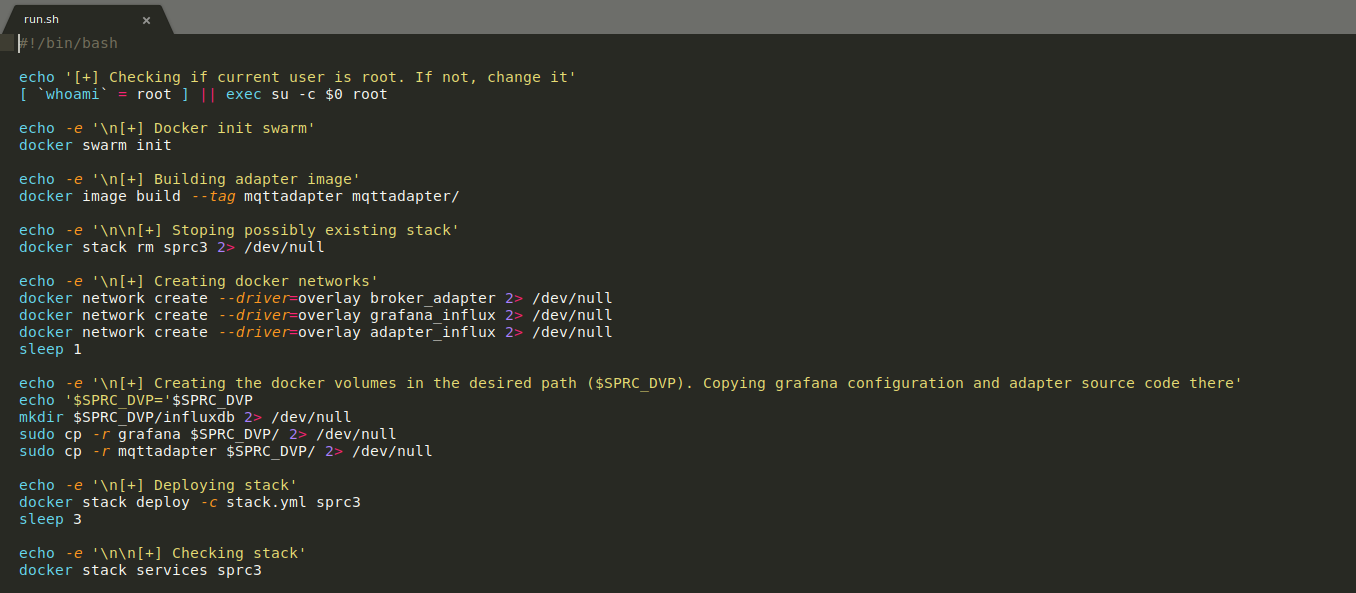
**SPRC Tema3 README**

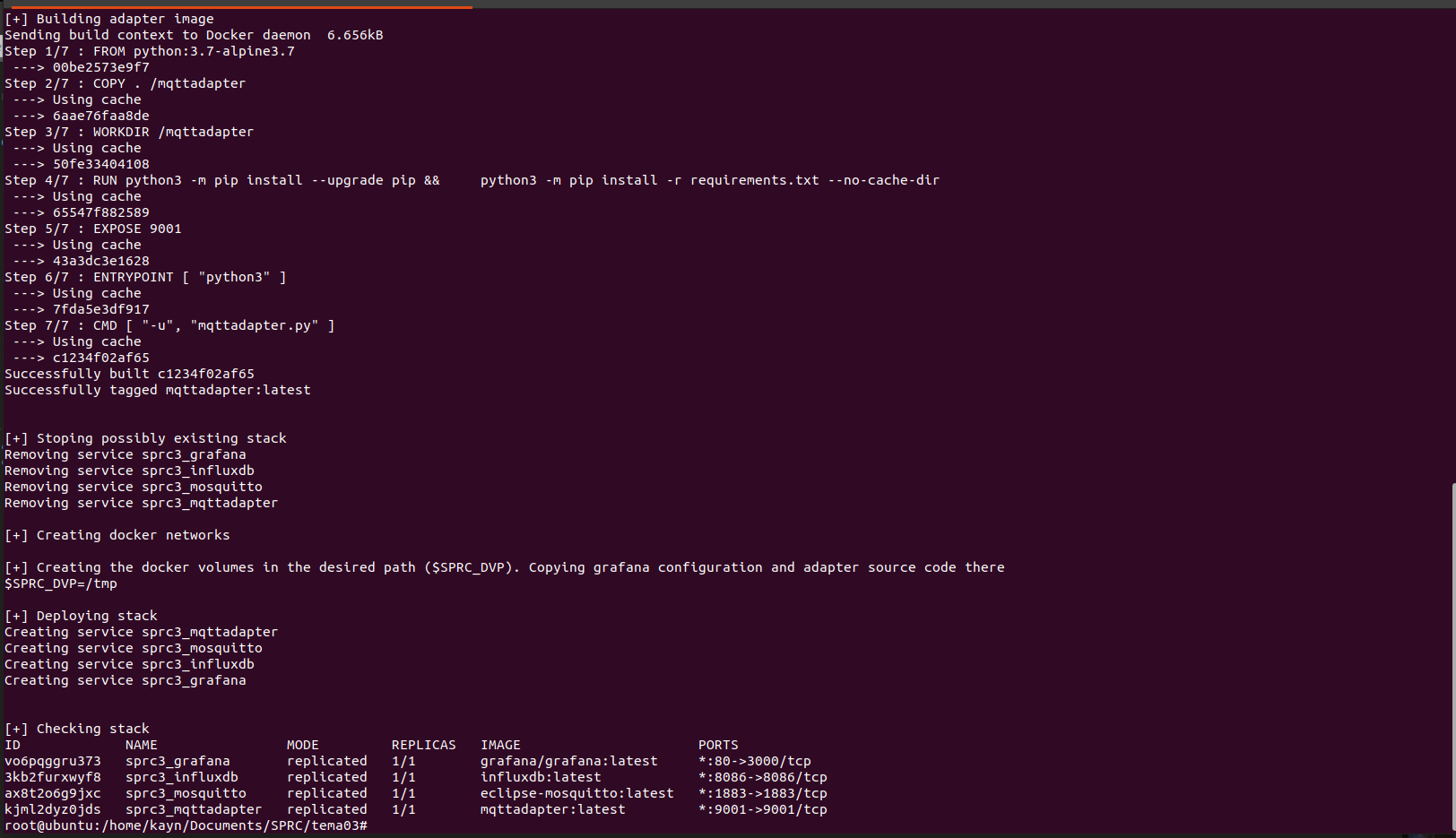
1. **Integrarea în stack a solutiilor open-source & Separarea corectă a traficului între containere**

Asa cum este specificat in enunt, tema a fost translatata intr-un stack de docker. Aceasta arhitectura poate fi pornita folosind scriptul run.sh.



Asa cum putem observa din script, trebui sa initializam docker swarm dupa care sa facem build la imaginea adaptorului nostru de date. De asemenea, tot aici, vom crea retelele de internet aditionale pentru a segrega traficul intre containere(asa cum s-a dorit in enunt). In final, copiem directoarele grafana si mqttadapter in folderul specificat de variabila SPRC\_DVP si rulam stack-ul de docker. Pentru a confirma ca totul s-a realizat cu success, listam serviciile curente si verificam daca la replicas avem valoarea 1.

In stack-ul de docker am specificat ca retelele sunt externe (sunt create de script automat) si dupa le-am distribuit dupa cum se cerea in enunt. Pentru grafana, am setat user-ul si parola folosind variabile de mediu.





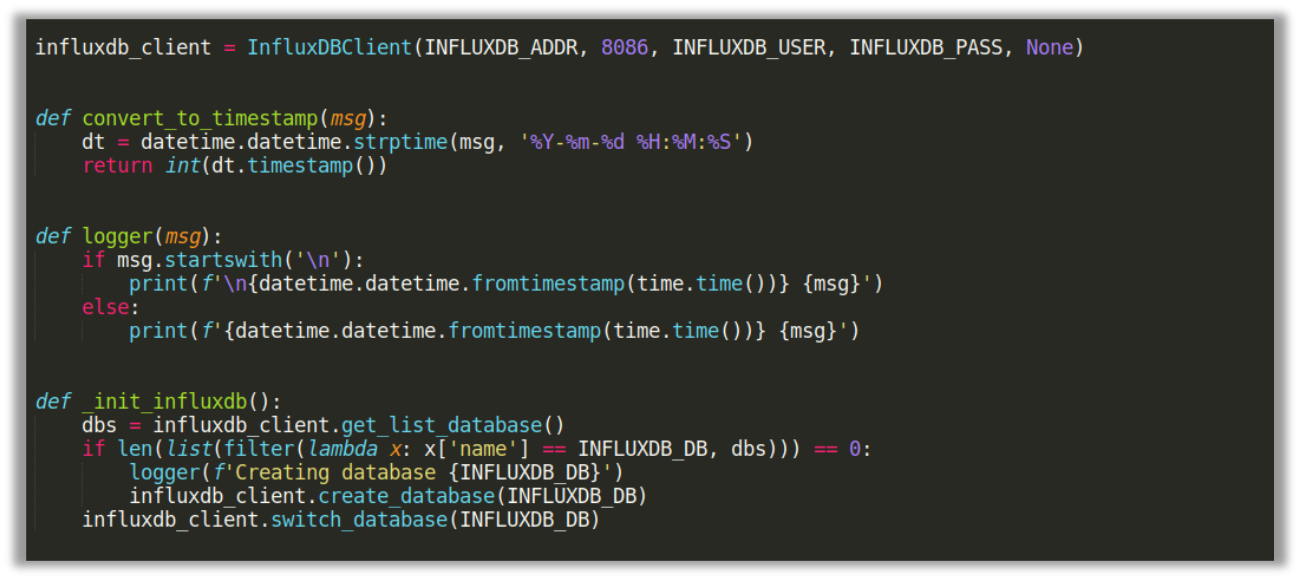
1. **Dezvoltarea adaptorului si integrarea componentelor având ca rezultat stocarea valorilor în baza de date**

Adaptorul a fost construit folosind python si pornind de la scheletul din laboratorul in care am folosit MQTT. In prima faza ne-am conectat la broker-ul de MQTT oferit de docker la care am definit functiile on\_message si on\_connect. La conectare, ne subscriem la toate topicurile folosind ‘#’ iar pe parcurs, odata ce noi mesaje sunt publicate, adaptorul le citeste si parcurge urmatorii pasi:

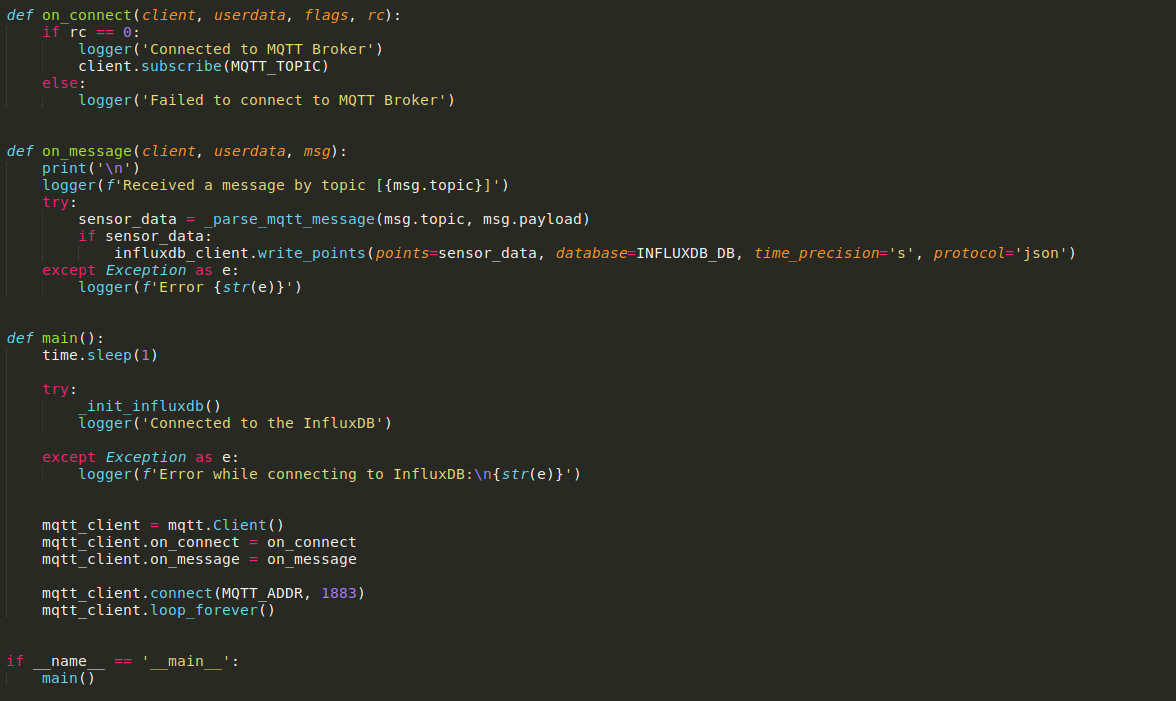
* Se preia locatia, statia si payload-ul in format json din mesaj
* Se converteste timestamp-ul la epochs iar daca nu exista se adauga cel curent (time.time())
* Se creaza o lista de dictionare, fiecare avand 4 chei standarde (measurement, tag, fields, time) in care se introduc datele preluate din mesaj
* Se scrie in baza de date acest vector in format json

Evident, pentru a putea efectua acestea, atunci cand scriptul este rulat, trebuie sa ne conectam la baza de date, sa cream una noua (‘sprc’), si sa schimba baza curenta la ea.

De mentionat este ca la fiecare pas, am afisat la stdout mesaje informationale(logs) pentru a fi usor de determinat posibile erori.



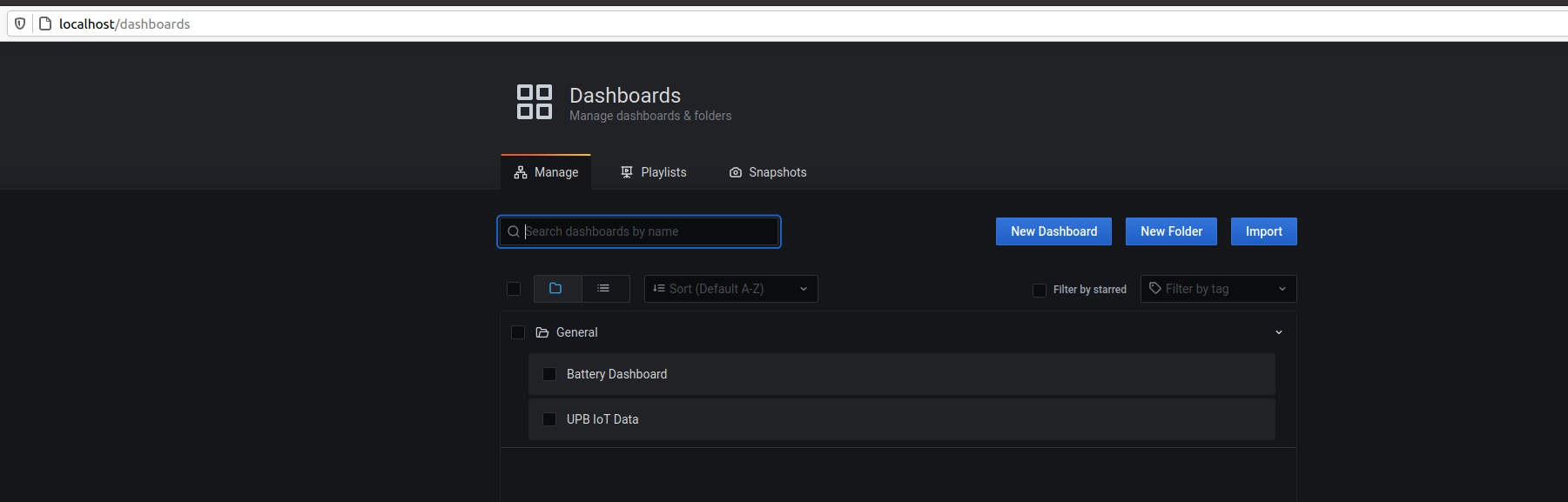


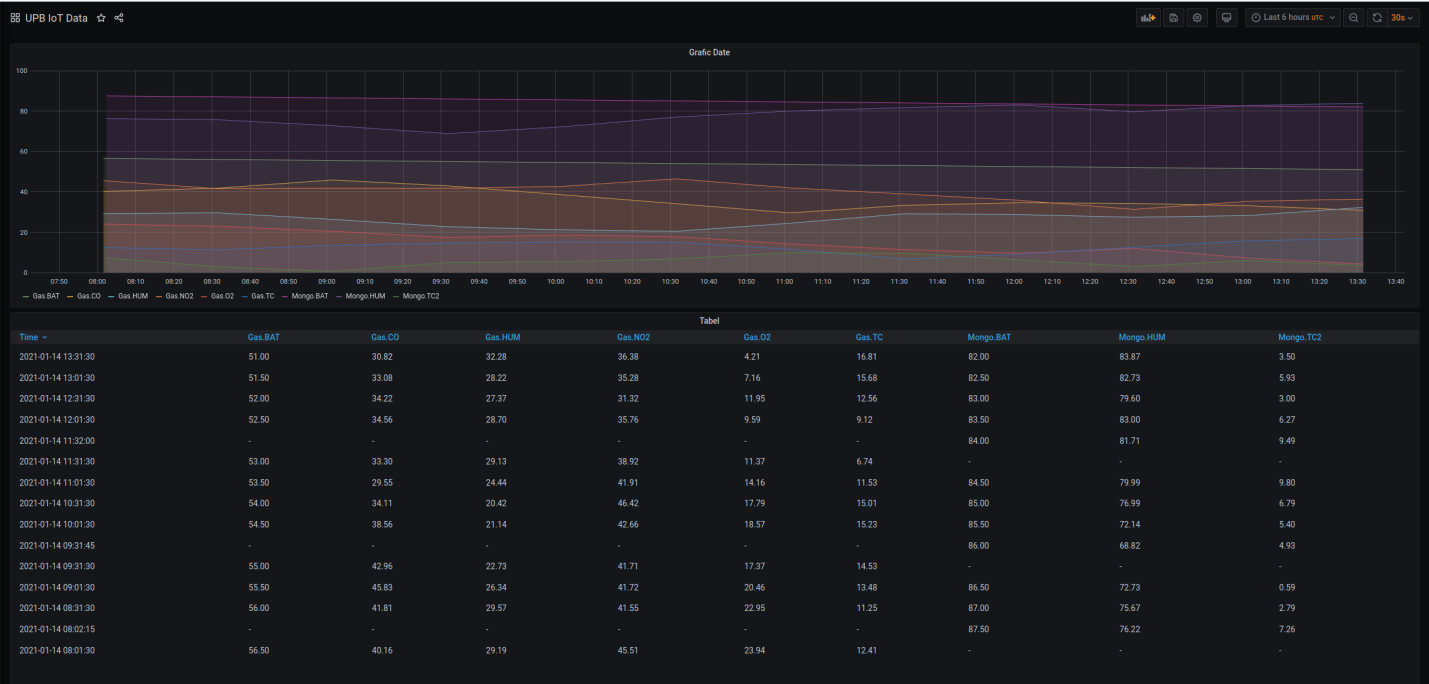


**3. Configurarea interfetei de vizualizare având ca rezultat cel putin un dashboard functional & realizarea corectă a dashboard-urilor default**

Acum ca avem setate toate componentele si am asigurat persistenta prin utilizare de volume, trebuie sa configuram interfata de Grafana pentru a corespunde cu cerintele.

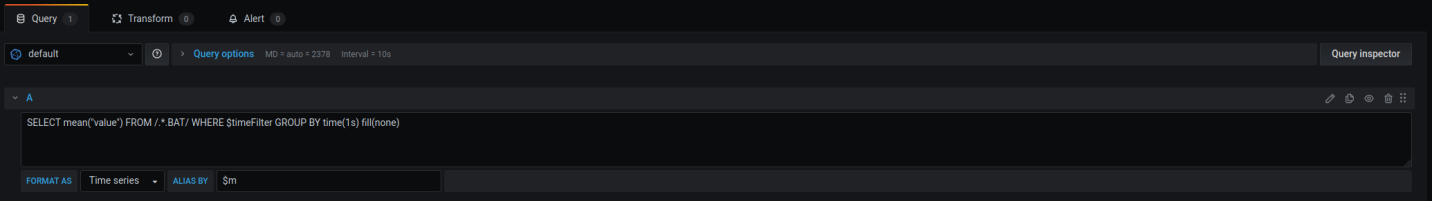
**Primul pas**: Cream 2 dashboarduri, fiecare avand 2 panel-uri, unul cu grafic si unul cu tabel

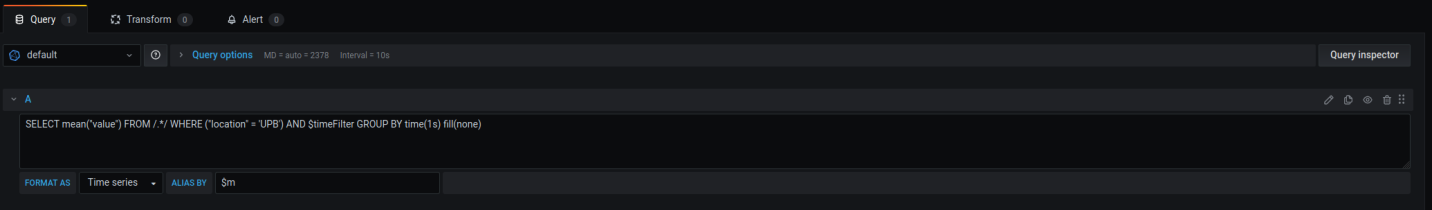




****

**Pasul 2**: Modificam query-urile tabelelor pentru a avea datele in formatul dorit:





Pentru fiecare dashboard, am folosit un regex pentru a selecta doar datele dorite (ori doar .BAT ori toate statiile din UPB). Time(1s) este intervalul de grupare a datelor iar fill(none) a fost folosit pentru a afisa doar datele ne-nule.

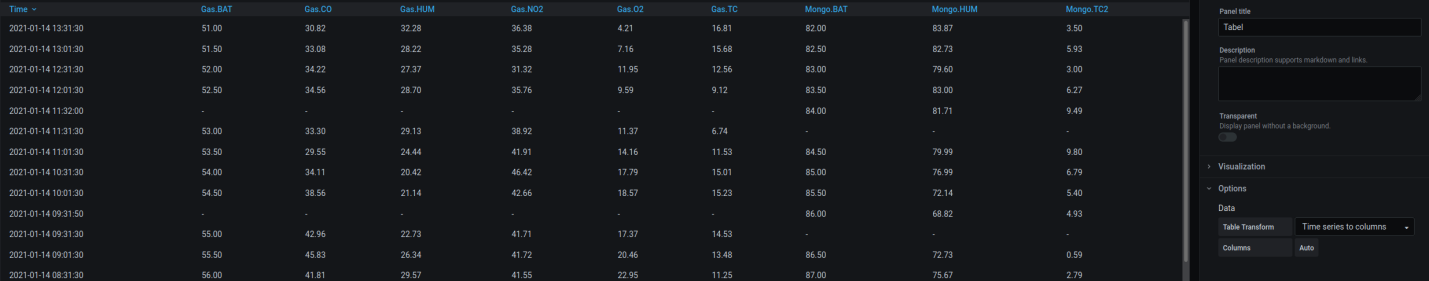
De asemenea, pentru a putea avea formatul numelor seriilor de timp cel dorit, am formatat ca si ‘Time series’ folosind alias-ul $m(measurement).

**Pasul 3**: Setam timezone-ul sa fie UTC, perioada de afisare ultimele 6 ore, dashboard refresh

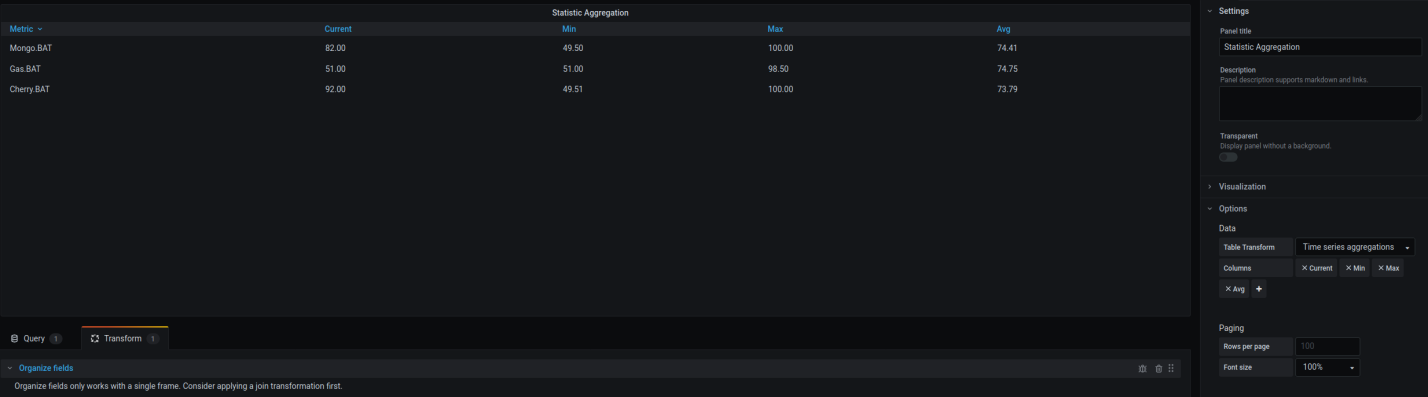




**Pasul 4**: Setam Data Table Transform pentru a avea numele coloanelor ca si measurement(UPB IoT Data). Ne trebuie time series to columns care este asa by default.



**Pasul 5**: Setam Data Table Transform pentru a avea aggregations (Battery Dashboard)



**Mentiuni:**

* De preferat a folosi UTC ca si timezone pentru datele generate
* Pentru testarea adaptorului, in folderul tests, avem un MQTT publisher
* Singurul lucru care nu a iesit a fost la battery dashboard sa separ Metrica de locatie.
* Scriptul se va executa ca si root folosind sh –c run.sh root asa ca, trebuie sa cunoasteti PAROLA de ROOT