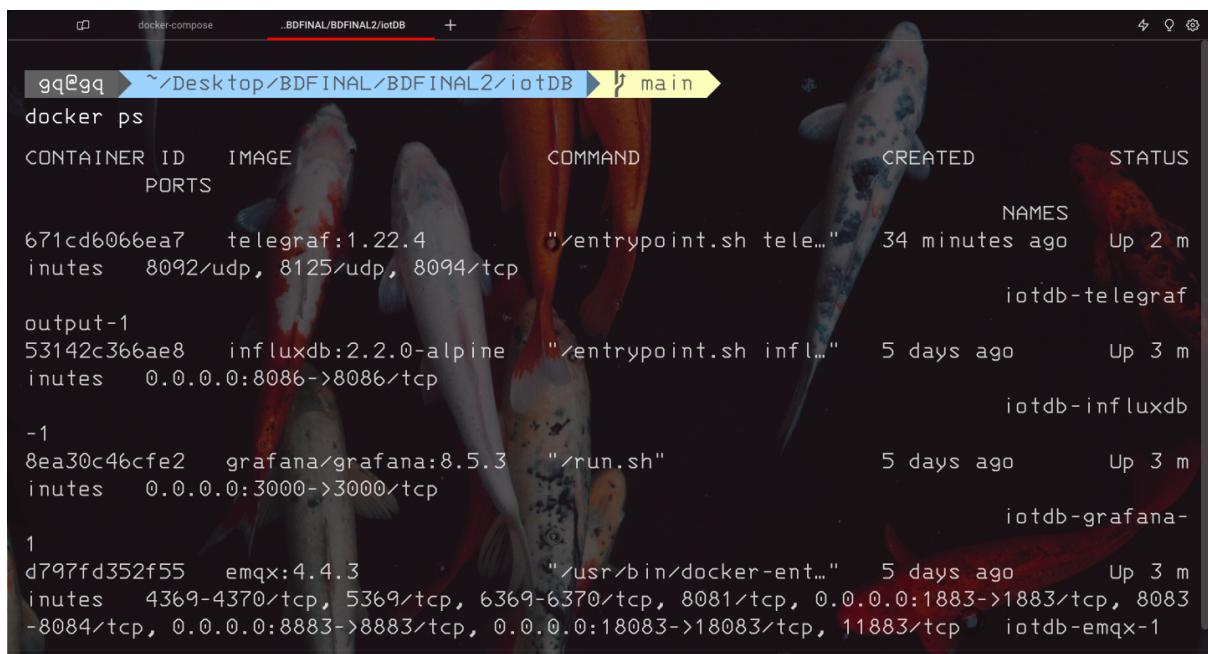
## Iniciando los servicios

Una vez configurado todo el entorno de trabajo resta iniciar y acceder a los servicios que se definieron el docker compose para ello ejecutaremos docker-compose up



```
gq@gq: ~/Desktop/BDFINAL/BDFINAL2/iotDB docker-compose up
[+] Running 4/0
  ✓ Container iotdb-telegrafoutput-1 Created
    0.0s
  ✓ Container iotdb-grafana-1 Created
    0.0s
  ✓ Container iotdb-emqx-1 Created
    0.0s
  ✓ Container iotdb-influxdb-1 Created
    0.0s
Attaching to iotdb-emqx-1, iotdb-grafana-1, iotdb-influxdb-1, iotdb-telegrafoutput-1
```

Una vez iniciado el contenedor podemos corroborar que están corriendo todas las imágenes con el comando docker ps

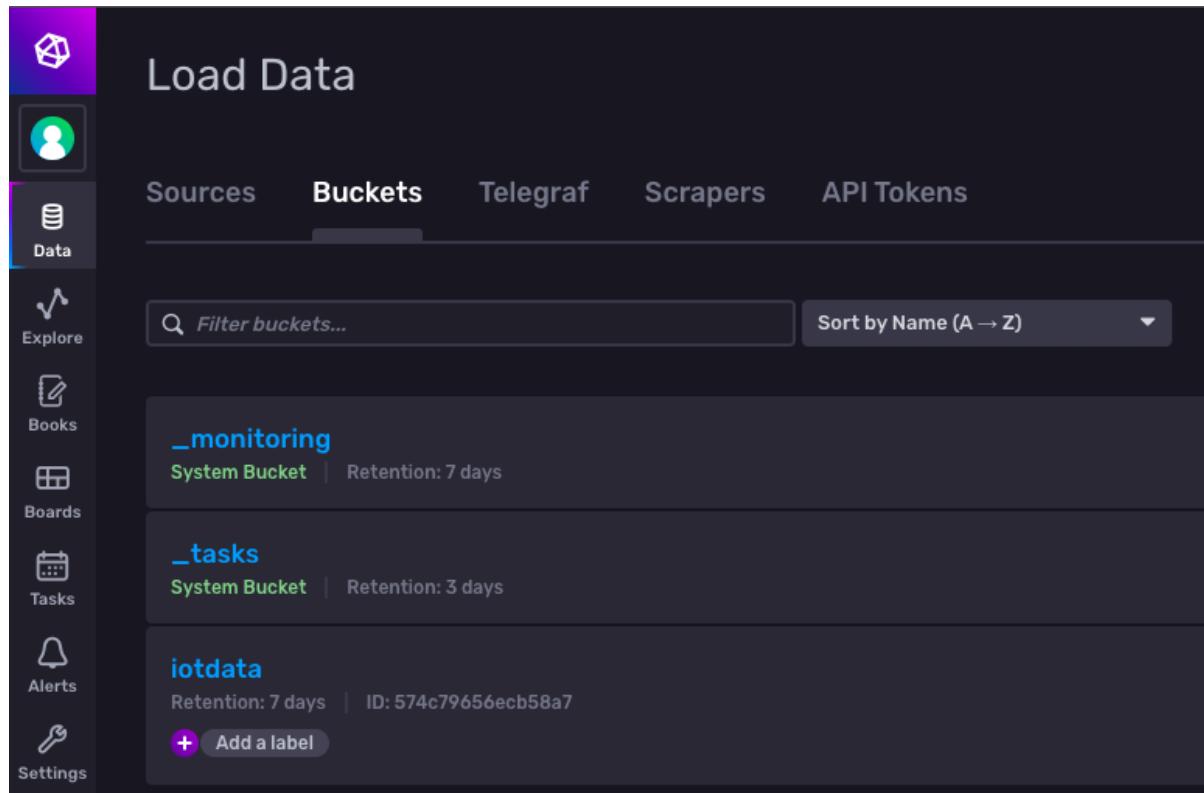


CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
PORTS					
671cd6066ea7	telegraf:1.22.4	"./entrypoint.sh tele..."	34 minutes ago	Up 2 m	iotdb-telegraf
53142c366ae8	influxdb:2.2.0-alpine	"./entrypoint.sh infl..."	5 days ago	Up 3 m	iotdb-influxdb
8ea30c46cf2	grafana/grafana:8.5.3	"./run.sh"	5 days ago	Up 3 m	iotdb-grafana-1
d797fd352f55	emqx:4.4.3	"./usr/bin/docker-ent..."	5 days ago	Up 3 m	iotdb-emqx-1
		4369-4370/tcp, 5369/tcp, 6369-6370/tcp, 8081/tcp, 0.0.0.0:1883->1883/tcp, 8083-8084/tcp, 0.0.0.0:8883->8883/tcp, 0.0.0.0:18083->18083/tcp, 11883/tcp			

Para alimentar de datos a nuestro broker y por consiguiente a nuestra base de datos iniciaremos el nodo publicador.

```
qq@qq ~$ cd nodo_publicador/
qq@qq ~$ python3 nodo_publicador.py
Medición generada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 29.128524808220703, 'timestamp': 1689284357.1276958}
Medición enviada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 29.128524808220703, 'timestamp': 1689284357.1276958}
Medición generada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 25.87073629654365, 'timestamp': 1689284357.629256}
Medición enviada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 25.87073629654365, 'timestamp': 1689284357.629256}
Medición generada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 28.376509017181238, 'timestamp': 1689284358.130191}
Medición enviada: {'device_id': 'e2e78334', 'client_id': 'c03d5155', 'sensor_type': 'Temperatura', 'temperature': 28.376509017181238, 'timestamp': 1689284358.130191}
```

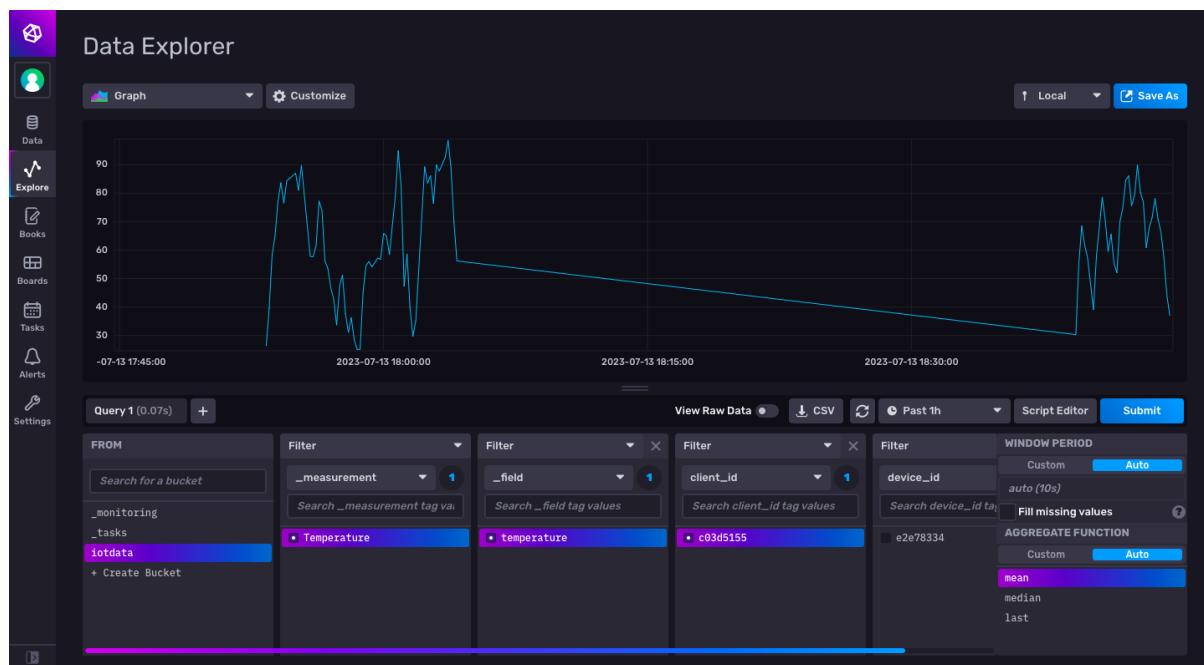
Luego de esto podemos acceder al panel de influxDB y ver nuestro bucket **iotdata** preconfigurado en el docker compose. Y al hacer click en el bucket y corroborar que el canal de “Temperatura” está recibiendo datos y podemos visualizar los mismos aunque para la tarea de visualización y análisis de datos utilizaremos grafana.



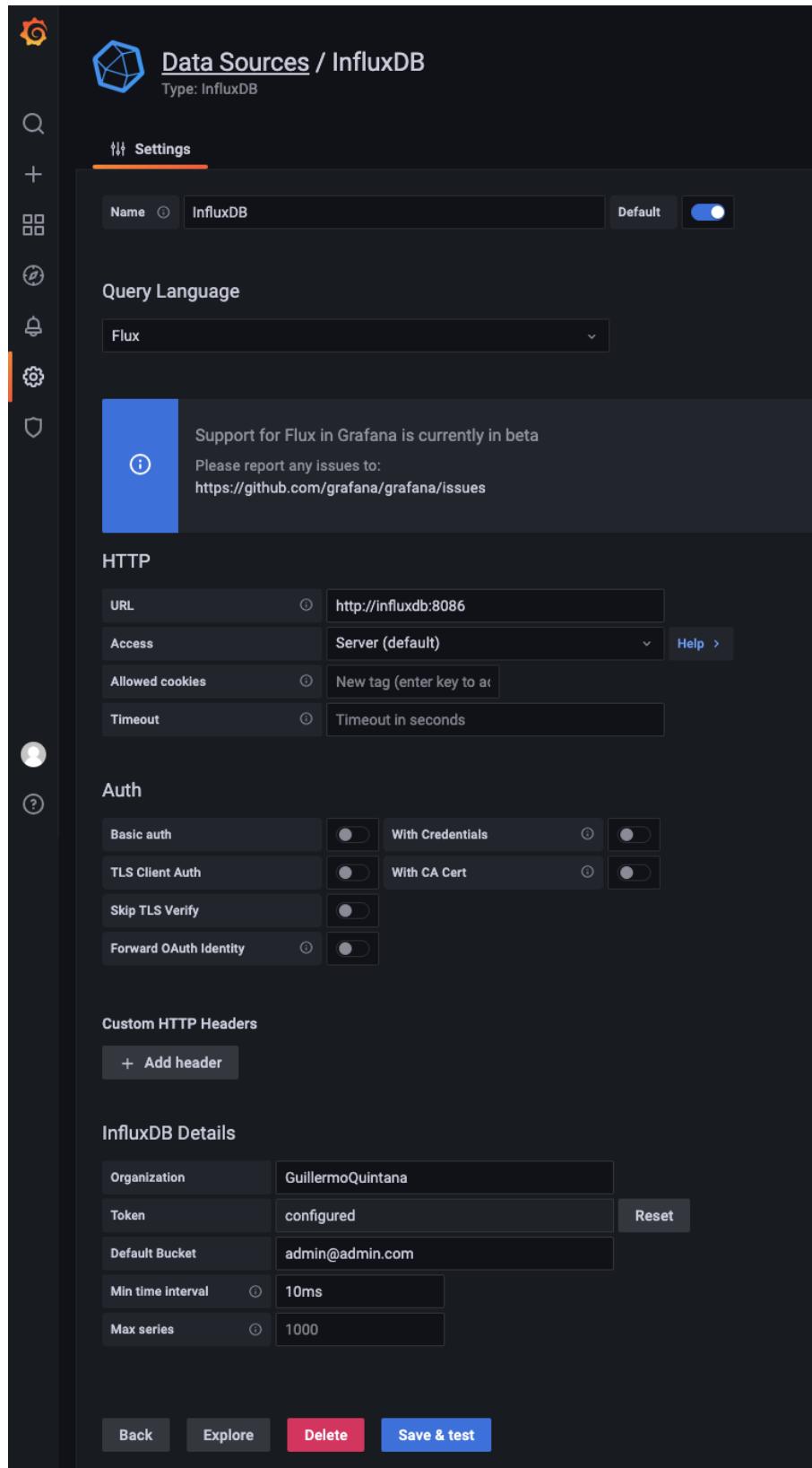
The screenshot shows the InfluxDB interface with the 'Buckets' tab selected. On the left sidebar, there are icons for Load Data, Sources, Buckets, Telegraf, Scrapers, API Tokens, Explore, Books, Boards, Tasks, Alerts, and Settings. The 'Data' icon is highlighted. The main area displays three buckets:

- \_monitoring**: System Bucket | Retention: 7 days
- \_tasks**: System Bucket | Retention: 3 days
- iotdata**: Retention: 7 days | ID: 574c79656ecb58a7

A button labeled '+ Add a label' is visible next to the 'iotdata' bucket.



Ahora que sabemos que nuestra base de datos se encuentra recibiendo y persistiendo datos podemos configurar graffana para que consuma esos datos y pueda procesarlos para transformarlos en información utilizable.



The screenshot shows the 'Data Sources / InfluxDB' configuration page in Grafana. The 'Settings' tab is selected. The 'Name' field is set to 'InfluxDB'. A 'Default' toggle switch is turned on. The 'Query Language' dropdown is set to 'Flux'. A note indicates that support for Flux in Grafana is currently in beta and provides a link to report issues: <https://github.com/grafana/grafana/issues>. The 'HTTP' section shows the URL as 'http://influxdb:8086', access as 'Server (default)', and timeout as 'Timeout in seconds'. The 'Auth' section includes options for Basic auth, TLS Client Auth, Skip TLS Verify, and Forward OAuth Identity, all of which are disabled. The 'Custom HTTP Headers' section has a '+ Add header' button. The 'InfluxDB Details' section contains fields for Organization ('GuillermoQuintana'), Token ('configured'), Default Bucket ('admin@admin.com'), Min time interval ('10ms'), and Max series ('1000'). At the bottom are 'Back', 'Explore', 'Delete', and 'Save & test' buttons.

Una vez definida nuestra TSDB dentro de graffana como fuente de datos podemos proceder a crear un dashboard nuevo, cada dashboard puede tener múltiples paneles de diferente fuentes, y son altamente configurables. Para esto se utiliza un lenguaje de querys llamado flux, el cual especificamos en la sección Query Language cuando definimos nuestra fuente. Flux es un lenguaje de consulta desarrollado por InfluxData para trabajar con la base de datos InfluxDB. Las consultas en Flux siguen una estructura de canalización (pipeline) donde los datos se pasan a través de varias etapas, aplicando operaciones y transformaciones en cada paso. Cada panel de un dashboard en graffana toma sus datos a partir de una query y en este caso utilizaremos la siguiente consulta.

```
from(bucket: "iotdata")
|> range(start: v.timeRangeStart, stop:v.timeRangeStop)
|> aggregateWindow(every: 1s,fn: mean)
|> filter(fn: (r) =>
r._measurement == "Temperature" and
r.device_id == "e2e78334"
)
```

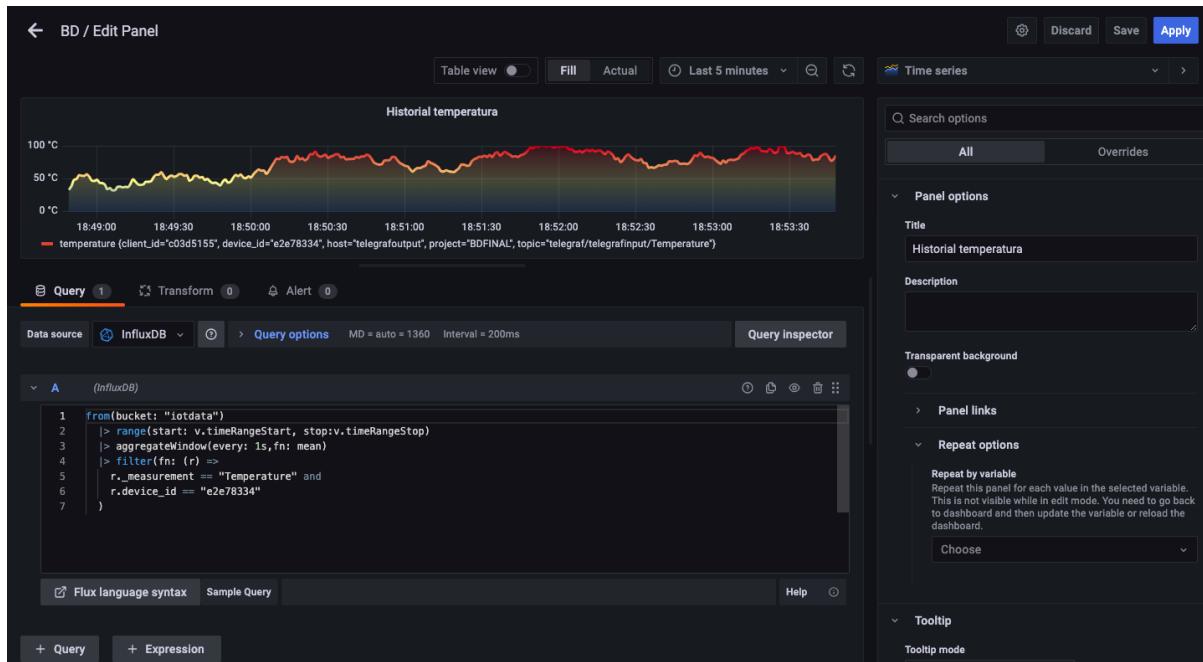
**from(bucket: "iotdata"):** Esta línea especifica la fuente de datos para la consulta. Indica que los datos se extraen del bucket (almacén) llamado "iotdata".

**|> range(start: v.timeRangeStart, stop:v.timeRangeStop) :** Esta línea establece el rango de tiempo para la consulta. start y stop son parámetros que indican el inicio y fin del rango de tiempo. Los valores v.timeRangeStart y v.timeRangeStop se utilizan para obtener los límites de tiempo establecidos por el entorno de ejecución.

**|> aggregateWindow(every: 1s,fn: mean) :** Esta línea agrega una ventana de tiempo y calcula el promedio de los valores dentro de cada ventana. every especifica la duración de la ventana, que en este caso es de 1 segundo. fn indica la función de agregación utilizada, en este caso, "mean" (promedio).

**|> filter(fn: (r) => r.\_measurement == "Temperature" and r.device\_id == "e2e78334") :** Esta línea filtra los resultados en función de ciertas condiciones. fn indica la función de filtrado utilizada, que en este caso es una función anónima (r) => .... El filtro se aplica a cada registro (r) y se verifica si el campo \_measurement es igual a "Temperature" y el campo device\_id es igual a "e2e78334".

Múltiples parámetros y distintos tipos de visualización pueden ser configurados para un panel. Otras opciones de grafana incluyen alertas por valores críticos dentro del mismo panel.



En este ejemplo se presenta un dashboard que permite visualizar la temperatura actual, junto con un histograma y una tabla de datos en crudo.



Por último pero no menos importante demostraremos como un nodo suscripto puede tomar decisiones en base a los datos recibidos por el broker iniciando nuestro nodo cliente. En este caso solo se emite una alerta a modo de mensaje pero suponiendo que este nodo puede ser un microcontrolador como una ESP32 este podría actuar, por ejemplo, sobre un módulo de relés que se encargue de controlar algún tipo de refrigeración para mantener las temperaturas por debajo del margen definido.

```
qq@qq ~ /Desktop/BDFINAL/BDFINAL2/iotDB/nodo_suscriptor main
python3 nodo_suscriptor.py

Temperatura actual: 68.11
¡Alerta! La temperatura ha excedido los 60°C: 68.11
```

También podemos apreciar en el dashboard del broker EQMX los clientes conectados, en este caso se tratan de nuestros dos nodos en python y la instancia de telegraf.

The screenshot shows the EQMX dashboard interface. On the left, there is a sidebar with various menu items: Monitor, Clients (which is highlighted in green), Topics, Subscriptions, Rule Engine, Analysis, Plugins (with a checked checkbox), Modules, Tools, and Alarms. Below the sidebar, the user is logged in as 'admin'. The main area is titled 'Clients' and displays a table of connected clients. The table has columns for Client ID, Username, IP Address, Keepalive(s), Expiry Inte, and Operation. There are three entries in the table:

Client ID	Username	IP Address	Keepalive(s)	Expiry Inte	Operation
MzExNjE4MzgyMDk...	undefined	172.20.0.1:41964	60	0	<button>Kick Out</button>
MzExNjE3OTYyMDM...	undefined	172.20.0.1:47032	60	0	<button>Kick Out</button>
Telegraf-Consumer...	undefined	172.20.0.2:42626	60	0	<button>Kick Out</button>

Para concluir, este trabajo de investigación ha explorado el uso de una base de datos de serie de tiempo (TSDB) en el contexto de la infraestructura IoT. Mediante la implementación de un entorno Docker Compose, se logró integrar de manera efectiva un sensor IoT, un broker MQTT, una instancia de Telegraf, una base de datos InfluxDB y una instancia de Grafana.

El uso de una base de datos de serie de tiempo como InfluxDB resultó fundamental para la gestión eficiente de los datos generados por los nodos IoT.

La arquitectura propuesta, con el broker MQTT EMQX como punto central de comunicación, la instancia de Telegraf como consumidor de mensajes, la base de datos InfluxDB para el almacenamiento de datos y la instancia de Grafana para la visualización, demostró ser eficiente y escalable. Mediante el uso de contenedores Docker y Docker Compose, se logró una implementación modular y replicable de la infraestructura, lo que facilita la gestión y el despliegue en entornos de producción.

Una opción de escalabilidad que brinda esta solución es la utilización de Kubernetes para gestionar un cluster de alta disponibilidad o HA CLUSTER. La utilización de Kubernetes como plataforma de orquestación de contenedores permite aprovechar sus capacidades de escalabilidad, tolerancia a fallos y administración centralizada para mejorar aún más la infraestructura. Mediante la implementación de un clúster en Kubernetes, se logra una mayor resiliencia y capacidad de respuesta en entornos IoT dinámicos.

El escalado horizontal proporcionado permite adaptar la infraestructura a las demandas cambiantes de los dispositivos IoT, garantizando un rendimiento óptimo incluso en situaciones de alta carga de trabajo. Además, la tolerancia a fallos integrada en asegura que la infraestructura siga siendo operativa en caso de errores o interrupciones, minimizando los tiempos de inactividad y brindando continuidad en el procesamiento y almacenamiento de los datos.

En general, este trabajo ha brindado una comprensión más profunda de las bases de datos de serie de tiempo y su aplicación en el contexto de la infraestructura IoT. Se ha demostrado cómo una TSDB, como InfluxDB, puede jugar un papel clave en la gestión de datos en tiempo real y en el análisis histórico de datos IoT. Además, la utilización de herramientas como Grafana ha permitido una visualización efectiva y personalizada de los datos, brindando a los usuarios una interfaz amigable para explorar y comprender los patrones y tendencias presentes en los datos generados por los sensores IoT.

Se han sentado las bases para el desarrollo de soluciones IoT más robustas y escalables, donde las bases de datos de serie de tiempo juegan un papel fundamental en la gestión y el análisis de datos en tiempo real. El conocimiento adquirido en este trabajo proporciona una base sólida para futuras investigaciones y aplicaciones en el campo de las bases de datos y la infraestructura IoT.