DEPLOY AND USE OF GRAFANA

INDEX

1. CREATION AND EXECUTION OF THE IMAGE IN DOCKER

HOP Ubiquitous has a version of their fiware avaiable in a Docker container which “se puede” can download from Github. If you haven´t done this step we recommend you that you download the repository https://github.com/HOP-Ubiquitous/fiware-docker-infrastructure and you follow the recommend steps for its running.

Once we have execute the HOP Ubiquitous’s container, we must have the next processes execute on Docker (we can see it using the command “docker ps” in terminal):

```
CONTAINER ID        IMAGE                            PORTS
3996f3dc01b5        iotagent                         0.0.0.0:5693->5683/udp
dockercompose_iotagent_1
e94378a74a38        smartsdk/quantumleap             0.0.0.0:8668->8668/tcp
dockercompose_quantumleap_1
a57a750c86c8        grafana/grafana                  0.0.0.0:3000->3000/tcp
dockercompose_grafana_1
e6ef4afd30f7        telefonicaiot/fiware-sth-comet   0.0.0.0:8666->8666/tcp
dockercompose_sth_1
fa5fbce0c89e        fiware/orion                     0.0.0.0:1026->1026/tcp
dockercompose_orion_1
6a22f61d772f        fiware/cygnus-ngsi               5050/tcp, 0.0.0.0:8081->8081/tcp
dockercompose_cygnus_1
a29f3c7f682f        crate:1.0.5                      0.0.0.0:4200->4200/tcp,
0.0.0.0:4300->4300/tcp, 5432-5532/tcp   dockercompose_crate_1
a4652c49e3fa        mongo:3.2                        0.0.0.0:27016->27017/tcp
dockercompose_mongo_1
```

In case it is not executing some of this processes (as Grafana), we do the next process:

1. We stop the execution from Docker container – "docker-compose down"
2. We remove the image of the application which is not running. – "docker rmi imageName"
3. We back to execute the Docker container – "docker-compose up - d"

2. DATA SOURCE DEFINITION TO GRAFANA

Grafana is an open source API to generate dashboards from data contained in a CrateDB database. CrateDB is an open source SQL database design to store data of IoT devices. CrateDB is not suitable with Orion Context Broker, which manage the entire lifecycle of information, so we need a connector between CrateDB and Orion Context Broker. QuantumLeap is an API to connect CrateDB and Orion Context Broker, in other words, it convert Orion Context Broker information in a suitable datatype with CrateDB and store it in this database.

2.1. Create QuantumLeap`s subscription to Orion

The next step is connect Data Source to Grafana. First we have to create a subscription from QuantumLeap to Orion to receive the changes from devices and it save on CrateDB database. For it, we execute the POSTMAN program and we open the Orion collection. We enter in the "v2" → "subscriptions" → "create subscription v2" section and the body section we defined the next subscription:

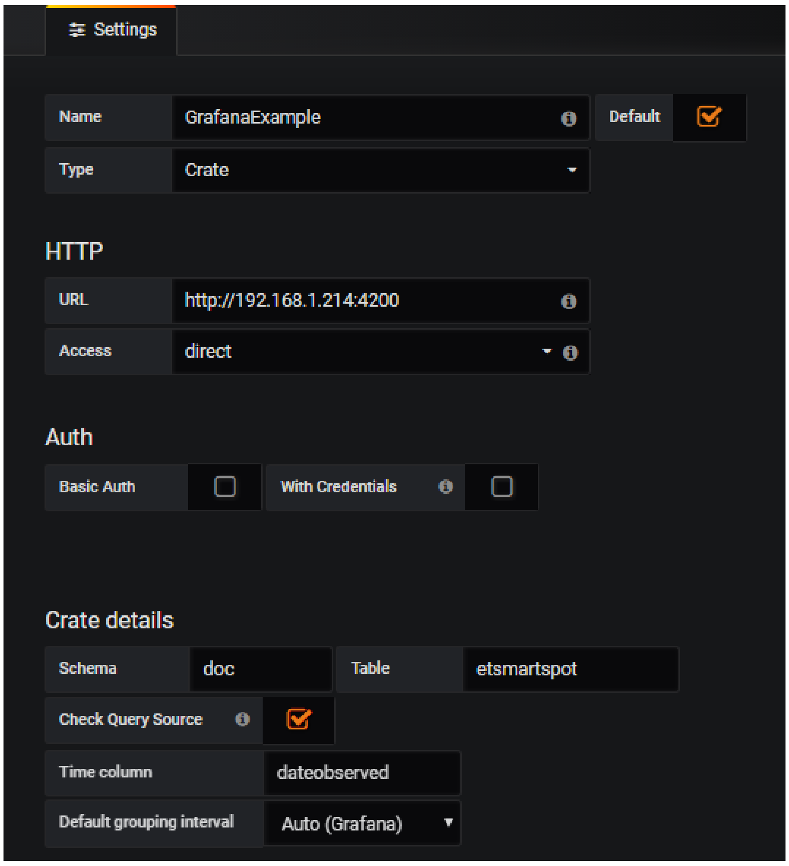
```
{
    "description": "grafana_subscription",
    "subject": {
        "entities": [
        {
            "idPattern": ".*",
            "type": "SmartSpot"
        }
        ]
    },
    "notification": {
        "http": {
            "url": "http://192.168.1.16:8668/notify"
        },
        "attrs": [
        "CO",
        "H2S",
        "NO2",
        "O3",
        "SO2",
        "batteryLevel",
        "humidity0",
        "humidity1",
        "temperature0",
        "temperature1",
        "nearDevicesHour",
        "nearDevicesMinute",
        "nearDevicesTenMinutes",
        "noise"
        ],
        "metadata": ["dateCreated", "dateModified"]
    }
}
```

We prove that the subscription has been made correctly and it is active. Once it is active, we check the QuantumLeap log to see if work correctly. To access the log, we execute the command "`docker logs --follow dockercompose_quantumleap_1`". We must prove that QuantumLeap receive the subscription and it support the attributes of an entity type that we want store in CrateDB.
We have already connect our devices with the CrateDB database from QuantumLeap. Now is the moment to defined our data source on Grafana. Before configured CrateDB, we can check if it is executing correctly. For it, we execute in Orion a GET method to the next address: http:/tu_ip:4200/_sql and we check that exist the adequate data structure.

2.2. Connection of Grafana with CrateDB

The first step is discover what our device ip is (with ifconfig command). Once we know our IP, we access to the next URL: http://your_ip:3000 (3000 is the port of grafana).

In the login from this URL we write in the user field: "admin" and in the password field: "admin". We enter the data source section and we leave the setting file like this.


Settings

Name	GrafanaExample	Default	✓
Type	Crate		

HTTP

URL	http://192.168.1.214:4200
Access	direct

Auth

Basic Auth	☐	With Credentials	☐

Crate details

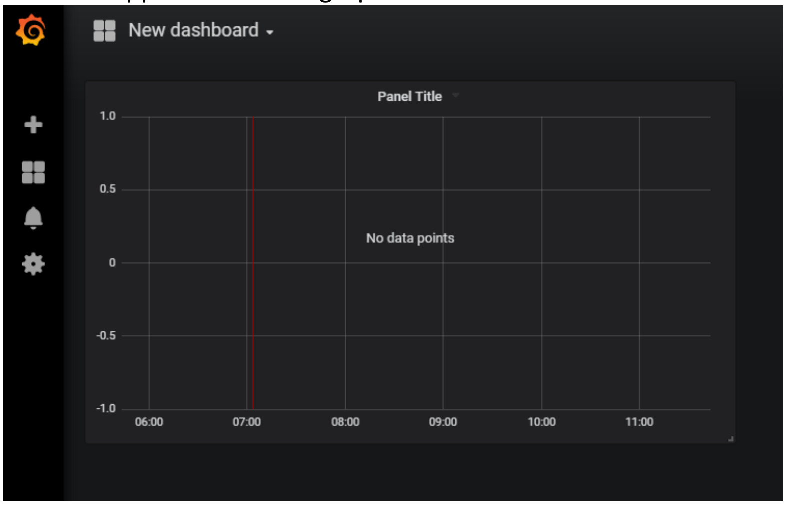
Schema	doc	Table	etsmartspot
Check Query Source	✓		
Time column	dateobserved		
Default grouping interval	Auto (Grafana)		

- Data source type in our case is, as we have named before, CrateDB.
- The URL is you IP again, and 4200 is CrateDB port by default.
- When everything is working inside the localhost, the access is direct.
- QuantumLeap create a table "et" + name of the entity type which we have subscribed. In our case, being SmartSpot entities, the table it call "etsmartspot".
- The name of the index from time column is "dateobserved". This name can see in the CrateDB structure.

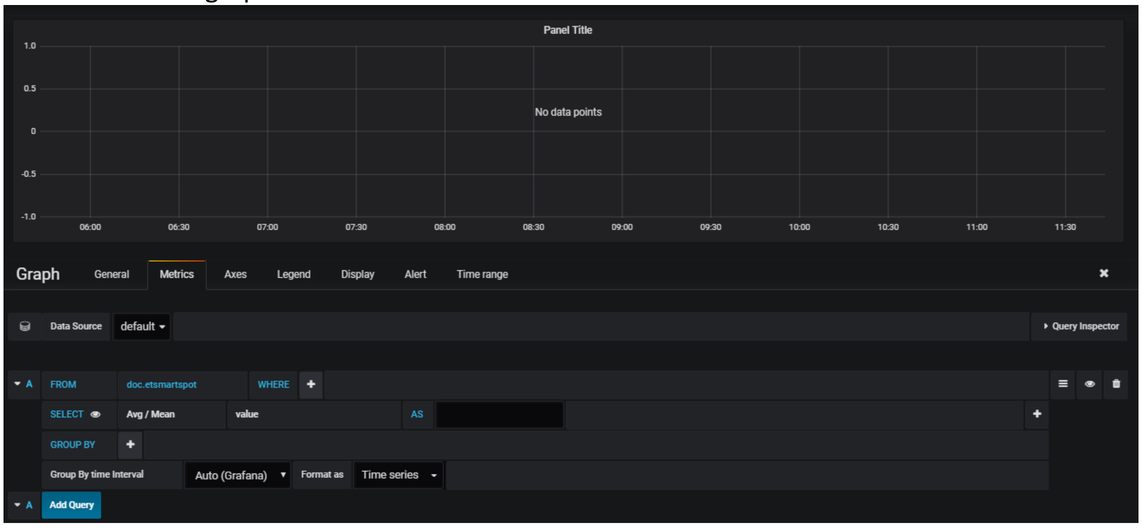
* Grafana configuration to administrative level do it in the file **/etc/grafana/grafana.ini**

3. Create graphs in the dashboard

The first step is to go to create menu and then pulse in new dashboard and we select the option “graph” and will appears the next graph:



We press in “panel title” and different options will appear and we press “edit” option and we will see the next graph:



Data Source: default

FROM doc.etsmartspot WHERE +

SELECT Avg / Mean value AS

GROUP BY +

Group By time Interval: Auto (Grafana) Format as: Time series

The first that we have to do to create the graph is select the source where collect the data, for it in the “Data Source” section we have to press the “default” option and select “Data Source” that we want to use.

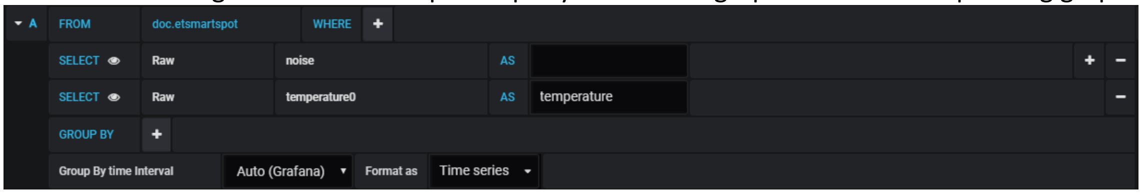
Once the source is selected we have to realise a query to the database so that Grafana creates us a graph with that data. In the next section the Grafana interface provide us a query structure to the database.

To select which attribute to show on the graph, we have to write on the “value” box of the “Select” section the attribute name which we want to look for. If we want to on the graph appears the change of more than one attribute we have to press on “+” symbol which place on the right part of the “Select” section and another “Select” section will be shown us.

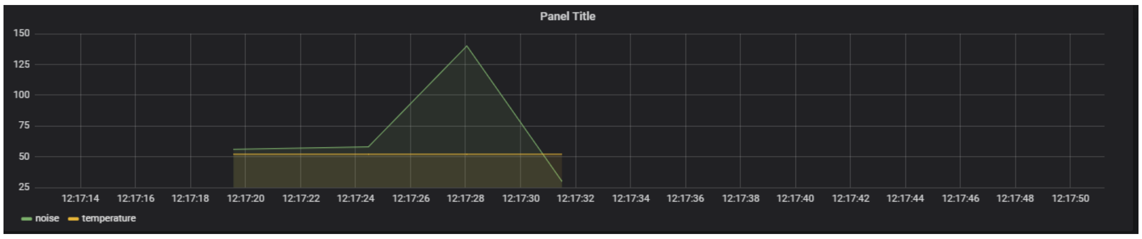
The “Select” section has by default the “Avg/Mean” option. This option returns the data mean of that attribute. This option is useful if we want to know all changes of an attribute over time and there are multiple entities in the database. In the case that only we want to show the change of values of an attribute belonging to an entity, we have to insert in the “Where” section the comparison “id == EntityName” and we have to press on “Avg/Mean” and select the “raw” option so that return us the changes of the attribute underwent over time.

Once written the query we press the “Query Inspector” button and if the query which we have inserted it has been success then Grafana create a graph with the data of the attributes insert in the query, else show us an error and reason for the error.

The next two images show an example of query to create a graph and his corresponding graph.



A	FROM	doc.etsmartspot	WHERE	+	
	SELECT	Raw	noise	AS	
	SELECT	Raw	temperature0	AS	temperature
	GROUP BY	+			
	Group By time Interval	Auto (Grafana)	Format as	Time series	



4. Realise the query to the CrateDB database.

To realise the query to the database we have to execute the “POSTMAN” program and inside “fiware-grafana” collection there is a method called “retrieve grafana database”. This method returns all IDs of the attributes and for each entity the values of their attributes, for it in the method body we define the next query:

```
{
  "stmt": "SELECT * FROM \"doc\".\"etsmartspot\""
}
```

If we want to realise a query more specific to the database we only have to modify the method body and insert the query which we want to realise.