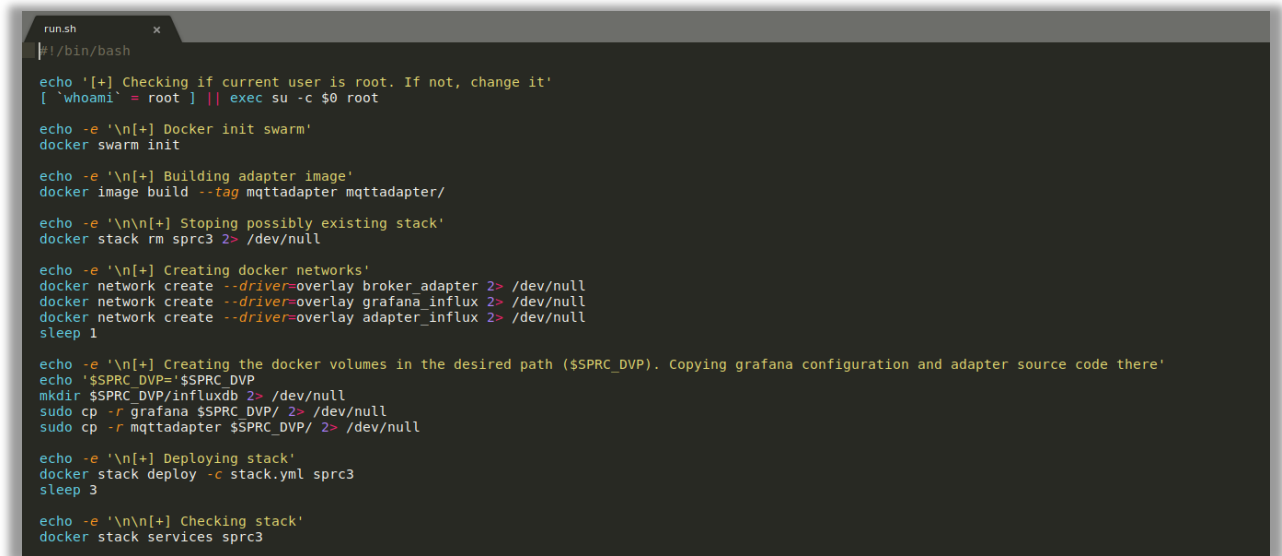


SPRC Tema3 README

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

Asa cum este specificat în enunț, tema a fost translatată într-un stack de docker. Această arhitectură poate fi pornită folosind scriptul run.sh.



```
run.sh x
#!/bin/bash

echo '[+] Checking if current user is root. If not, change it'
[ `whoami` = root ] || exec su -c $0 root

echo -e '\n[+] Docker init swarm'
docker swarm init

echo -e '\n[+] Building adapter image'
docker image build --tag mqttadapter mqttadapter/

echo -e '\n\n[+] Stopping possibly existing stack'
docker stack rm sprc3 2> /dev/null

echo -e '\n[+] Creating docker networks'
docker network create --driver=overlay broker_adapter 2> /dev/null
docker network create --driver=overlay grafana_influx 2> /dev/null
docker network create --driver=overlay adapter_influx 2> /dev/null
sleep 1

echo -e '\n[+] Creating the docker volumes in the desired path ($SPRC_DVP). Copying grafana configuration and adapter source code there'
echo '$SPRC_DVP=$SPRC_DVP'
mkdir $SPRC_DVP/influxdb 2> /dev/null
sudo cp -r grafana $SPRC_DVP/ 2> /dev/null
sudo cp -r mqttadapter $SPRC_DVP/ 2> /dev/null

echo -e '\n[+] Deploying stack'
docker stack deploy -c stack.yml sprc3
sleep 3

echo -e '\n\n[+] Checking stack'
docker stack services sprc3
```

Asa cum putem observa din script, trebuie să initializăm Docker Swarm după care să facem build la imaginea adaptorului nostru de date. De asemenea, tot aici, vom crea rețelele de internet adiționale pentru a segrega traficul între containere (asa cum s-a dorit în enunț). În final, copiem directoarele Grafana și mqttadapter în folderul specificat de variabila SPRC_DVP și rulăm stack-ul de Docker. Pentru a confirma că totul s-a realizat cu succes, listăm serviciile curente și verificăm dacă la replicas avem valoarea 1.

În stack-ul de Docker am specificat că rețelele sunt externe (sunt create de script automat) și după le-am distribuit după cum se cerea în enunț. Pentru Grafana, am setat user-ul și parola folosind variabile de mediu.

```
[+] Building adapter image
Sending build context to Docker daemon 6.656kB
Step 1/7 : FROM python:3.7-alpine3.7
--> 00be2573e9f7
Step 2/7 : COPY . /mqttadapter
--> Using cache
--> 6aae76faa8de
Step 3/7 : WORKDIR /mqttadapter
--> Using cache
--> 50fe33404108
Step 4/7 : RUN python3 -m pip install --upgrade pip && python3 -m pip install -r requirements.txt --no-cache-dir
--> Using cache
--> 65547f802589
Step 5/7 : EXPOSE 9001
--> Using cache
--> 43a3dc3e1028
Step 6/7 : ENTRYPOINT [ "python3" ]
--> Using cache
--> 7fdae53df917
Step 7/7 : CMD [ "-u", "mqttadapter.py" ]
--> Using cache
--> c1234f02af65
Successfully built c1234f02af65
Successfully tagged mqttadapter:latest

[+] Stopping possibly existing stack
Removing service sprc3_grafana
Removing service sprc3_influxdb
Removing service sprc3_mosquitto
Removing service sprc3_mqttadapter

[+] Creating docker networks

[+] Creating the docker volumes in the desired path ($SPRC_DVP). Copying grafana configuration and adapter source code there
$SPRC_DVP=/tmp

[+] Deploying stack
Creating service sprc3_mqttadapter
Creating service sprc3_mosquitto
Creating service sprc3_influxdb
Creating service sprc3_grafana

[+] Checking stack


| ID           | NAME              | MODE       | REPLICAS | IMAGE                    | PORTS            |
|--------------|-------------------|------------|----------|--------------------------|------------------|
| vo6pqggru373 | sprc3_grafana     | replicated | 1/1      | grafana/grafana:latest   | *:80->3000/tcp   |
| 3kb2furxwyf8 | sprc3_influxdb    | replicated | 1/1      | influxdb:latest          | *:8086->8086/tcp |
| ax8t2o6g9jvc | sprc3_mosquitto   | replicated | 1/1      | eclipse-mosquitto:latest | *:1883->1883/tcp |
| kjnl2dyz0jds | sprc3_mqttadapter | replicated | 1/1      | mqttadapter:latest       | *:9001->9001/tcp |


root@ubuntu:/home/Kayn/Documents/SPRC/tena03#
```

```
stack.yml
version: '3'

networks:
  broker_adapter:
    external: True
  grafana_influx:
    external: True
  adapter_influx:
    external: True

services:
  mosquitto:
    image: eclipse-mosquitto
    ports:
      - 1883:1883
    networks:
      - broker_adapter

  influxdb:
    image: influxdb
    ports:
      - 8086:8086
    volumes:
      - ${SPRC_DVP}/influxdb:/var/lib/influxdb
    environment:
      - INFLUXDB_DB=sprc
    networks:
      - grafana_influx
      - adapter_influx

  grafana:
    image: grafana/grafana
    user: "0"
    depends_on:
      - influxdb
    ports:
      - 80:3000
    volumes:
      - ${SPRC_DVP}/grafana:/var/lib/grafana
    environment:
      - GF_SECURITY_ADMIN_USER=asistent
      - GF_SECURITY_ADMIN_PASSWORD=grafanaSPRC2020
    networks:
      - grafana_influx

  mqttadapter:
    image: mqttadapter
    depends_on:
      - mosquitto
      - influxdb
    ports:
      - 9001:9001
    volumes:
      - ${SPRC_DVP}/mqttadapter:/usr/src/mqttadapter/
    networks:
      - broker_adapter
      - adapter_influx
```

2. 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 urmatoorii 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 standard (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 informativale(logs) pentru a fi usor de determinat posibile erori.

```
influxdb_client = InfluxDBClient(INFLUXDB_ADDR, 8086, INFLUXDB_USER, INFLUXDB_PASS, None)

def convert_to_timestamp(msg):
    dt = datetime.datetime.strptime(msg, '%Y-%m-%d %H:%M:%S')
    return int(dt.timestamp())

def logger(msg):
    if msg.startswith('\n'):
        print(f'\n{datetime.datetime.fromtimestamp(time.time())} {msg}')
    else:
        print(f'{datetime.datetime.fromtimestamp(time.time())} {msg}')

def _init_influxdb():
    dbs = influxdb_client.get_list_database()
    if len(list(filter(lambda x: x['name'] == INFLUXDB_DB, dbs))) == 0:
        logger(f'Creating database {INFLUXDB_DB}')
        influxdb_client.create_database(INFLUXDB_DB)
    influxdb_client.switch_database(INFLUXDB_DB)
```

```

def _parse_mqtt_message(topic, payload):
    data = json.loads(payload.decode('utf-8'))
    location, station = topic.split('/')

    if "timestamp" in data.keys():
        ttime = parse(data['timestamp'])
        logger(f'Data timestamp is: {time}')
    else:
        ttime = int(time.time())
        logger(f'Data timestamp is: {datetime.datetime.fromtimestamp(int(ttime))}')

    json_body = [{ 'measurement': f'{station}.{key}',
                    'tags': {
                        'location': location
                    },
                    'fields': {
                        'value': value
                    },
                    'time': ttime
                  }
                for key, value in data.items() if type(value) == int or type(value) == float]

    for key, value in data.items():
        if type(value) == int or type(value) == float and key != 'timestamp':
            logger(f'{location}.{station}.{key} {value}')

    return json_body

```

```

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        logger('Connected to MQTT Broker')
        client.subscribe(MQTT_TOPIC)
    else:
        logger('Failed to connect to MQTT Broker')

def on_message(client, userdata, msg):
    print('\n')
    logger(f'Received a message by topic [{msg.topic}]')
    try:
        sensor_data = _parse_mqtt_message(msg.topic, msg.payload)
        if sensor_data:
            influxdb_client.write_points(points=sensor_data, database=INFLUXDB_DB, time_precision='s', protocol='json')
    except Exception as e:
        logger(f'Error {str(e)}')

def main():
    time.sleep(1)

    try:
        _init_influxdb()
        logger('Connected to the InfluxDB')
    except Exception as e:
        logger(f'Error while connecting to InfluxDB:\n{str(e)}')

    mqtt_client = mqtt.Client()
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    mqtt_client.connect(MQTT_ADDR, 1883)
    mqtt_client.loop_forever()

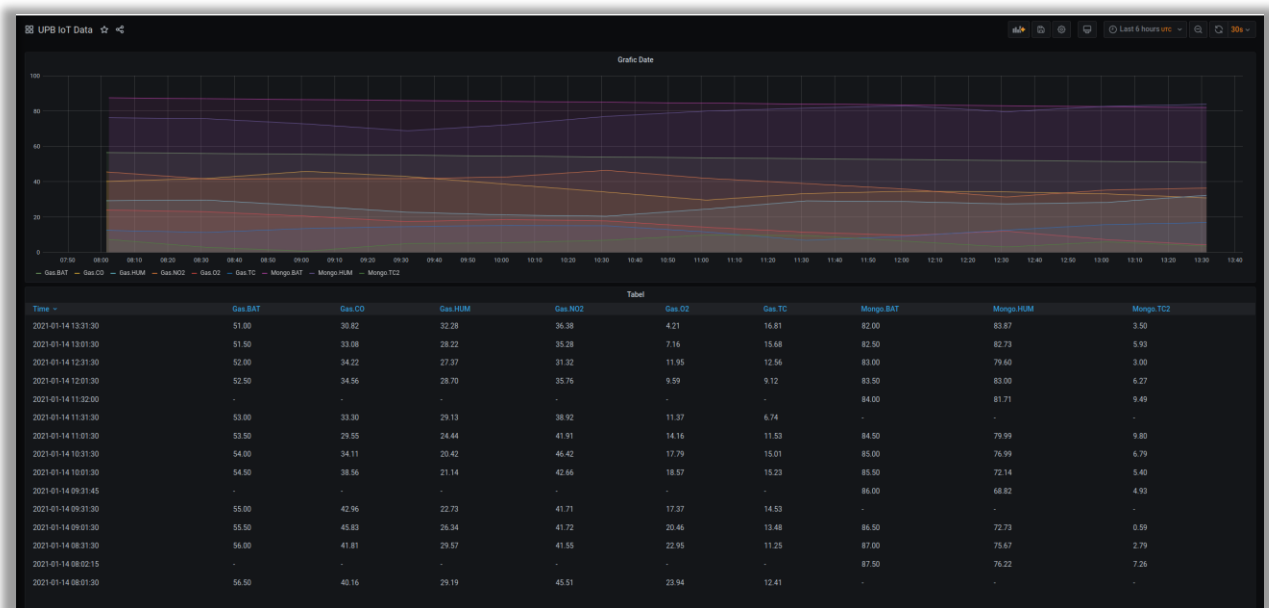
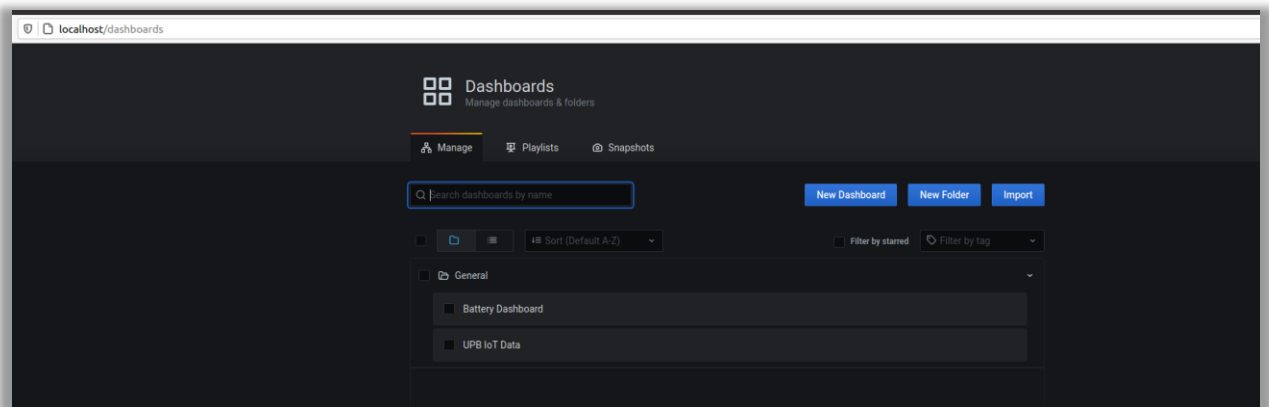
if __name__ == '__main__':
    main()

```

3. Configurarea interfeței de vizualizare având ca rezultat cel puțin 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 tabelor pentru a avea datele in formatul dorit:

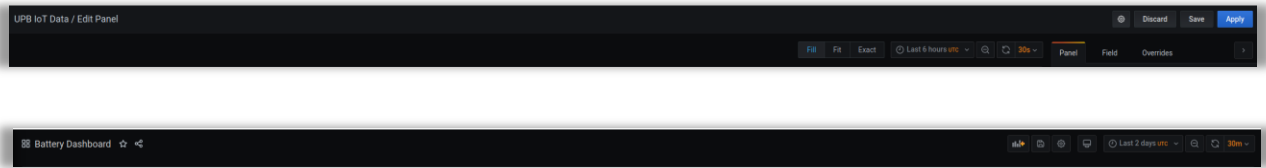
```
Query options: MD + auto + 2378 Interval + 10s
SELECT mean("value") FROM //^BAT/ WHERE $timeFilter GROUP BY time(1s) fill(none)
```

```
Query options: MD + auto + 2378 Interval + 10s
SELECT mean("value") FROM //^/ WHERE ("location" = 'UPB') AND $timeFilter GROUP BY time(1s) fill(none)
```

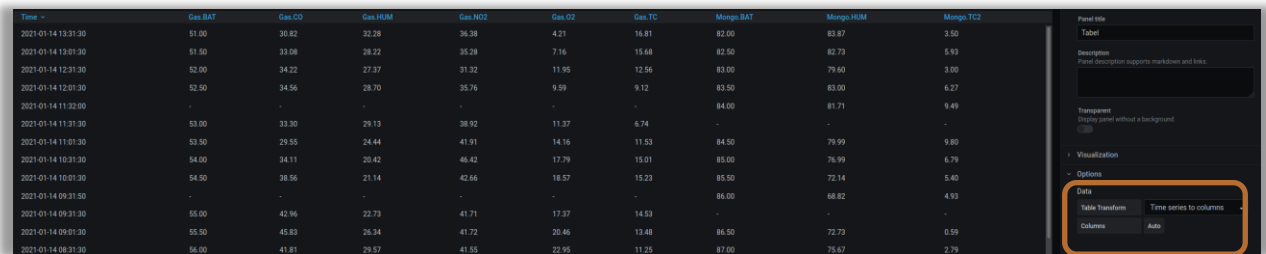
Pentru fiecare dashboard, am folosit un regex pentru a selecta doar datele dorite (ori doar .BAT ori toate statiile din UPB). Time(1s) este intervalul de grupare a datelor iar fill(none) a fost folosit pentru a afisa doar datele ne-nule.

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

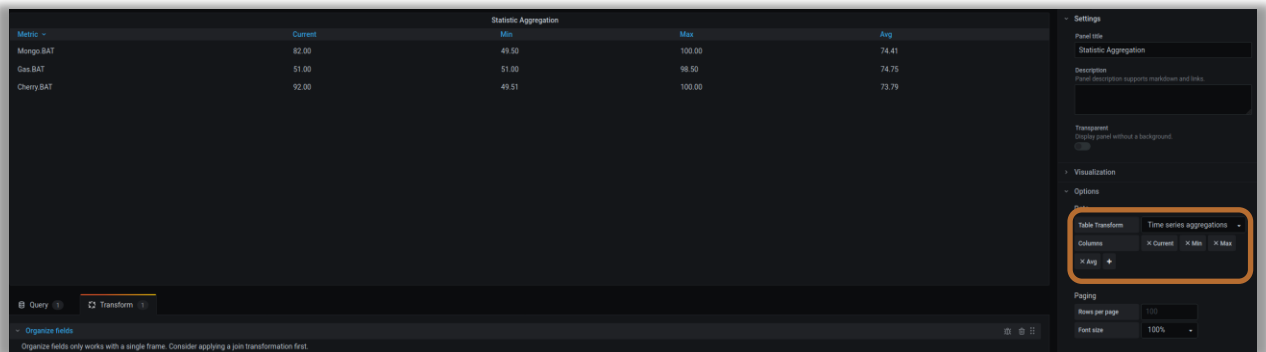
Pasul 3: Setam timezone-ul sa fie UTC, perioada de afisare ultimele 6 ore, dashboard refresh



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