Hardware:

- Raspberry Pi4 board
- DHT11 sensor
- USB to TTL adapter for serial console
- 16GB SD card with Reader

Software:

- Laptop running on Ubuntu 20.04
- Installed: Docker, docker-compose, git , picocom, postman

Setting up the edgex cluster:

Clone the following repo on your laptop. You need to run the docker-compose.yml under edgex-deployment/containers/edgex/recipe to start the edgex cluster.

```
git clone https://github.com/msatpathy26/edgex-deployment.git
cd edgex-deployment/containers/edgex
```

But before you run, update the *WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_ADDRESS* in the docker-compose.yml with the local IP* address. An example below shows that we update the address to 192.168.140.182.

** Please note that 0.0.0.0 will not work here even if you are running the MQTT message broker in the same host environment.*

```
sed -i
's/WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_ADDRESS.*/WRITABLE_PIPELIN
E_FUNCTIONS_MQTTSEND_ADDRESSABLE_ADDRESS: 192.168.140.182/'
docker-compose.yml
```

Now fetch the respective container images used by the edgex cluster

```
docker-compose pull
docker image ls
```

Now using the following commands start the cluster and verify the running containers.

```
docker-compose up -d
docker-compose ps
```

Also you can verify the running services by accessing http://0.0.0.0:8500/ on the web browser.

Open postman and disable SSL certificate verification.

Now add these following REST APIs.

SET value descriptors
(for temperature and humidity data.)

```
Method:POST
URI:http://0.0.0.0:48080/api/v1/valuedescriptor
Payload settings:Set Body to “raw” and “JSON”

Payload data:
{
      "name": "humidity",
      "description": "Ambient humidity in percent",
      "min": "0",
      "max": "100",
      "type": "Int64",
      "uomLabel": "humidity",
      "defaultValue": "0",
      "formatting": "%s",
      "Labels":   [
            "Environment",
            "Humidity"
      ]
}
```

On posting the REST API you should receive a 200 OK on success.

```
Method:POST
URI:http://0.0.0.0:48080/api/v1/valuedescriptor
Payload settings:Set Body to “raw” and “JSON”
Payload data:
{
      "name": "temperature",
      "description": "Ambient temperature in deg C/F",
      "min": "-100",
      "max": "120",
      "type": "Int64",
      "uomLabel": "humidity",
      "defaultValue": "0",
      "formatting": "",
      "Labels":   [
            "Environment",
            "Temperature"
      ]
}
```

Upload the device profile:

Method: POST

URI: http://0.0.0.0:48081/api/v1/deviceprofile/uploadfile

Payload settings:

- Set Body to "form-data"
- Hover over KEY and select "File"
- Select the yaml file: sensorClusterDeviceProfile.yaml (under edgex-deployment/containers/edgex in cloned repo)
- In the KEY field, enter "file" as key

Device registration:

Method: POST

URI:http://0.0.0.0:48081/api/v1/device

Payload settings: Set Body to "raw" and "JSON"

Payload data:

```
{
      "name": "Temp_and_Humidity_sensor_cluster_01",
      "description": "Raspberry Pi sensor cluster",
      "adminState": "unlocked",
      "operatingState": "enabled",
      "protocols": {
            "example": {
                  "host": "dummy",
                  "port": "1234",
                  "unitID": "1"
                  EdgeX Foundry: Demo
            }
      },
      "labels": [
            "Humidity sensor",
            "Temperature sensor",
            "DHT11"
      ],
      "location": "Bangalore",
      "service": {
            "name": "edgex-device-rest"
      },
      "profile": {
            "name": "SensorCluster"
      }
}
```

Testing with data:

Send a data sample:

```
Method:POST
URI:http://0.0.0.0:49986/api/v1/resource/Temp_and_Humidity_sensor_cluster_01/temperature
Payload settings:
  Set Body to “raw” and “text”
  Payload data: 30 (any integer between -100 & 120)
```

View Received Data:

```
Method:GET
URI: http://0.0.0.0:48080/api/v1/reading
```

Data Count:

```
Method:GET
URI:http://0.0.0.0:48080/api/v1/event/count
```

If your sent and received data sample matches in value & count the edgex cluster is functioning correctly.

Booting the raspberry Pi

- **Downloading Ubuntu for arm**

```
mkdir  -p image
cd image
wget
http://old-releases.ubuntu.com/releases/focal/ubuntu-20.04.2-preinstalle
d-server-arm64+raspi.img.xz
```

- **Extract image**

```
unxz ubuntu-20.04.2-preinstalled-server-arm64+raspi.img.xz
cd -
```

- **Format SD card**

```
sudo dd if=/dev/zero of=/dev/sdb bs=4096 status=progress
```

- **Flash image to SD card**

```
cd edgex-deployment/scripts/
sudo ./flash-image.sh \
../image/ubuntu-20.04.2-preinstalled-server-arm64+raspi.img sdb
```

- **Load SD card**
  Insert the SD card into the card slot of the Pi board.

- **Serial port connection**
  Connect the raspberry pi GPIO pins to TTL in the following way:
  Raspberry Pi GPIO14 (UART Tx) ---> TTL (UART Rx)
  Raspberry Pi GPIO15 (UART Rx) ---> TTL (UART Tx)
  Raspberry Pi PIN6 (GND) ---> TTL (GND)

  Refer Figure1 below.

  Plug in the USB end of TTL-USB adapter to the laptop's USB port.

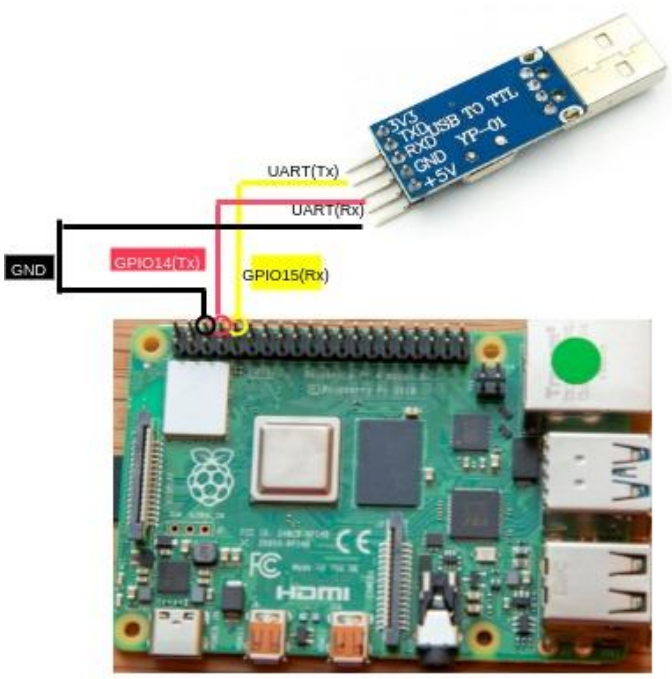


***Figure1**: Serial Port connection*

- **Serial port console**
  Open terminal on Linux laptop and run..

```
sudo picocom -b 115200 /dev/ttyUSB0
```

- **Boot and login**
  Power on the Pi board. The boot logs should show up on the serial port console.
  Provide credentials for first login.

```
Login: ubuntu
Password: ubuntu
```

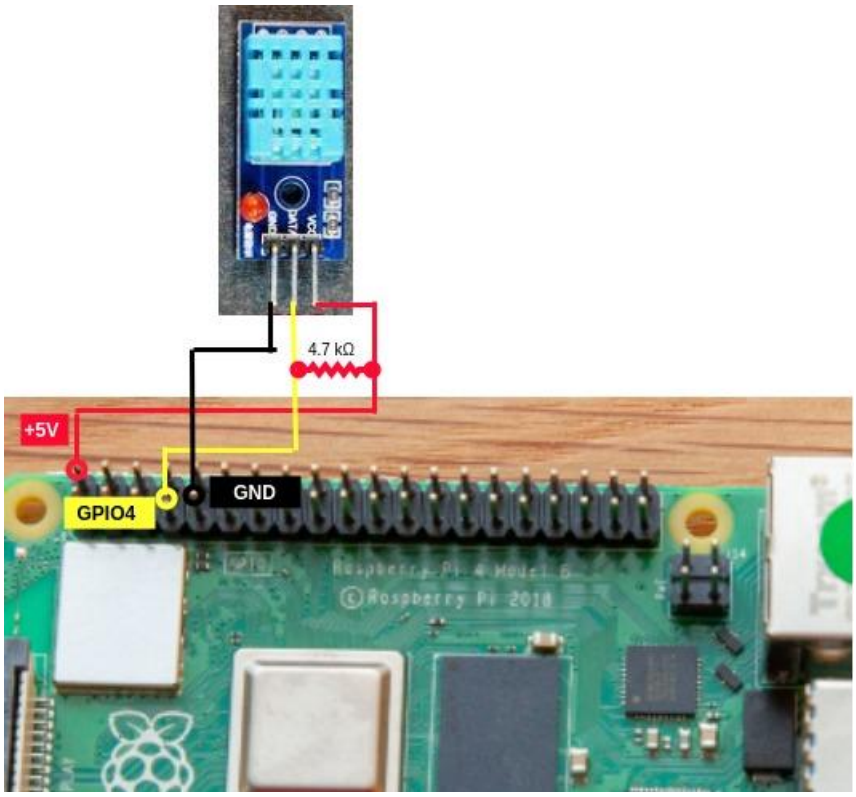
After that it will ask to change password and confirm password. Following password change the Pi board will be ready for use.

<u>Connecting the sensor</u>

Connect the sensor in the following way:

Sensor VDD <----------> Rpi PIN2 (+5V)
Sensor DATA <----------> Rpi PIN7 (GPIO4)
Sensor GND <----------> Rpi PIN9 (GND)

A pull-up resistor of about 4.7KΩ must be placed between the VDD and DATA pin of the sensor. Without the pull-up resistor, the sensor may not work correctly.



-

Figure2: *DHT11 sensor connection*

The LED on the sensor should glow if connection is done correctly.

<u>Configure the target platform</u>:

- Once booting and logging into raspberry pi is done, run the following steps.
- Copy the target directory under edgex-deployment/scripts to the target.
  scp -r target ubuntu@192.168.140.73:/home/ubuntu

- On the target run the following commands to install dependencies.

```
sudo apt-get install python3-pkg-resources -y
sudo apt-get install python3-pip -y

cd /home/ubuntu/target
pip install -r requirements.txt
cd -
```

```
sudo apt install libgpiod-dev git build-essential -y
sudo apt install libgpiod2 -y
```

Now replace libgpiod_pulsein provided by Adafruit-Blinka as it's built for 32bit with our own version of 64bit

```
git clone https://github.com/adafruit/libgpiod_pulsein.git

cd libgpiod_pulsein/src
make

cp libgpiod_pulsein
~/.local/lib/python3.8/site-packages/adafruit_blinka/microcontroller/bcm283x/
pulseio/libgpiod_pulsein
```

Now go back to target directory and run the script to read sensor data:

```
sudo chmod og+rwx /dev/gpio*
```

Update the EDGEX server IP or DNS within the script.
In the below command we update it to "192.168.140.182".

```
sed -i 's/^edgexip = .*/edgexip = "192.168.140.182"/g' sensor.py
```

Run the script to fetch data from sensor and post it to the EDGEX server

```
python3 sensor.py
```

```
ubuntu@ubuntu:~/target$ python3 sensor.py
Temp: 87.8 F / 31.0 C    Humidity: 68%
Temp: 84.2 F / 29.0 C    Humidity: 57%
```

The fetched data from the sensor will be printed on the console. The same data should also get posted to the edgex cluster running on your laptop.

You can either use postman or run the cluster as a foreground process ('docker compose up' , without '-d') to verify the data is being received. In the latter case the messages get printed on the console running edgex.

Starting the MQTT broker

Before starting the MQTT broker it needs some customization. Run the following commands to customize the docker-compose.yml file used for running the mosquitto MQTT message broker.

```
cd edgex-deployment/scripts/
./mqtt-broker-setup.sh
```

Once the script runs without any error, we can start the MQTT broker. To start the broker run the following commands.

```
cd ../containers/mqtt-broker/
docker-compose up -d
```

You can also verify the running MQTT broker like this..

```
$ docker ps |grep mosquitto

df39d2c38901   eclipse-mosquitto:2.0.11   "/docker-entrypoint.…"   24 minutes
ago   Up 24 minutes   mosquitto
```

Test the MQTT broker

Now to test if the message broker is able to receive and deliver data correctly a test script has also been provided. Go back to the scripts directory and run mqtt-subscribe-test.py. You need to have paho-mqtt python module installed as a required dependency before running the script.

```
cd ../../scripts/
./mqtt-subscribe-test.py
```

On successfully connecting to the MQTT broker it should display “Connected with result code 0” Every data sample being posted to the MQTT broker must be printed on the console thereafter.

The following example shows a successful subscription to the topic ‘sensor-data-dht11’.

```
$ ./mqtt-subscribe-test.py

Connected with result code 0
sensor-data-dht11
b'{"id":"7c8ac02c-5c8a-427c-9ed1-21db4b297df2","device":"Temp_and_Humidity_sens
or_cluster_01","origin":1631976204089140278,"readings":[{"id":"dca3187d-ff63-49
13-835a-9990a437bbe4","origin":1631976204089077755,"device":"Temp_and_Humidity_
sensor_cluster_01","name":"temperature","value":"33","valueType":"Int64"}]}'
```

Start the InfluxDB database

Run the following commands to start the influxdb database to store the device data.

```
cd ../containers/influxdb/
docker-compose up -d
```

You can verify the running container and the database in the following way.

```
$ docker ps|grep influxdb
41623827e4bc influxdb:1.8    "/entrypoint.sh infl…"   2 hours ago   Up 2 hours

$ docker exec -it influxdb /bin/bash
root:/# influx -username root -password password
Connected to http://localhost:8086 version 1.8.9
InfluxDB shell version: 1.8.9
> SHOW DATABASES;
name: databases
name
----
sensordata
_internal
>
```

Connect database to MQTT broker

Now we need to connect the database with the MQTT broker to receive device data.
For that we will run an application called *'messenger'*. Messenger will subscribe to MQTT broker and fetch device data being sent from the sensor via edgex. The same data will be written to influxdb by *'messenger'*.

```
#Go to messenger directory
cd ../../containers/messenger

#Check app.py and Dockerfile are available
ls
app.py  Dockerfile

#Build the image with following command
docker build --rm --tag messenger:1.0 .

#Now run messenger with following command
docker run -d --name messenger  --network host messenger:1.0
```

* You can even run messenger on foreground (without -d option) to view the device data transaction log on-screen
* Every time you want to change the subscription topic, broker ip, database ip, database credentials you must update app.py and rebuild the image. Default settings are as following:

```
broker_address  = "0.0.0.0"
topic           = "sensor-data-dht11"
dbhost          = "0.0.0.0"
dbport          = 8086
dbuser          = "root"
dbpassword      = "password"
dbname          = "sensordata"
```

Once connected successfully, you can verify the device data being populated into the database by logging into the influxdb instance.

```
$ docker exec -it influxdb /bin/bash
root:/# influx -username root -password password
Connected to http://localhost:8086 version 1.8.9
InfluxDB shell version: 1.8.9
> use sensordata
Using database sensordata
> SHOW measurements;
name: measurements
name
----
Temp_and_Humidity_sensor_cluster_01
> SELECT * FROM "Temp_and_Humidity_sensor_cluster_01";
name: Temp_and_Humidity_sensor_cluster_01
time                gateway                           location    temperature
----                -------                           --------    -----------
1631999448000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 31
1631999496000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 31
1631999509000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 31
1631999510000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 31
1631999511000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 31
1631999515000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 30
1631999516000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 30
1631999529000000000 Temp_and_Humidity_sensor_cluster_01 Bangalore 29
```

Data visualization with grafana

By now we have seen how to start all the above mentioned services (edgex, mosquitto mqtt broker, influxdb and messenger) and verify that data is being populated into the influxdb database.

To visualize the captured device data we need to start the grafana and configure it.

To start the grafana service use either of the following two ways:

1. Using docker pull / run command:

```
docker pull grafana/grafana:8.0.1-ubuntu
docker run -d --rm --name=grafana -p 3000:3000 --network host
grafana/grafana:8.0.1-ubuntu
```

OR

2. Using docker-compose

```
cd ../../containers/grafana
docker compose up -d
```

The running of grafana service can be verified as

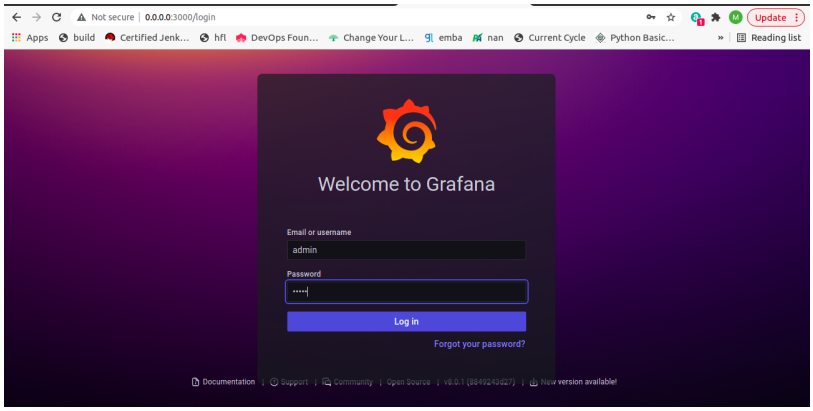
```
$ docker ps|grep grafana

0c43924d6852   grafana/grafana:8.0.1-ubuntu   "/run.sh"   51 minutes ago      Up
51 minutes   grafana
```

Once grafana service is up, we need to open the grafana UI from the web-browser at http://0.0.0.0:3000. Then follow the steps as given below to configure the grafana dashboard.

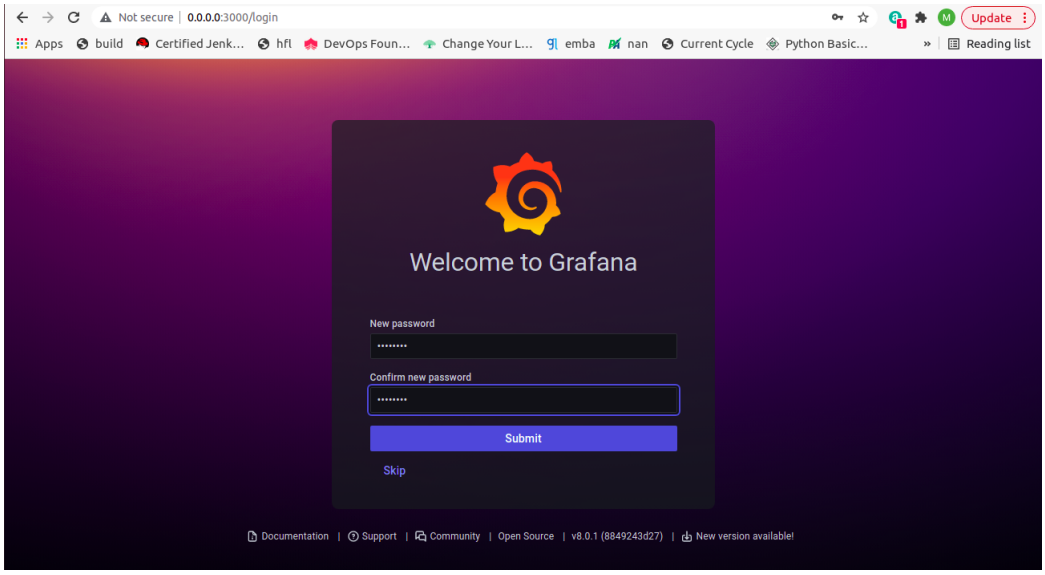
1. Initial login

Username: admin
Password: admin



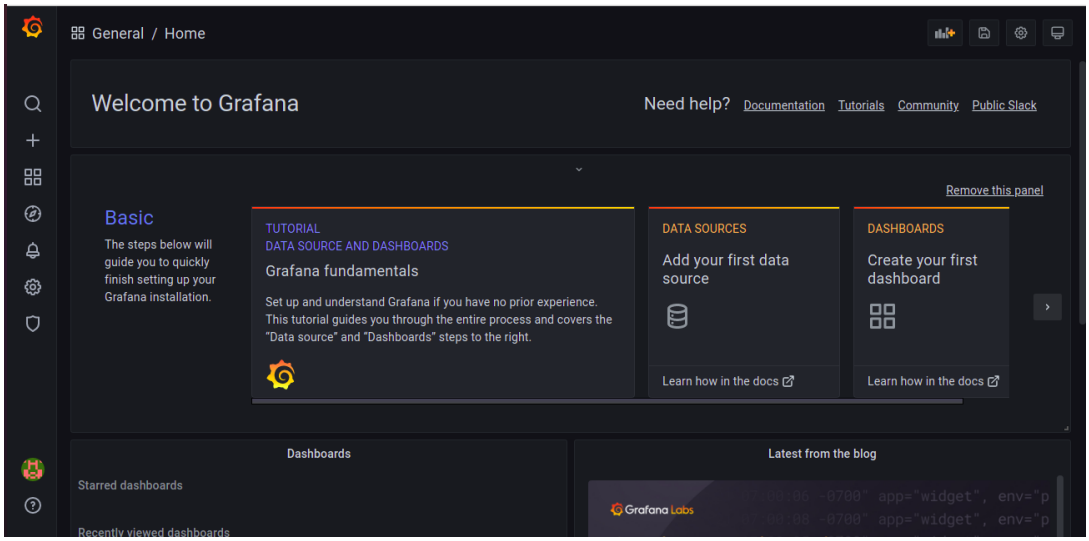
2. Change password

Immediately after first login, you are asked to change password

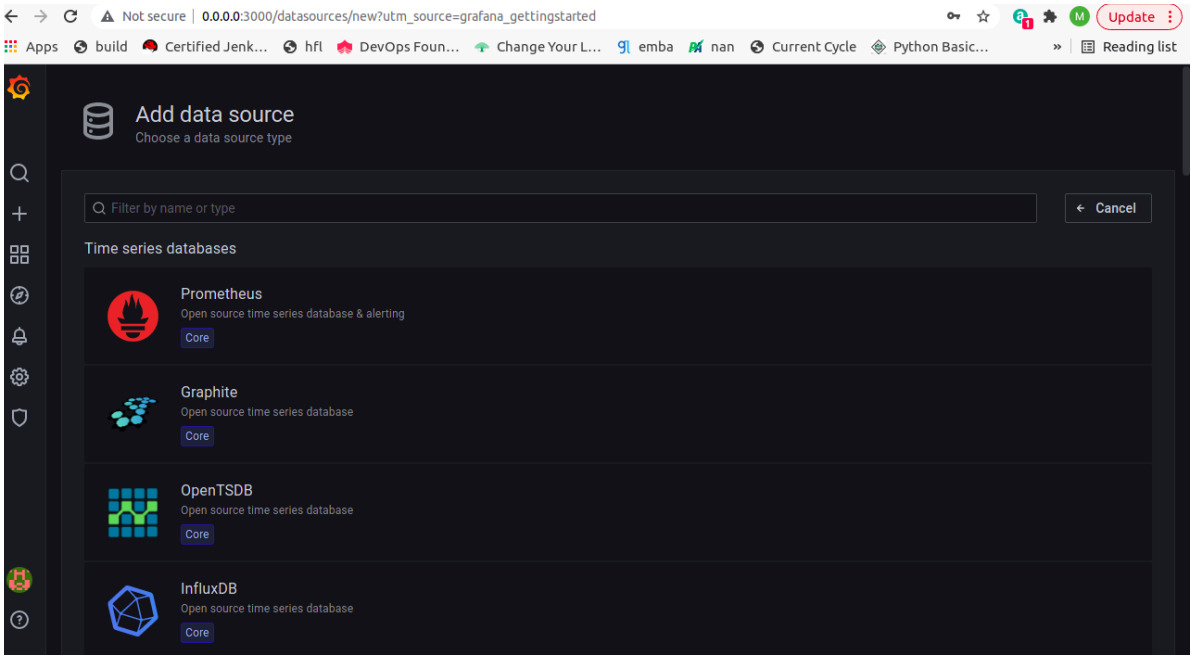


3. **Add data source** :

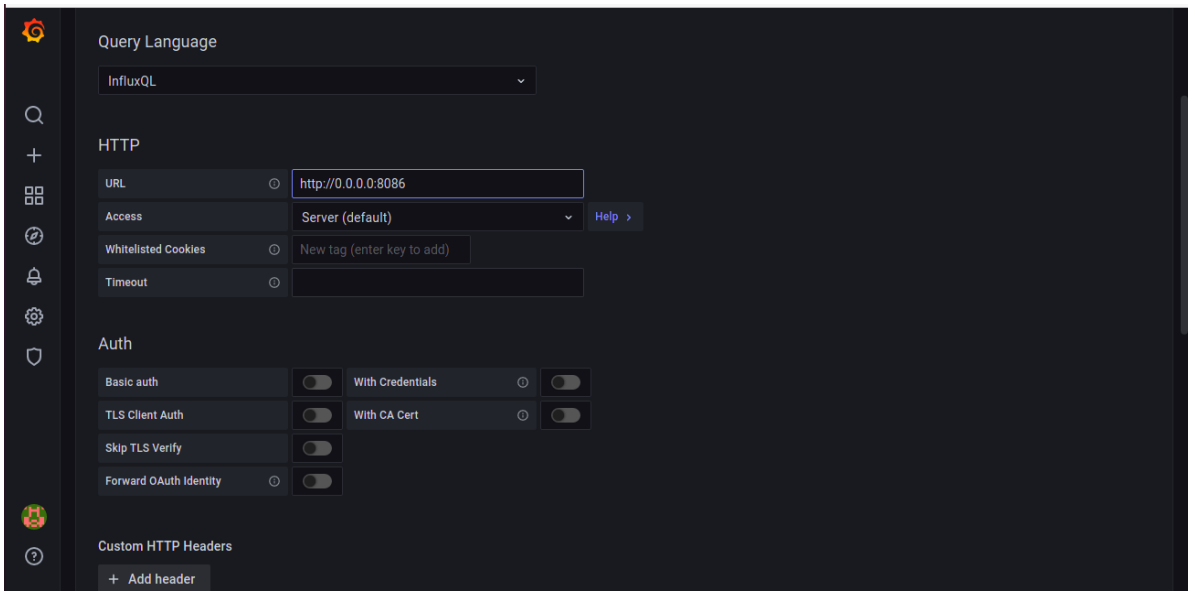
Go to '<u>Add your first data source</u>' to link grafana with the influxdb device database.



Select ‘InfluxDB’ from the list of databases.



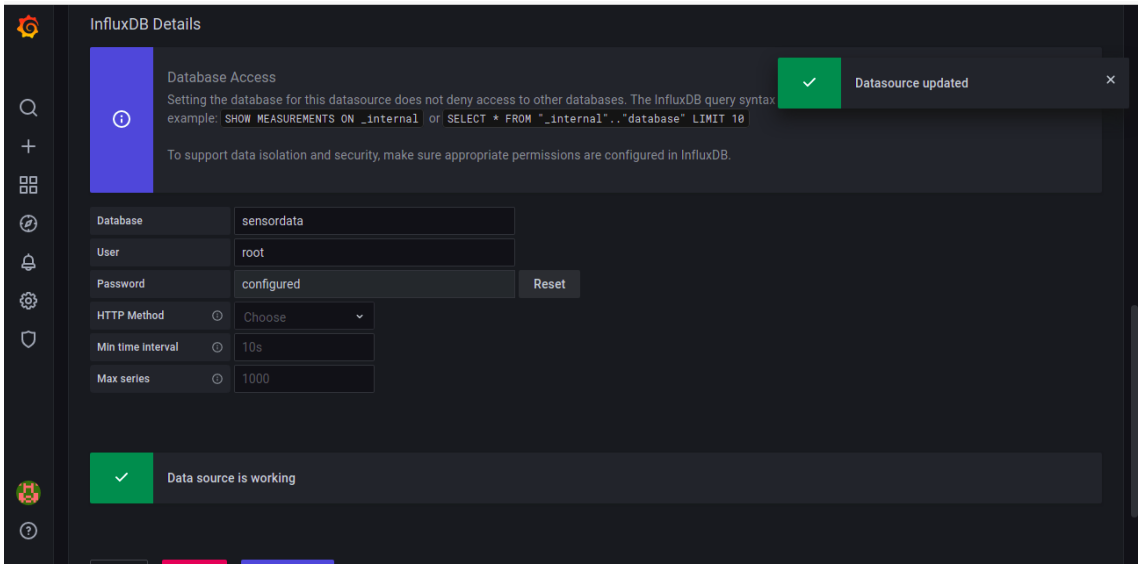
Add HTTP > URL as ‘http://0.0.0.0:3000’



Now fill in the following fields to access the database:
Database: sensordata

User: root *(influxDB userid)*
Password: password *(influxDB password)*

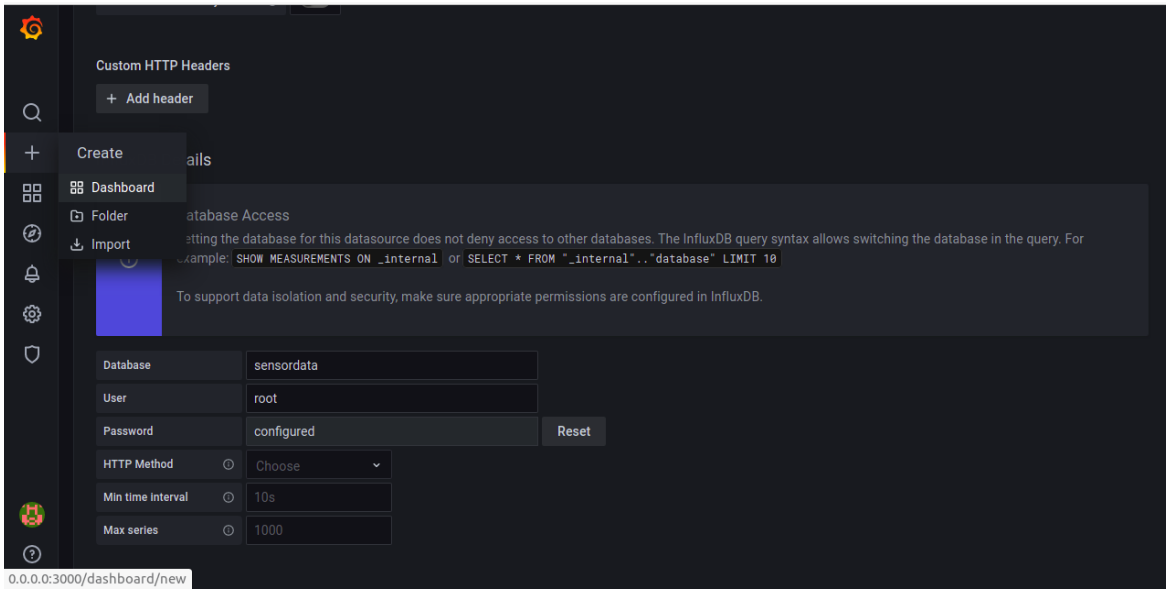
Click on ‘Save & test’ button given below.



On success, you will see a notification at the bottom of the page saying ‘Data source is working’. (As shown in the figure)

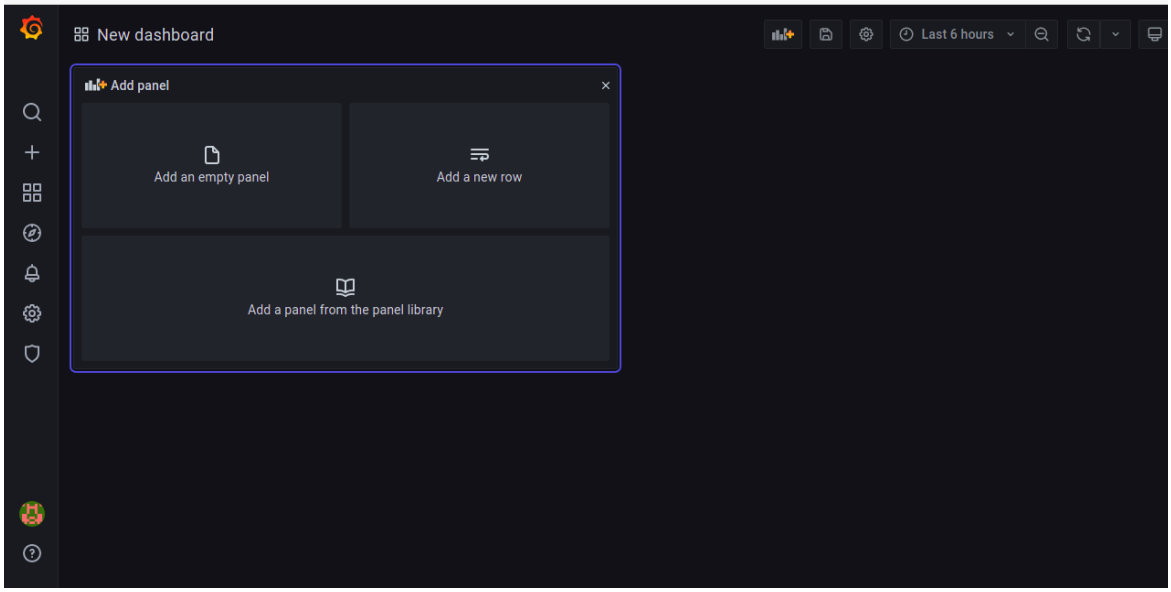
4. **Configure the dashboard**

To add a new dashboard, click on the ‘ + ‘ mark on the left side panel and go to the dashboard.

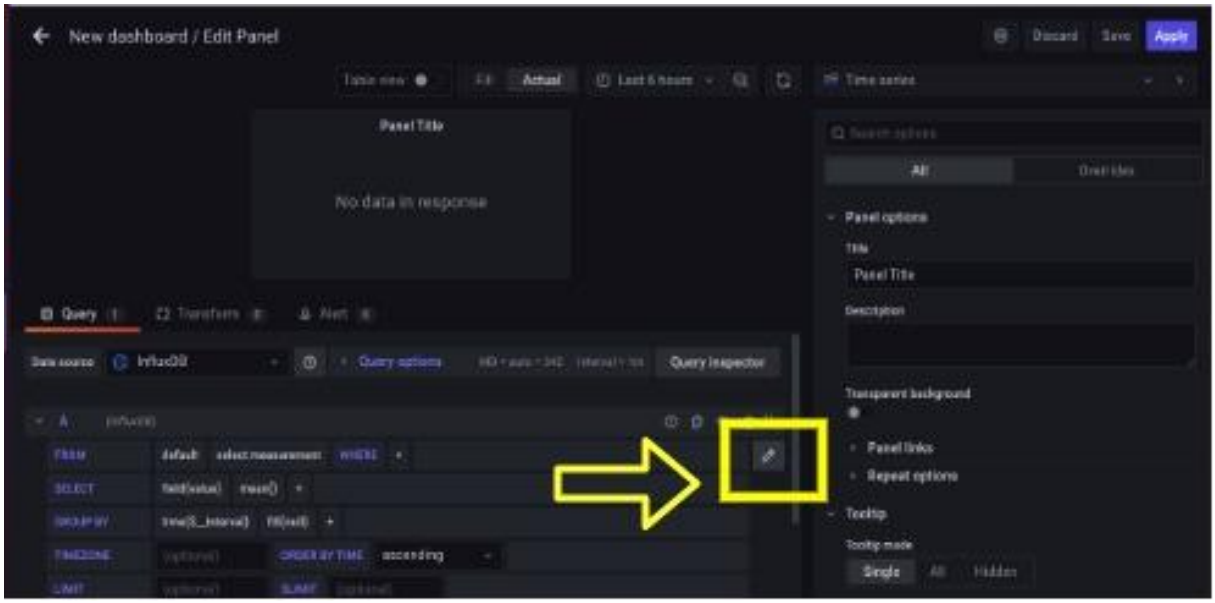


(You can also do the same from the home page > create your first dashboard)

Select ‘Add an empty panel’

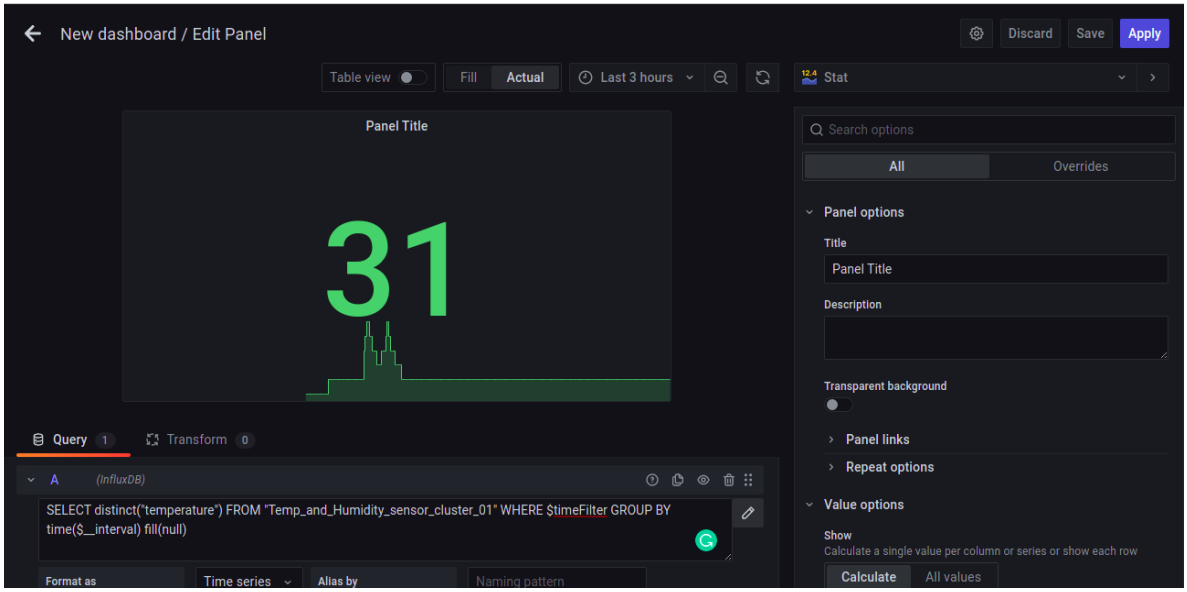


On the lower panel click on the edit (pen icon) to the right, and enter the query:



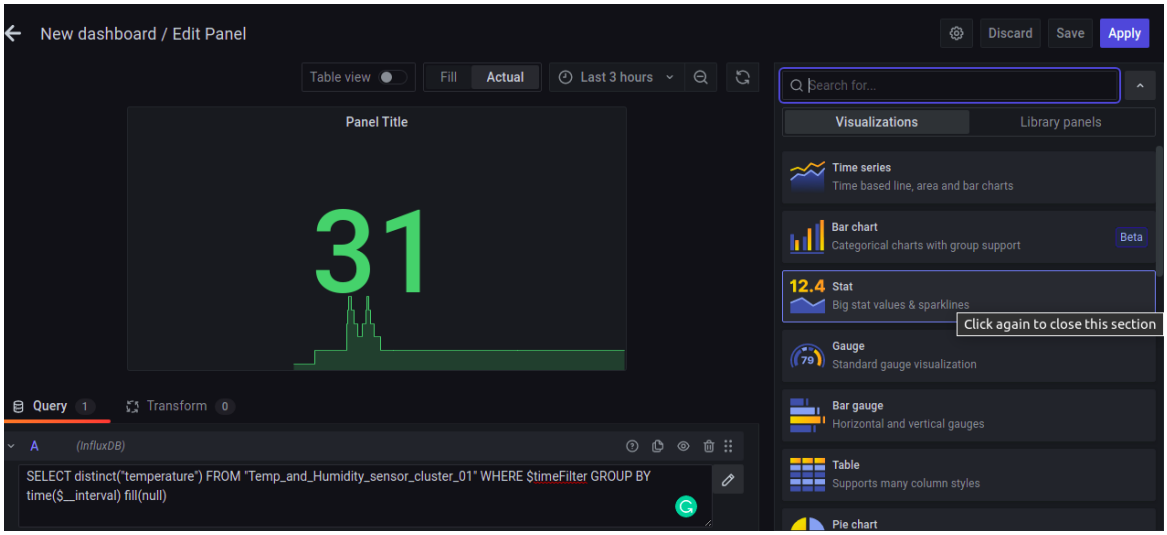
Paste the following query string into the query field as below:

SELECT distinct("temperature") FROM "Temp_and_Humidity_sensor_cluster_01" WHERE $timeFilter GROUP BY time($__interval) fill(null)



You can repeat the same for humidity data as well:
SELECT distinct("humidity") FROM "Temp_and_Humidity_sensor_cluster_01" WHERE $timeFilter GROUP BY time($__interval) fill(null)

Choose your visualization style from the list on the right. You can also adjust fonts, colours etc.



Save the dashboards after finalizing your visualization styles.