

Лабораторная работа 2. Практика по Docker

Выполнила студент гр. 5142704/30801 Порфирьева Е. В.

Шаг 1: Разработка Simulator для генерации данных

В файле sensor.py создаем четвертый тип датчиков CO, назначение которого измерять концентрацию угарного газа.

```
class CO(Sensor):  
    step = 0  
  
    def __init__(self, name):  
        super().__init__(name)  
        self.type = "carbon oxid"  
  
    def generate_new_value(self):  
        self.value = self.step * 1e6  
        self.step = self.step + 0.001
```

Далее в файле main.py реализуем клиента, который подключается к mqtt брокеру и публикует сообщения.

```
import paho.mqtt.client as paho  
from os import environ  
import time  
  
from entity.sensor import *  
  
broker = "localhost" if "SIM_HOST" not in environ.keys() else  
environ["SIM_HOST"]  
port = 1883 if "SIM_PORT" not in environ.keys() else environ["SIM_PORT"]  
name = "sensor" if "SIM_NAME" not in environ.keys() else  
environ["SIM_NAME"]  
period = 1 if "SIM_PERIOD" not in environ.keys() else  
int(environ["SIM_PERIOD"])
```

```
type_sim = "temperature" if "SIM_TYPE" not in environ.keys() else
environ["SIM_TYPE"]
```

```
sensors = {"temperature": Temperature, "pressure": Pressure, "current": Current,
"carbon_oxid": CO}
```

```
def on_publish(client, userdata, result): # create function for callback
    print(f"data published {userdata}")
    pass
```

```
sensor = sensors[type_sim](name=name)
client1 = paho.Client(sensor.name) # create client object
client1.on_publish = on_publish # assign function to callback
client1.connect(broker, port) # establish connection
while True:
    sensor.generate_new_value()
    ret = client1.publish("sensors/" + sensor.type + "/" + sensor.name,
sensor.get_data()) # publish
    time.sleep(period)
```

В Dockerfile, необходимом для того, что создать образ, указываем следующие инструкции:

```
FROM python:alpine3.19
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
RUN pip install -r requirements.txt
```

```
COPY . .
```

```
CMD ["python", "main.py"]
```

Инструкция FROM инициализирует новый этап сборки и устанавливает базовый образ для последующих инструкций. WORKDIR создает рабочий каталог для последующих инструкций Dockerfile. Инструкция COPY копирует файл requirements.txt из источника в указанное место внутри образа. Инструкция RUN задает команды, которые следует выполнить и поместить в новый образ контейнера. RUN описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен.

Содержимое файла requirements.txt приведено ниже: `raho_mqtt==1.6.1`

`raho_mqtt` предоставляет клиентский класс, который позволяет приложениям подключаться к MQTT-брокеру для публикации сообщений.

После сделанных выше операций можно создать образ командой:

```
docker build -t lenaporfireva/data-simulator .
```

Таким образом мы создали образ, из которого можно развернуть контейнер.

```
C:\Users\Lena\OneDrive\Рабочий стол\DockerPractice\vm\client\simulator>docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
lenaporfireva/data-simulator  latest     ab14d6ff1d65  2 minutes ago 94.7MB
```

Шаг 2: Запуск Mosquitto брокера

Для настройки протокола MQTT необходимо создать конфигурационный файл `mosquitto.conf` со следующим содержимым:

```
listener 1883
```

```
allow_anonymous true
```

Для более удобного запуска брокера создадим файл `docker-compose.yml` со следующим содержимым:

```
version: "3"
```

```
services:
```

```
  broker:
```

```
    image: eclipse-mosquitto
```

```
    container_name: broker
```

```
    volumes:
```

```
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
```

```
    ports:
```

```
      - "1883:1883"
```

Теперь создав контейнер (из Шага 1), получим следующее:

```
C:\Users\Lena\OneDrive\Рабочий стол\DockerPractice\vms\client\simulator>docker run -e SIM_HOST=192.168.0.101 -e SIM_TYPE=temperature --name test -d lenaporfireva/data-simulator
ac35634084c21544491668f39ee33c6dcd348366be536119cb54a596ec627cfa
```

Брокер отображает присоединившегося клиента:

```
C:\Users\Lena\OneDrive\Рабочий стол\DockerPractice\vms\gateway>docker compose up
time="2024-11-13T22:06:10+03:00" level=warning msg="C:\\Users\\Lena\\OneDrive\\Рабочий стол\\DockerPractice\\vms\\gateway\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 4/4
  █ broker Pulled                                18.7s
    █ da74935a2317 Download complete              0.5s
    █ da9db072f522 Download complete              7.6s
    █ de46c2114aca Download complete             12.6s
[+] Running 2/2
  █ Network gateway_default Created              0.1s
  █ Container broker Created                    0.5s
Attaching to broker
broker | 1731524790: mosquitto version 2.0.20 starting
broker | 1731524790: Config loaded from /mosquitto/config/mosquitto.conf.
broker | 1731524790: Opening ipv4 listen socket on port 1883.
broker | 1731524790: Opening ipv6 listen socket on port 1883.
broker | 1731524790: mosquitto version 2.0.20 running
broker | 1731524812: mosquitto version 2.0.20 terminating
```

Теперь можно запустить несколько датчиков. Но перед этим необходимо прописать *docker-compose.yml*:

version: "3"

services:

temp_sensor:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=TEMP1
- SIM_PERIOD=2
- SIM_TYPE=temperature

pressure_sensor:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=PRESS1
- SIM_PERIOD=2
- SIM_TYPE=pressure

current_sensor:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=CURRENT1
- SIM_PERIOD=2
- SIM_TYPE=current

co_sensor:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=CO1
- SIM_PERIOD=2
- SIM_TYPE=carbon_oxid

temp_sensor_2:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=TEMP2
- SIM_PERIOD=4
- SIM_TYPE=temperature

co_sensor_2:

image: lenaporfireva/data-simulator

environment:

- SIM_HOST=192.168.0.101
- SIM_NAME=CO2
- SIM_PERIOD=6
- SIM_TYPE=carbon_oxid

```

C:\Users\Lena\OneDrive\Рабочий стол\DockerPractice\vms\client\simulator>docker compose up
time="2024-11-13T22:07:16+03:00" level=warning msg="C:\\Users\\Lena\\OneDrive\\Рабочий стол\\DockerPractice\\vms\\client\\simulator\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 7/7
  ⬢ Network simulator_default          Created                                0.1s
  ⬢ Container simulator-temp_sensor_2-1 Created                                0.9s
  ⬢ Container simulator-co_sensor_2-1   Created                                0.9s
  ⬢ Container simulator-temp_sensor-1   Created                                0.7s
  ⬢ Container simulator-current_sensor-1 Created                                1.0s
  ⬢ Container simulator-pressure_sensor-1 Created                                0.7s
  ⬢ Container simulator-co_sensor-1     Created                                0.9s
Attaching to co_sensor-1, co_sensor_2-1, current_sensor-1, pressure_sensor-1, temp_sensor-1, temp_sensor_2-1

```

Шаг 3: Получение данных от симулятора

Первоначально необходимо настроить Telegraf, который подписывается на MQTT, где датчики публикуют данные. Данные будут сохраняться в InfluxDB. Отображение информации с датчиков будет происходить при помощи Grafana.

Telegraf

Перед использованием Telegram необходимо настроить его. Для этого откроем конфигурационный файл telegraf.conf и в разделе `'[[inputs.mqtt_consumer]]'` пропишем следующее:

```
servers = ["tcp://192.168.0.101:1883"] # адрес vm с mqtt-брокером
```

```
topics = [
    "sensors/#"
]
```

```
data_format = "value"
```

```
data_type = "float"
```

Тем самым мы настраиваем Telegraf на чтение данных с машины IP адрес которой 192.168.0.101 через порт 1883.

Также настроим раздел вывода `'[[outputs.influxdb]]'`:

```
urls = ["http://influxdb:8086"]
```

```
database = "sensors"
```

```
skip_database_creation = true
```

```
username = "telegraf"
```

```
password = "telegraf"
```

Настройка Telegraf на этом завершена.

InfluxDB

Для создания базы данных необходимо в конфигурационном файле influxdb-init.iql прописать следующее:

```
CREATE database sensors
```

```
CREATE USER telegraf WITH PASSWORD 'telegraf' WITH ALL PRIVILEGES
```

Grafana

Данные для отображения датчиков берутся из InfluxDB.

Для настройки в конфигурационном файле необходимо прописать следующее:

```
apiVersion: 1
```

```
datasources:
```

```
- name: InfluxDB_v1
```

```
  type: influxdb
```

```
  access: proxy
```

```
  database: sensors
```

```
  user: telegraf
```

```
  url: http://influxdb:8086
```

```
jsonData:
```

```
  httpMode: GET
```

```
secureJsonData:
```

```
  password: telegraf
```

Для запуска всех трех контейнеров воспользуемся docker-compose.

В docker-compose.yml пропишем следующее:

```
version: "3"
```

```
services:
```

```
  influxdb:
```

```
    image: influxdb:1.8
```

```
    container_name: influxdb
```

```
    volumes:
```

- ./influxdb/scripts:/docker-entrypoint-initdb.d
- influx_data:/var/lib/influxdb

networks:

- server-net

telegraf:

image: telegraf

container_name: telegraf

volumes:

- ./telegraf:/etc/telegraf:ro

restart: unless-stopped

networks:

- server-net

grafana:

image: grafana/grafana

container_name: grafana

volumes:

- grafana_data:/var/lib/grafana
- ./grafana:/etc/grafana/

environment:

- GF_SECURITY_ADMIN_USER=admin
- GF_SECURITY_ADMIN_PASSWORD=admin
- GF_USERS_ALLOW_SIGN_UP=false

restart: unless-stopped

ports:

- 3000:3000

networks:

- server-net

volumes:

```
influx_data: {}
```

```
grafana_data: {}
```

networks:

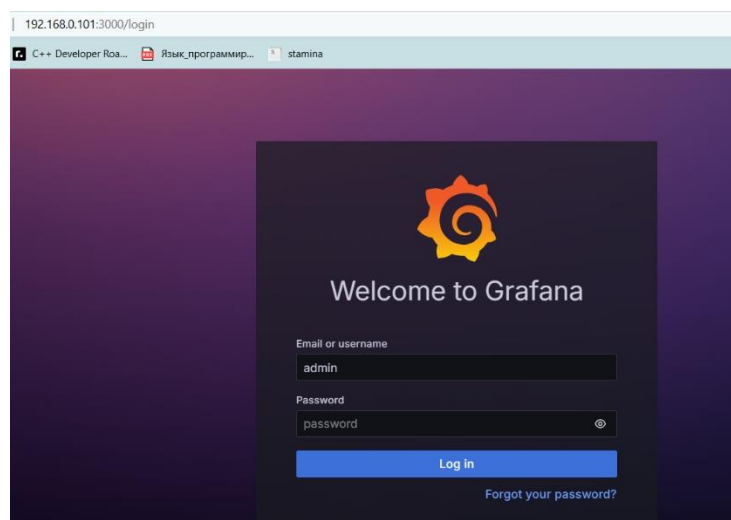
```
server-net: {}
```

После чего можно выполнить команду ‘docker compose up’

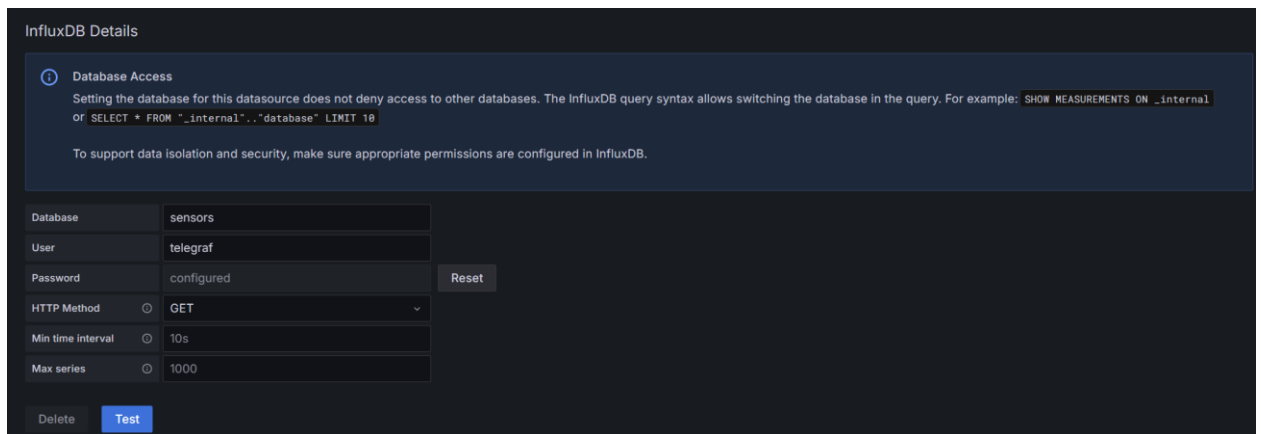
```
C:\Users\Lena\OneDrive\Рабочий стол\DockerPractice\vms\server\infra\telegraf>docker compose up
time="2024-11-13T22:08:17+03:00" level=warning msg="C:\\Users\\Lena\\OneDrive\\Рабочий стол\\DockerPractice\\vms\\server\\infra\\telegraf\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 25/25
  grafana Pulled
    85e1a6c1a7a0 Download complete 509.2s
    dc83b6e9d3ec Download complete 130.0s
    8d531a5572e1 Download complete 48.3s
    e77c72ff0af5 Download complete 489.7s
    d3f38b84c073 Download complete 496.3s
    51d43ac2fea5 Download complete 13.1s
    b303c3332c22 Download complete 25.2s
    0d17397518b9 Download complete 15.7s
    ae00aebb7f03 Download complete 19.1s
  influxdb Pulled
    d00843a9c58b Download complete 14.2s
    734d2edced1b Download complete 535.2s
    7e6ef86bdb80 Download complete 0.7s
    9439c0e98ae5f Download complete 2.4s
    99c539c6f532 Download complete 7.3s
    3f0a8853fbab Download complete 510.8s
    65b9c8d60da9 Download complete 231.4s
  telegraf Pulled
    c3cc7b6f0473 Download complete 525.8s
    aa9983172de0 Download complete 9.4s
    6cec5ef2e84f Download complete 556.5s
    b2b31b28ee3c Download complete 229.5s
    23f2e2047df7 Download complete 6.2s
    f1cef6be5300 Download complete 0.9s
[+] Running 6/6
  Network telegraf_server-net Created 0.1s
  Volume "telegraf_influx_data" Created 0.0s
  Volume "telegraf_grafana_data" Created 0.0s
  Container influxdb Created 1.6s
  Container grafana Created 1.6s
  Container telegraf Created 1.6s
Attaching to grafana, influxdb, telegraf
```

Настройка дашборда

После запуска контейнеров, в браузере переходим по 192.168.0.101:3000



Для проверки соединения перейдем в Menu -> Connections -> Data sources. Находим в источниках InfluxDB и проверяем подключение:



InfluxDB Details

Database Access

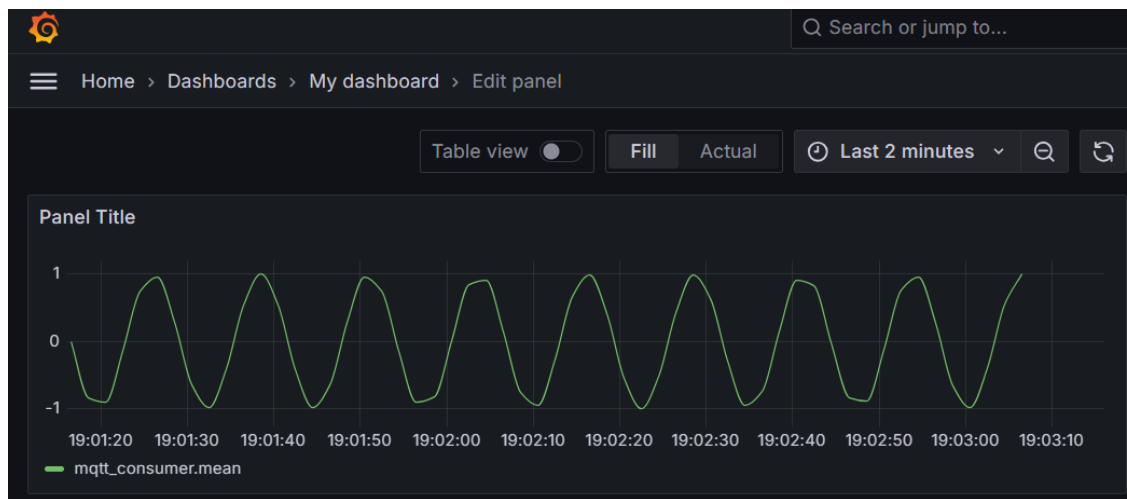
Setting the database for this datasource does not deny access to other databases. The InfluxDB query syntax allows switching the database in the query. For example: `SHOW MEASUREMENTS ON _internal` or `SELECT * FROM "_internal"."database" LIMIT 10`

To support data isolation and security, make sure appropriate permissions are configured in InfluxDB.

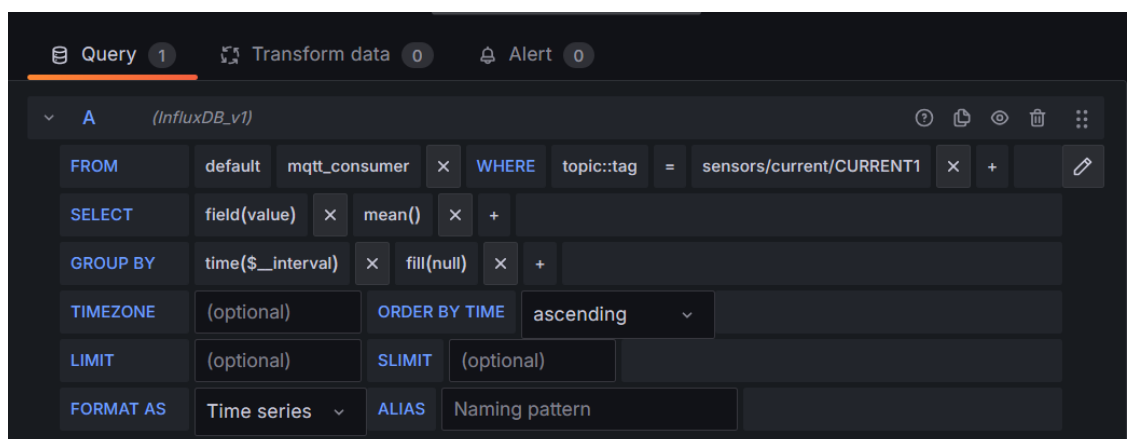
Database	sensors	
User	telegraf	
Password	configured	Reset
HTTP Method	GET	
Min time interval	10s	
Max series	1000	

Delete Test

После переходим через Меню в раздел Dashboards, где создаем собственный дашборд.



Для отображения информации с датчиков необходимо создать запрос (query).



Query 1 Transform data 0 Alert 0

A (InfluxDB_v1)

FROM	default	mqtt_consumer	×	WHERE	topic::tag	=	sensors/current/CURRENT1	×	+	
SELECT	field(value)	×	mean()	×	+					
GROUP BY	time(\$__interval)	×	fill(null)	×	+					
TIMEZONE	(optional)			ORDER BY TIME	ascending					
LIMIT	(optional)			SLIMIT	(optional)					
FORMAT AS	Time series			ALIAS	Naming pattern					

После создания необходимо количества графиков, отображающих информацию с Simulator, экспортируем дашборд как JSON-файл.

Для этого находим функцию Share -> ``Export->Save to file`. Сохраненный файл помещаем в папку
`vms\server\infra\grafana\provisioning\dashboards\mqtt.json`

Пример выполненной работы:

