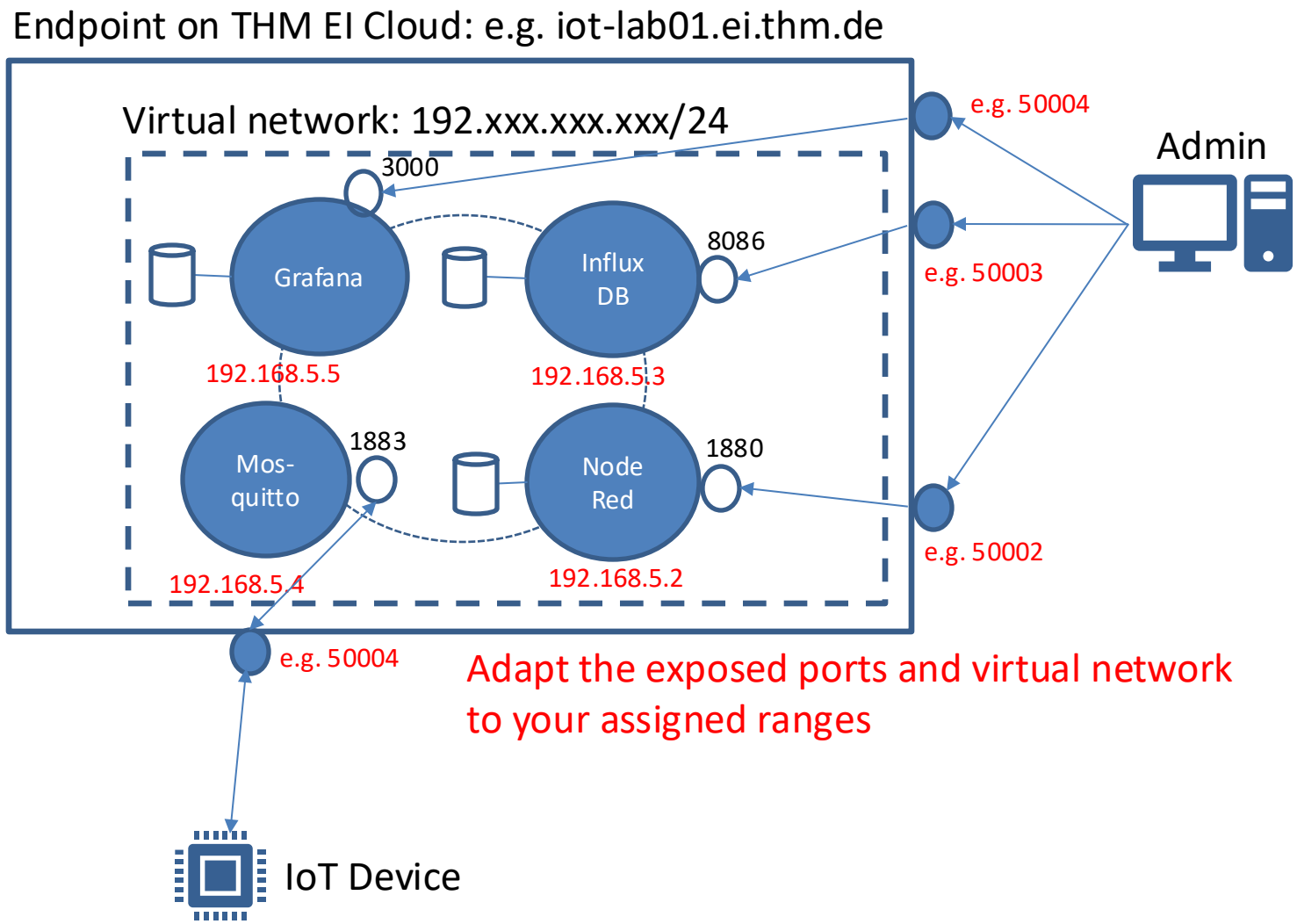


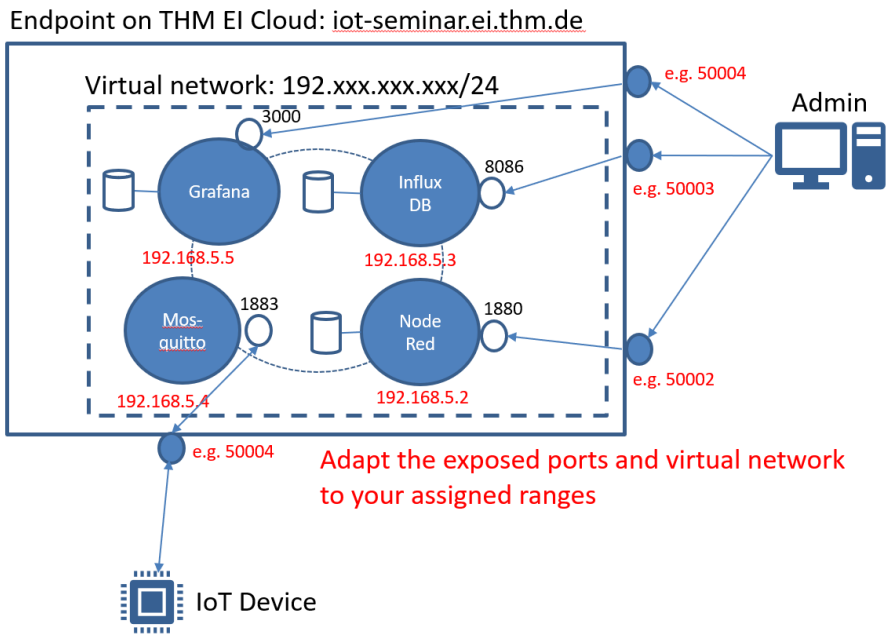
IoT Seminar Lab 2:

Deploying Docker Container Apps on the THM EI Cloud

Overview and Targets

- Register on the EI Cloud Platform using OAuth
- Login to get access to the endpoint which has been assigned to you
- Deploy and configure stand-alone container applications:
  - **NodeRed** with a local volume attached for persistent data
  - **Mosquitto Broker** (no local volume is necessary)
  - **InfluxDB** with a local volume attached for persistent data
  - **Grafana** with a local volume attached for persistent data
- Send **CPU load** via MQTT, store in InfluxDB and visualize with **Grafana** either from local NodeRed instance or external IoT Device
- Subscribe to the topics in different ways using NodeRed, MQTT Explorer & Mosquitto Client
- Learn how to secure your MQTT connection via TLS and how to use the secured MQTT Broker of the EI-Cloud in the future

- To perform this task, there are three options
  - a.) Portainer GUI
  - b.) Docker Command Line Interface ✅ ***(already done in Lab 1)***
  - c.) Container Orchestration: Docker Swarm

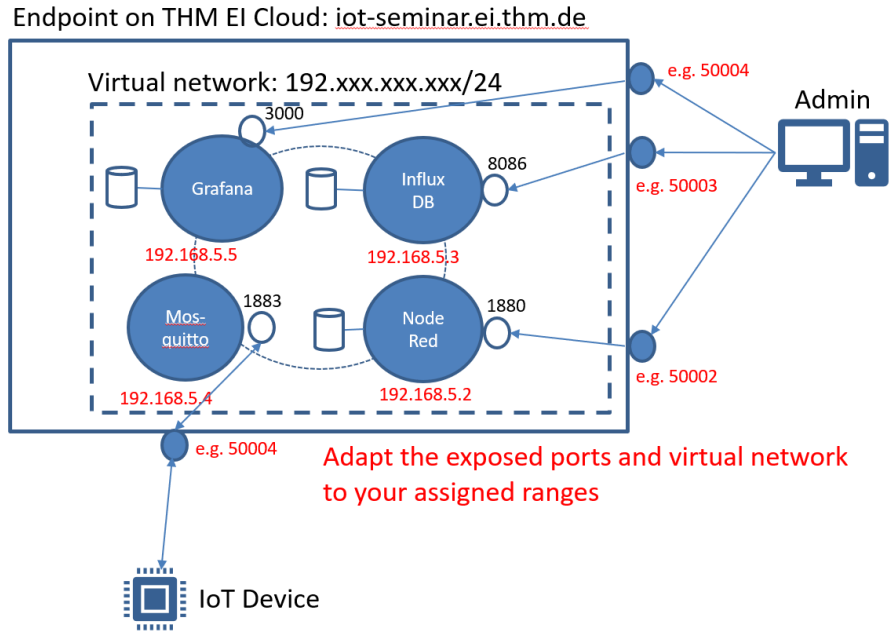


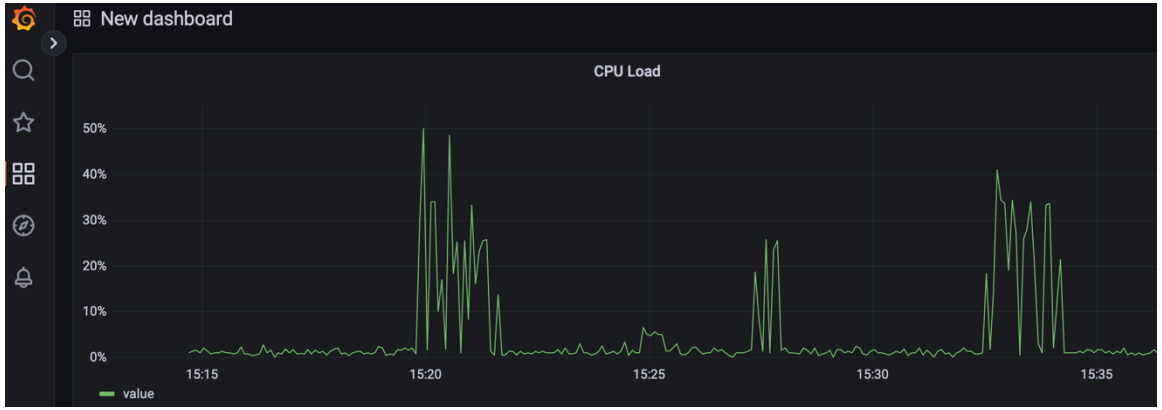
Steps in this Lab

- A.) Portainer GUI
  - 1.) Deploy NodeRed Container
  - 2.) Deploy Mosquitto Container
  - 3.) Configure and Test NodeRed and MQTT
  - 4.) Depoy InfluxDB Container
  - 5.) Configure InfluxDB (also in NodeRed)
  - 6.) Deploy Grafana Container
  - 7.) Configure Grafana and Display CPU Dashboard
  - 8.) Send CPU Load from external device via published Brokerport
  - 9.) Using MQTT Explorer and Container-Names

- B.) Use Command Line Interface (CLI)
  ✅ ***(already done in Lab 1)***

- C.) Use a YAML File (Docker compose)

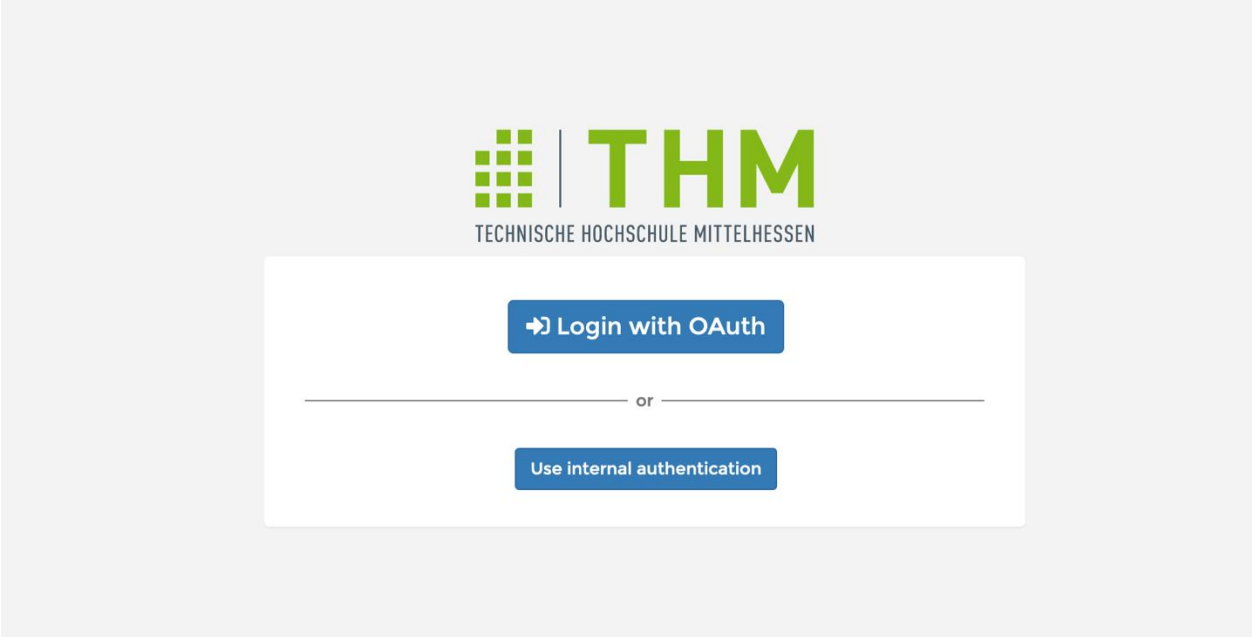




Register and login to THM EI Cloud

- Navigate to https://cloud.ei.thm.de*
- Follow Login with OAuth Protocol and register
- Follow Email Verification link and login with username or email
- If already configured you have access to your endpoint

- * Note: If you are not in THM network, connect via VPN client https://www.thm.de/its/campusnetz/vpn.html

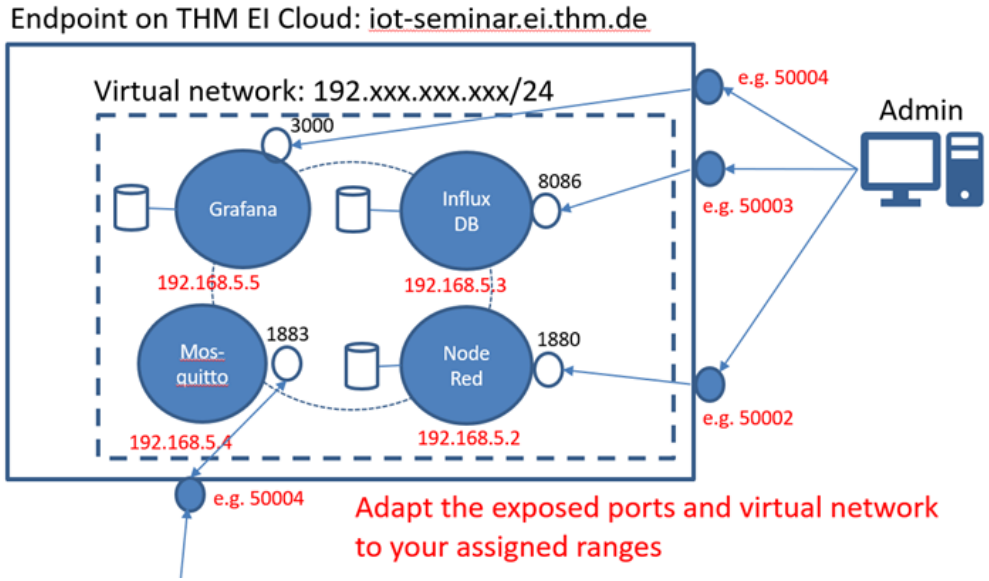


A.) Use Portainer GUI

- Perform the steps above using
  - A.) Portainer GUI
  - B.) Docker Command Line Interface (***done***)
  - C.) Docker manifest file: YAML

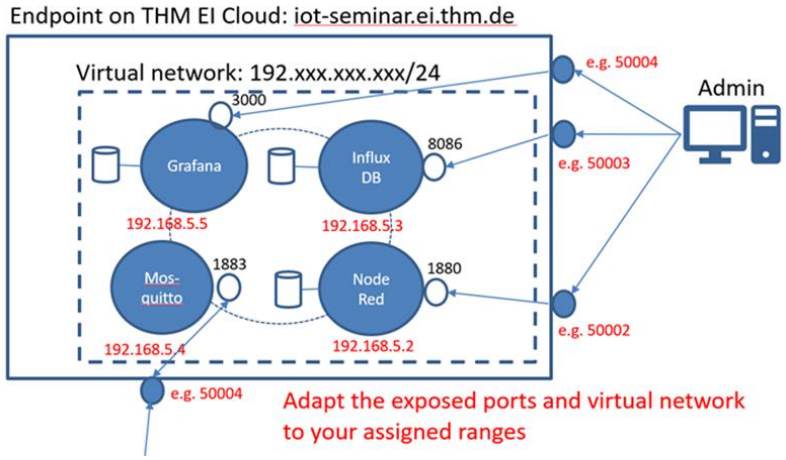
NOTE: 4
- Please use and note down the IP Address and Port range assigned to you in moodle / lecture and replace IPs and Ports in this manual with your assigned ranges!!!
- name networks and volumes unique:
e.g. **<yourname>_net** and **<yourname>_nodered_vol**

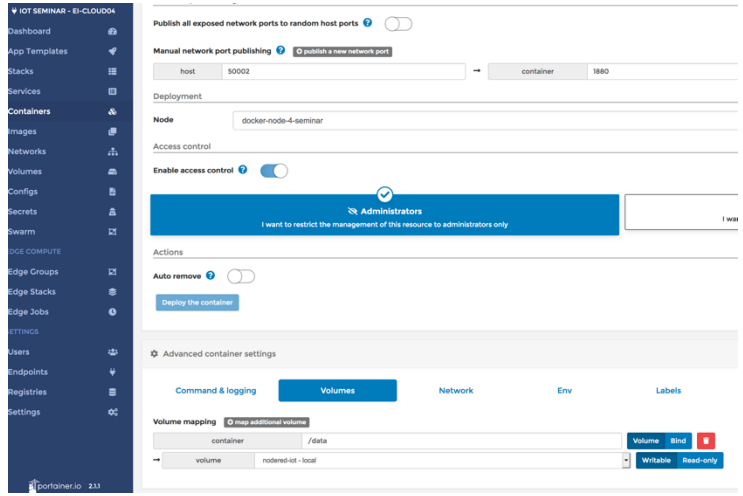
IP Range	Port Range



1.) Create your NodeRed Container

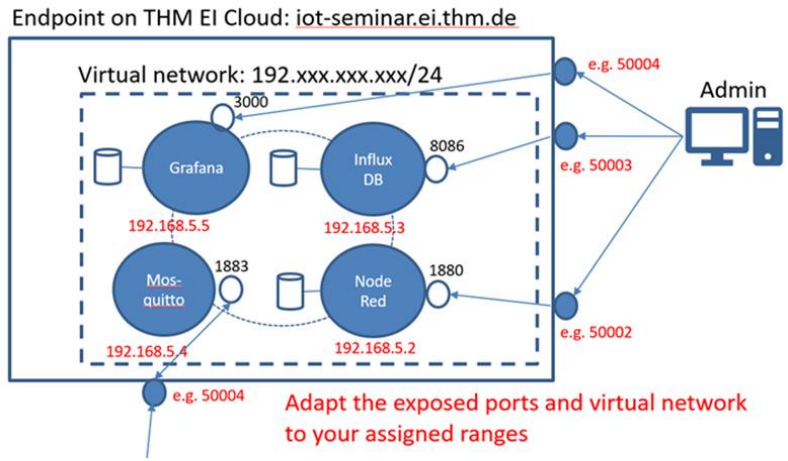
- **Login** to your endpoint
- **Create network:**
  Network: +Add Network and configure the following items:
  Name, Driver „Bridge“, specify subnet 192.xxx.xxx.xxx/24 => create
- View your network on network list
- **Create three volumes: NodeRed, InfluxDB and Grafana:**
  Volumes: +Add volume and configure the following items:
  Choose a suitable Name, Driver local => create
- **Deploy Nodered container in subnet and attach volumes:**
- Container: +Add container and configure the following items:
  Name, image „nodered/node-red:latest (Registry DockerHub)“, publish network port (Portmapping) **host port** <your port> **container port** 1880
  Go down to Volulmes (for persistent data): +map additional volume:
  container path „/data“ => choose your volume from list „Name –local“
- Network: choose your network and specify (for now) IPv4 Address:
  192.xxx.xxx.2
- Restart policy: „on failure“ (notice what else you can configure, i.e. memory, cpu resources etc.)

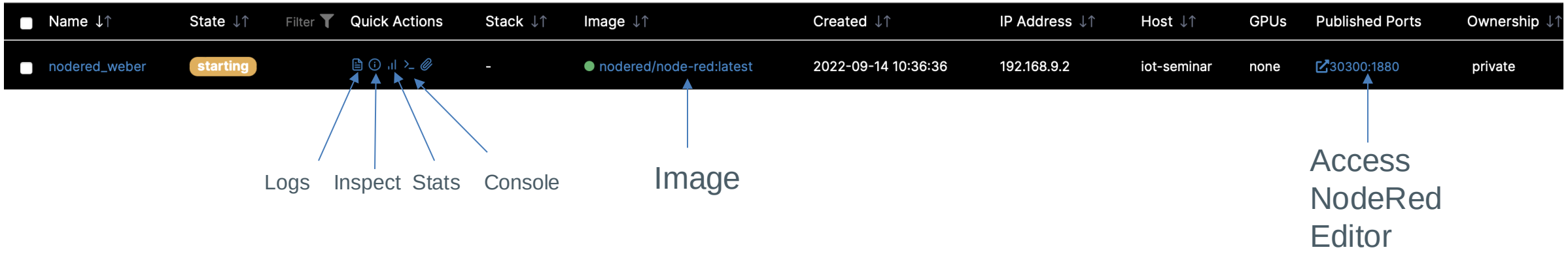




1.) Create your NodeRed Container

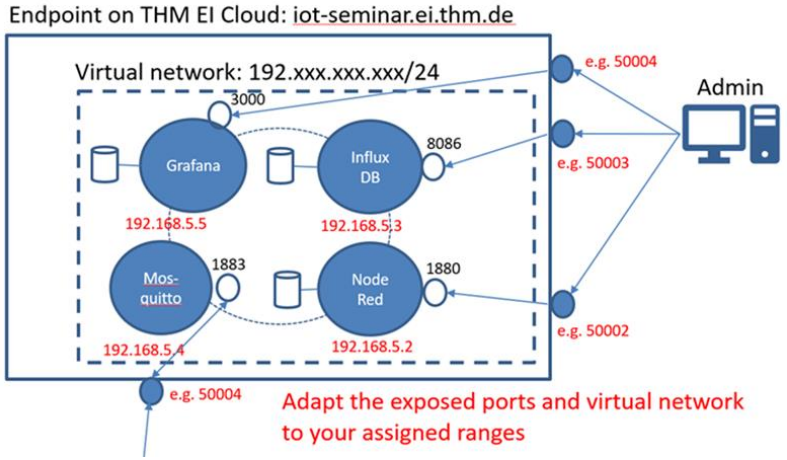
- Deploy container and have a look on your container in the container list
- Access Nodered Editor via link to test functionality and check URL
- Access your volume and see the stored files, such as flows.json (stores flows), settings.js (will be used later) etc.

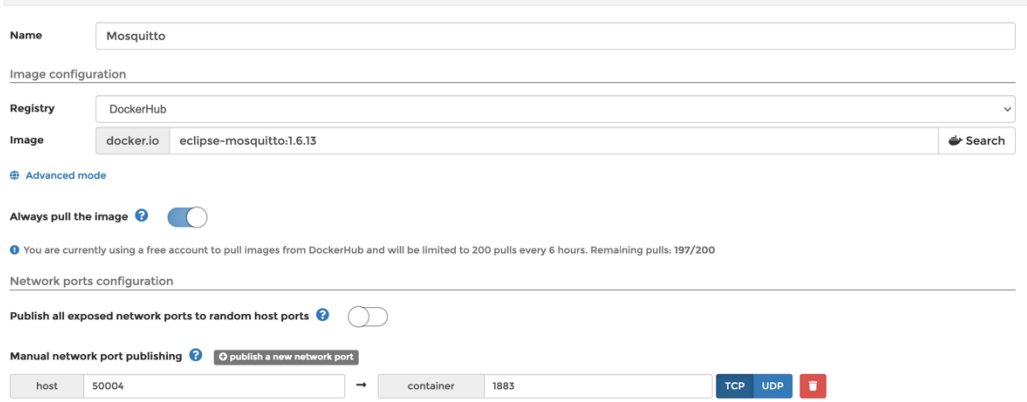




2.) Create a Mosquitto Container

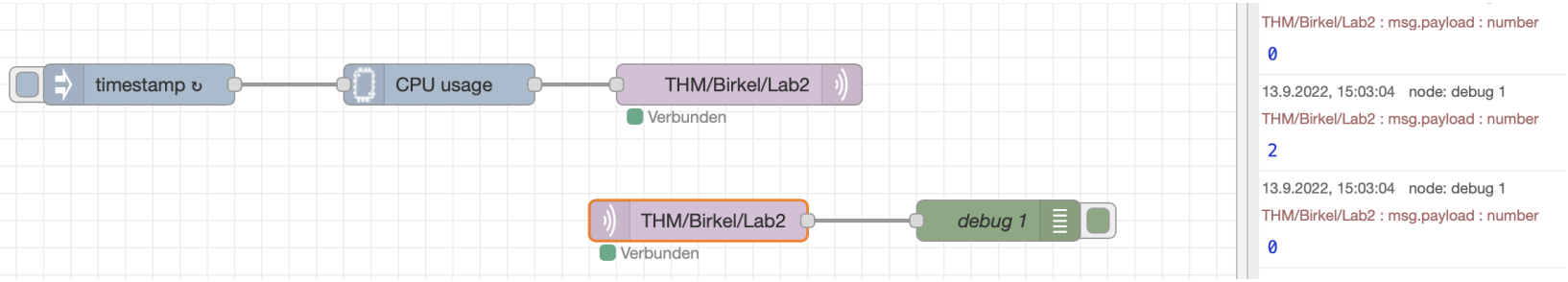
- **Deploy Mosquitto containers via unsecured port 1883:**
- Container +Add container and configure: name, image „eclipse-mosquitto:1.6.13“ (docker.io), container port 1883, published host port _____
  No persistant data is needed => no volume
  Specify your network and (for now) IPv4 Address: 192.xxx.xxx.4
  restart policy: „on failure“ (opt. specify Memory, CPU etc.)
- Deploy container and have a look on your containter in the container list:





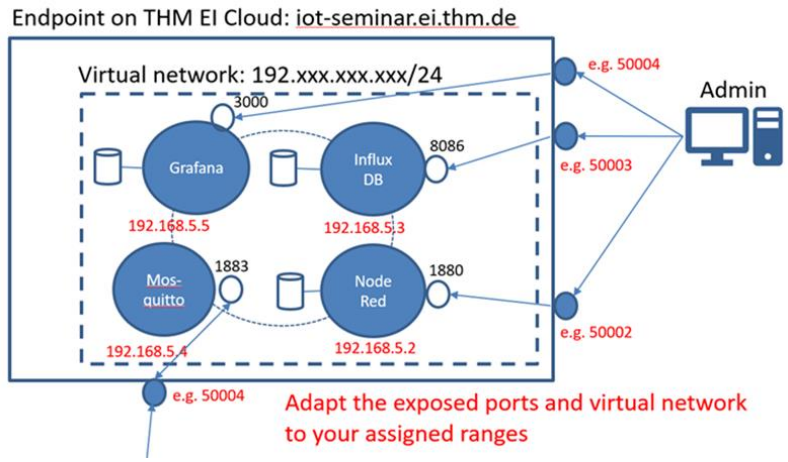
3.) Configure Node Red & MQTT

- **Open Node Red**
  - Go to manage palette and add CPU and node-red-contrib-influxdb node
  - Send & receive the CPU Load via MQTT and connect a debug node
  - Change the output to “a parsed JSON object”
  - Check if connection to MQTT server is successful. Try the following Broker options:
    - A.) use your own broker via
      - Public IP: _____ Public Port: _____
      - Private IP: _____ Private Port: _____
    - B.) Use EI Broker:
      mqtt.ei.thm.de:1993 user/pw: **iotlab/iotlab** Topic format: ***THM/IoTLab/yourname***



4.) Deploy InfluxDB Container

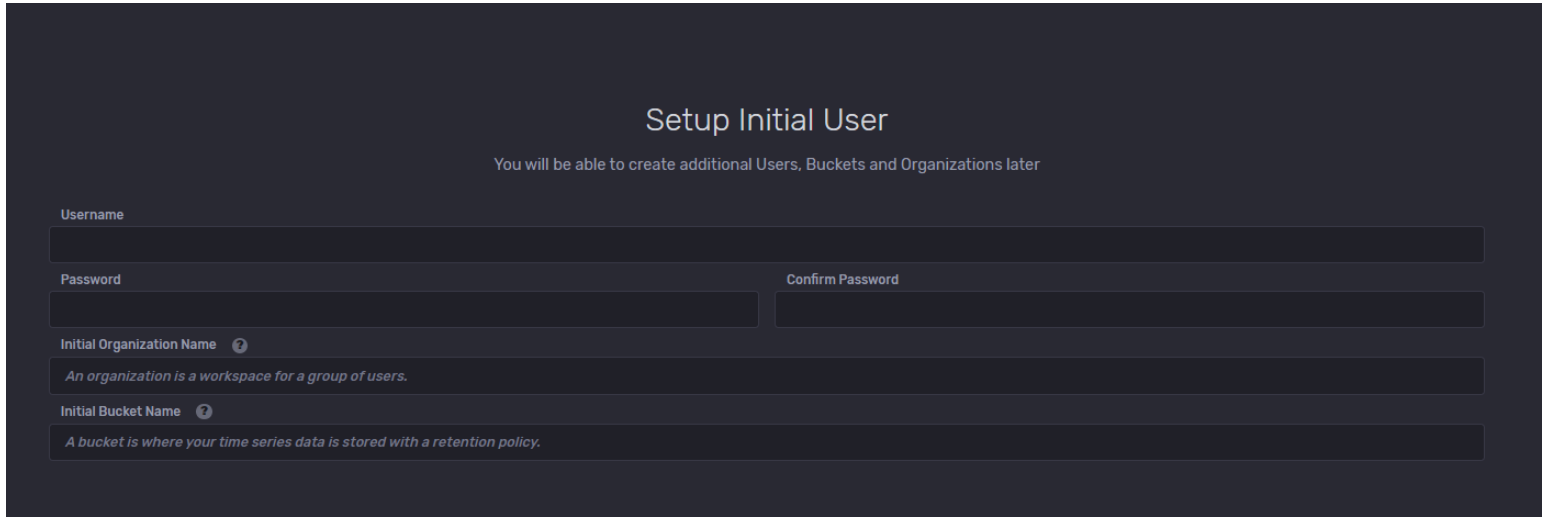
- **Deploy InfluxDB container in subnet and attach volume:**
  Container: +Add container and configure the follwoing items:
  name, image `influxdb:latest` (Registry DockerHub),
  +publish new network port: host 50xxx and container 8086
  go down to Volumes: +map addtional volume:
  container path according to influxDB documentation online:
  „`/var/lib/influxdb2`“ => choose the volume from list „name – local“
- Network: select your network and (for now) IPv4 Address: 192.xxx.xxx.3
- Restart policy and choose on failure
- Deploy container and have a look in the container list:



InfluxDB_Birkel_Lab2	running		-	influxdb:latest	2022-09-13 14:49:17	192.168.5.3	iot-seminar	none	50003:8086

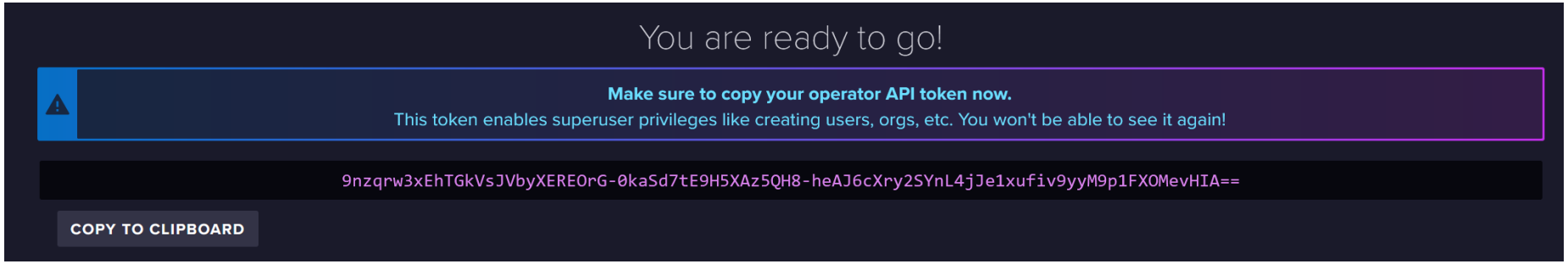
5.) Configure InfluxDB

- **Open InfluxDB**
  - Walk through the Get Started guide and setup initial user
    - Choose a username and password and note it down
    - Choose the Organization Name “EI”
    - Name your data bucket



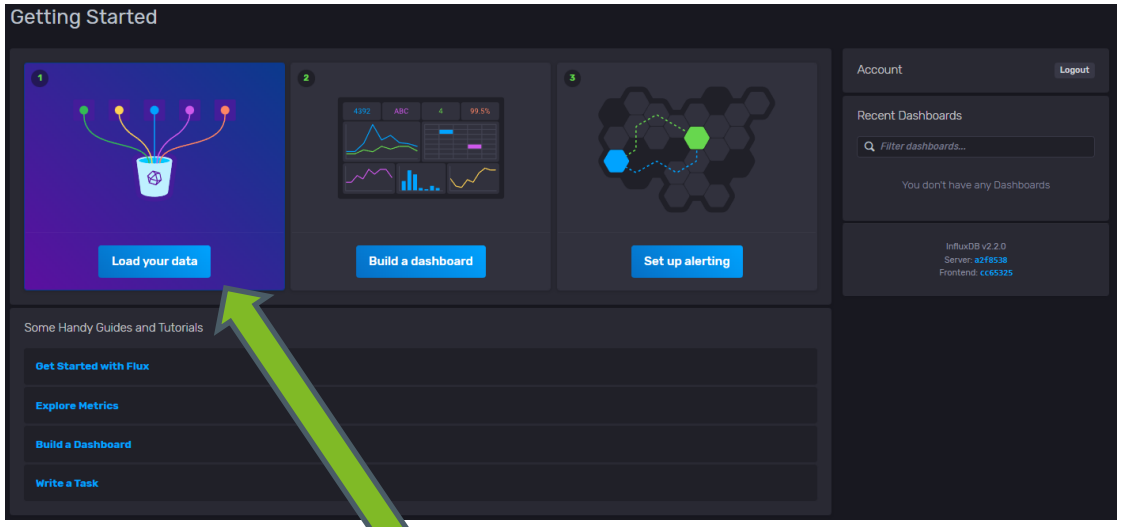
5.) Configure InfluxDB

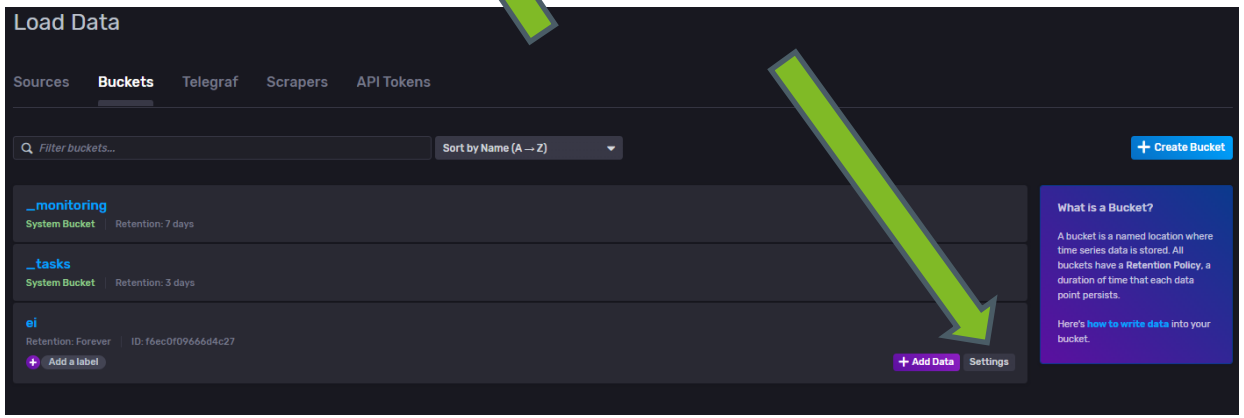
- **Save your InfluxDB API key**
  - Make a note of the API key, as we'll be using it later.
  - !! Later not readable anymore !!



5.) Configure InfluxDB

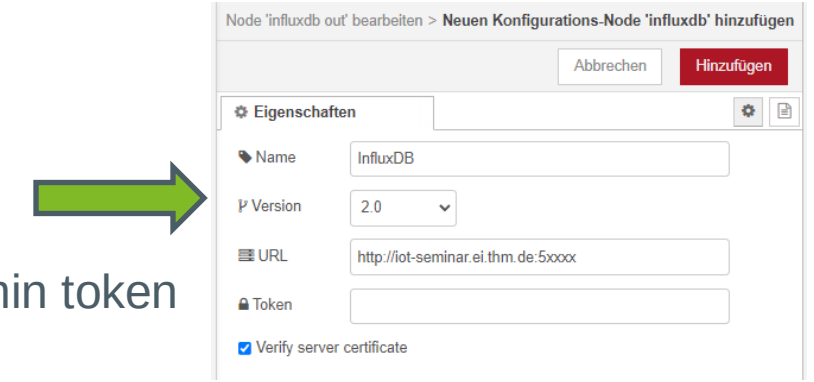
- **Configure InfluxDB**
  - Click “Load your data” on the Getting Started Page
  - In the “Load Data” menu, head to Buckets & change the retention policy in “settings”

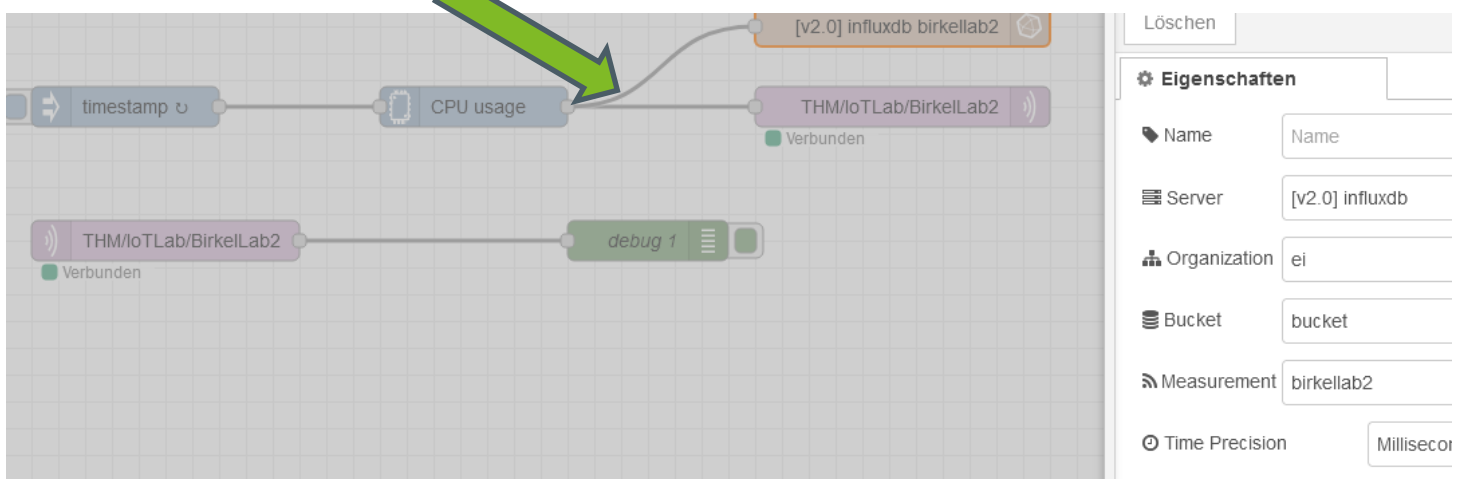


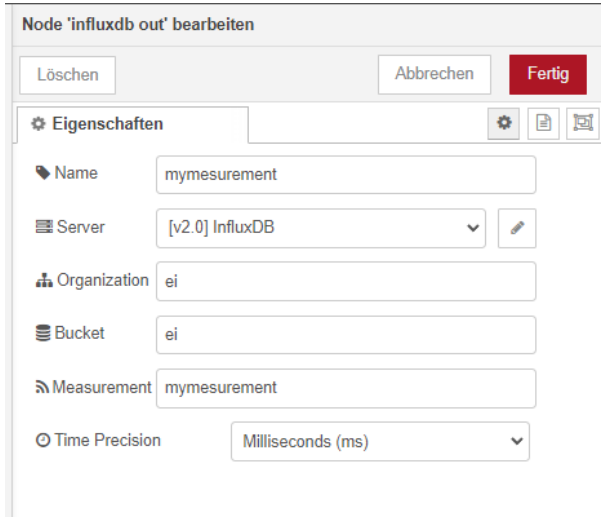


5.) Configure InfluxDB in NodeRed

- **Add InfluxDB Server to Node Red**
  - Go back to your Node Red Service and add an InfluxDB out node
  - Configure a new InfluxDB Server and select version 2
  - Enter your URL (e.g. iot-seminar.ei.thm.de:5xxxxx) and the copied admin token
  - Add the configuration
  - Edit the InfluxDB out node and enter your organization, bucket, measurement
  - Connect CPU usage node to your influxdb out node:

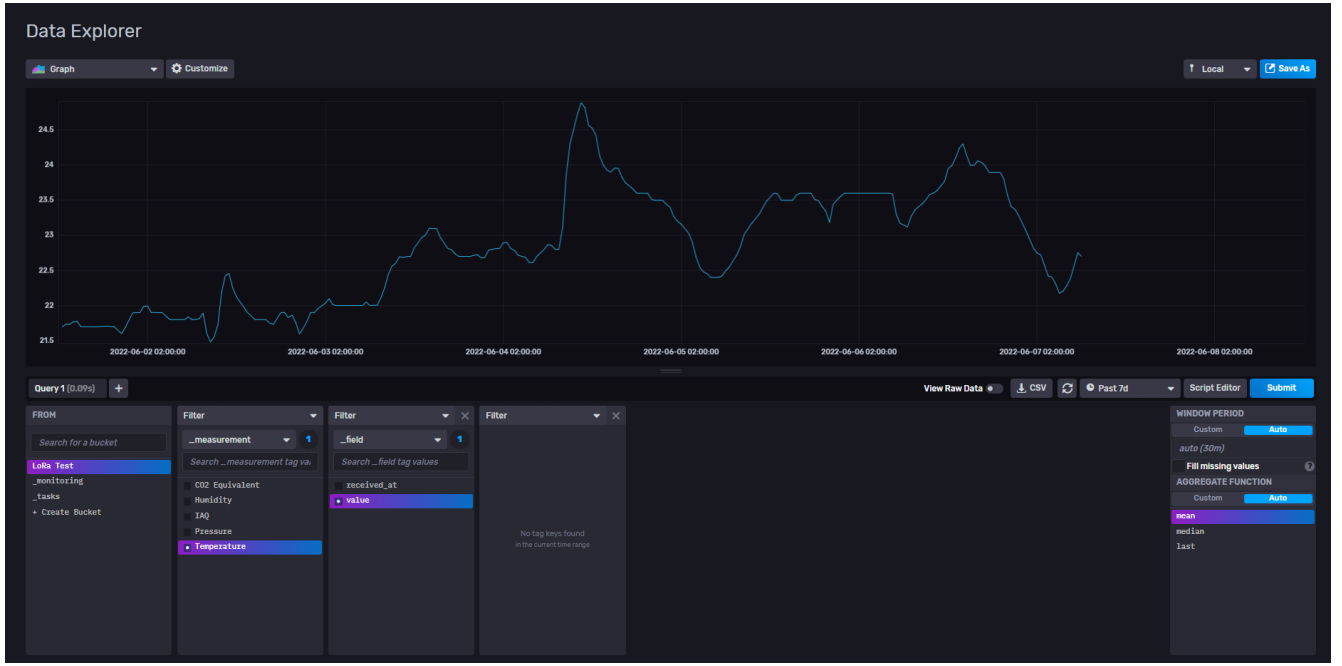






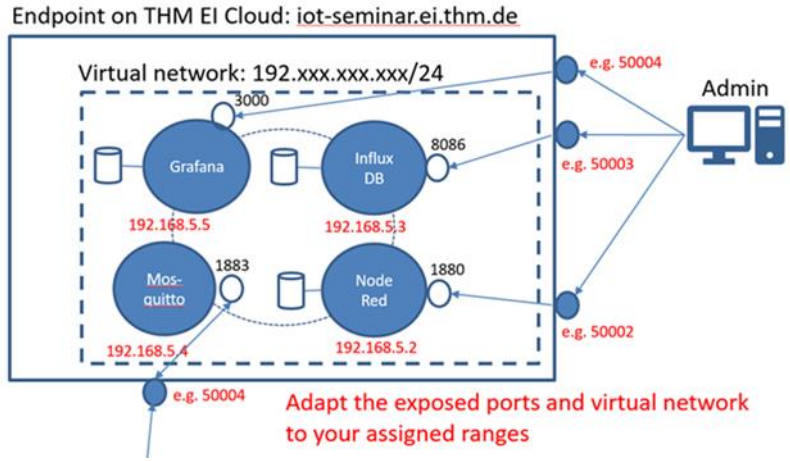
5.) Configure InfluxDB

- **View your measurements**
  - On your InfluxDB Web-UI go to the Explore menu
  - You can now select your bucket and view your measurements and click submit



6.) Deploy Grafana Container

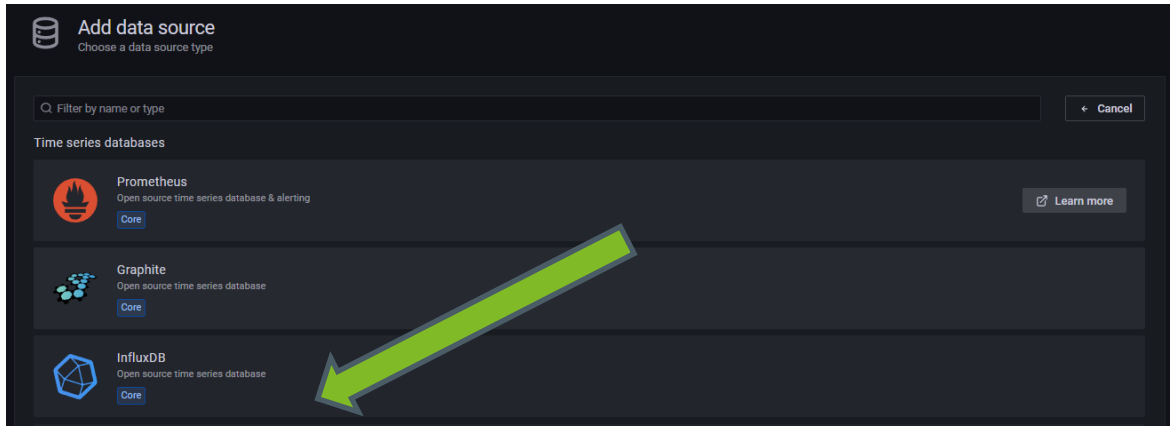
- **Deploy Grafana container in subnet and attach volume:**
  Container: +Add container and configure the follwoing items:
  name, image `grafana/grafana:latest` (Registry DockerHub),
  +publish new network port: host 50xxx and container 3000
  go down to Volumes: +map addtional volume:
  container path according to InfluxDB documentation online:
  „`/var/lib/grafana`“ => choose the volume from list „name –local“
- Network: select your network and (for now) assign your IPv4 Address e.g. 192.xxx.xxx.5 (see dedicated IP/Port range in Moodle)
- Restart policy and choose on failure
- Deploy container and have a look in the container list:



Lab2_Birkel_Grafana running - grafana/grafana:latest 2022-09-13 15:25:10 192.168.5.10 iot-seminar none 50010:3000

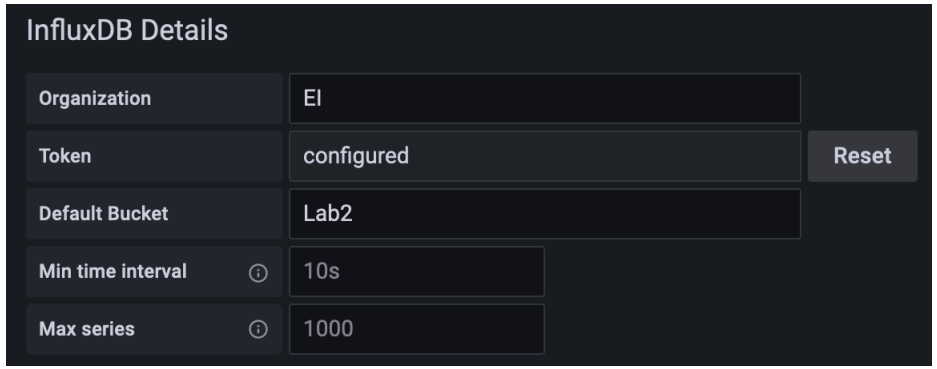
7.) Configure Grafana

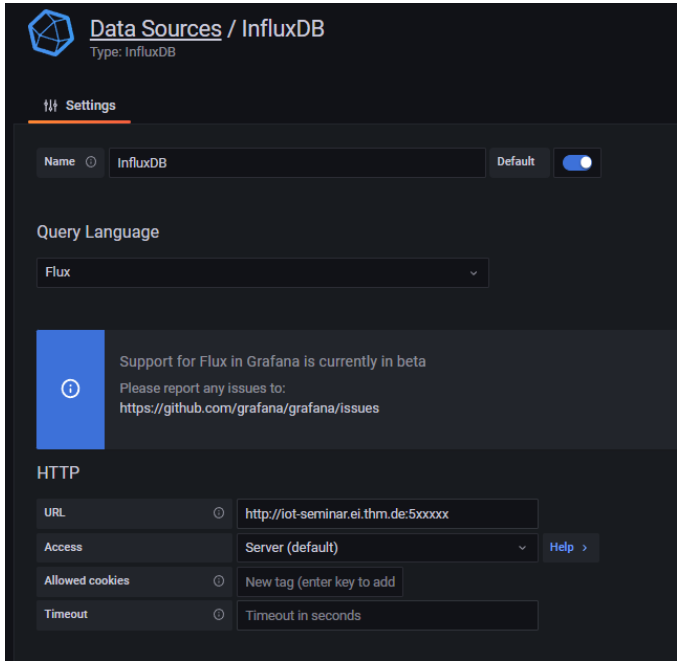
- **Open the Grafana Web Interface**
  - Login with the default user/password combination (admin/admin)
  - Choose a new password and note it down
  - On the Grafana Homepage click the “Add your first data source” Panel
  - Add InfluxDB as a source

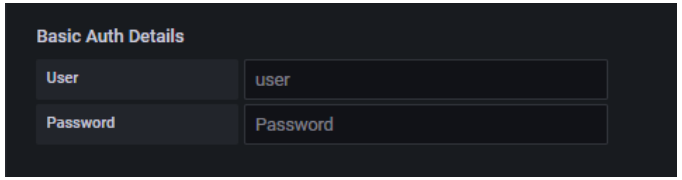


7.) Configure Grafana

- **Add InfluxDB to Grafana**
  - Change the Query Language to Flux
  - Enter the URL of your InfluxDB (e.g. iot-seminar.ei.thm.de:5xxxxx)
  - Enter your previously defined username and the admin's token under Basic Auth Details
  - In InfluxDB Details enter Organization, Token and Default Bucket
  - Click save and test

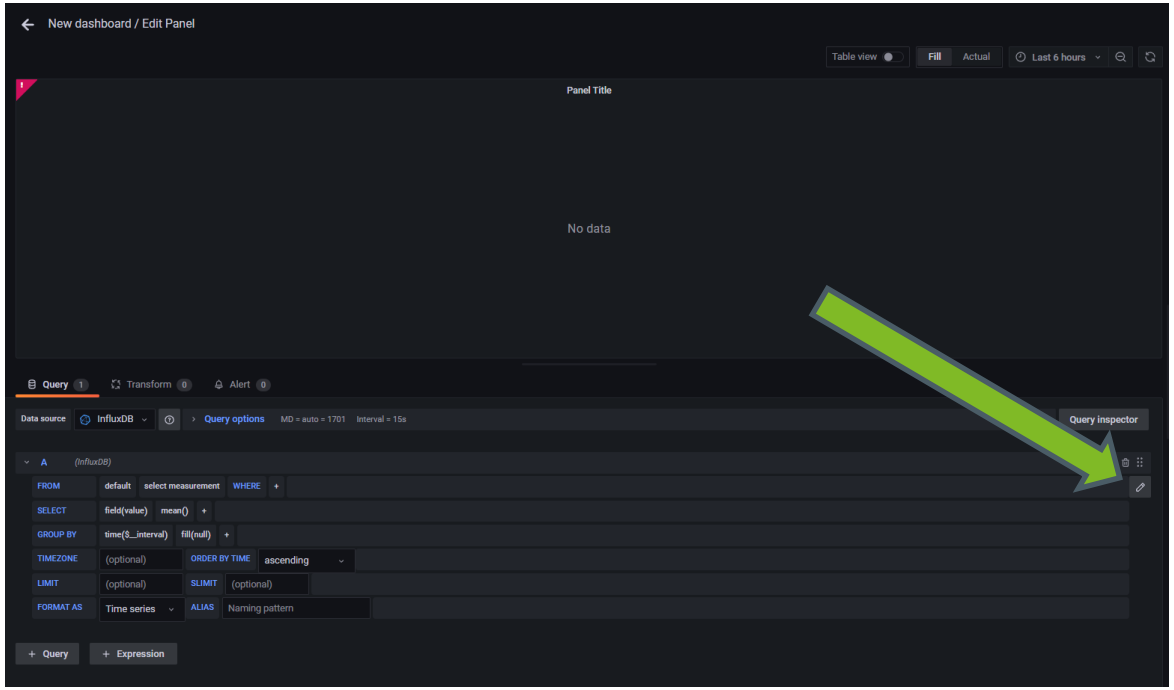






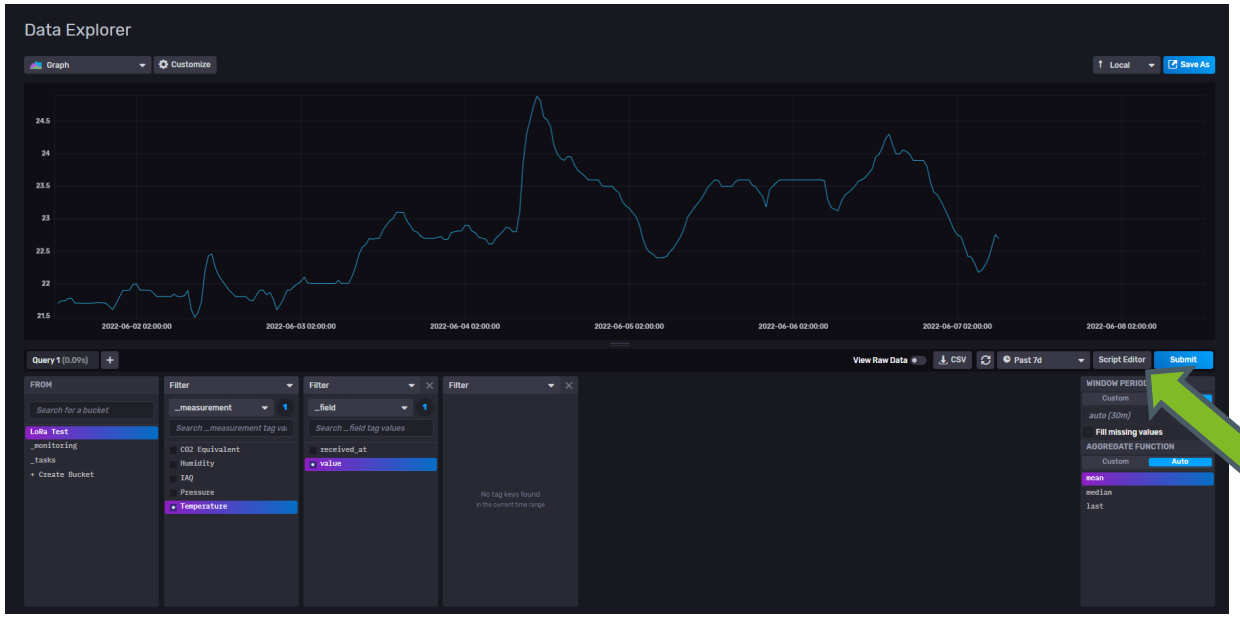
7.) Configure Grafana

- **Create a dashboard and display measurements**
  - Click on the “Create your first dashboard” panel at the Homepage of Grafana
  - Add visualization
  - Click the pencil to enter raw query editor



7.) Configure Grafana

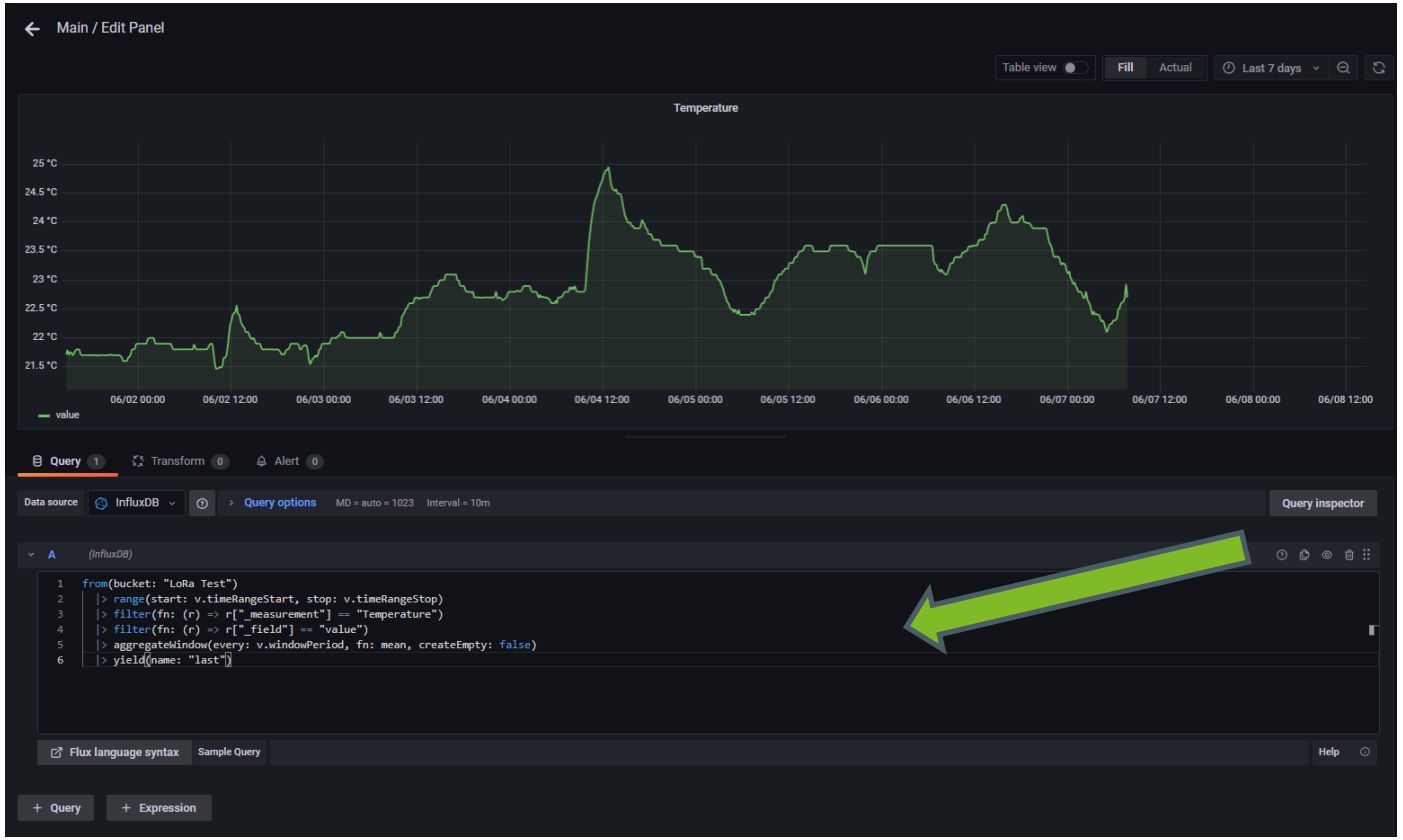
- **Go back to your InfluxDB Explorer**
  - Choose your bucket and a measurement you want to visualize (value)
  - Click on the script editor button
  - Copy the query text




```
from(bucket: "Lab2")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "mymeasurement")
  |> filter(fn: (r) => r["_field"] == "value")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

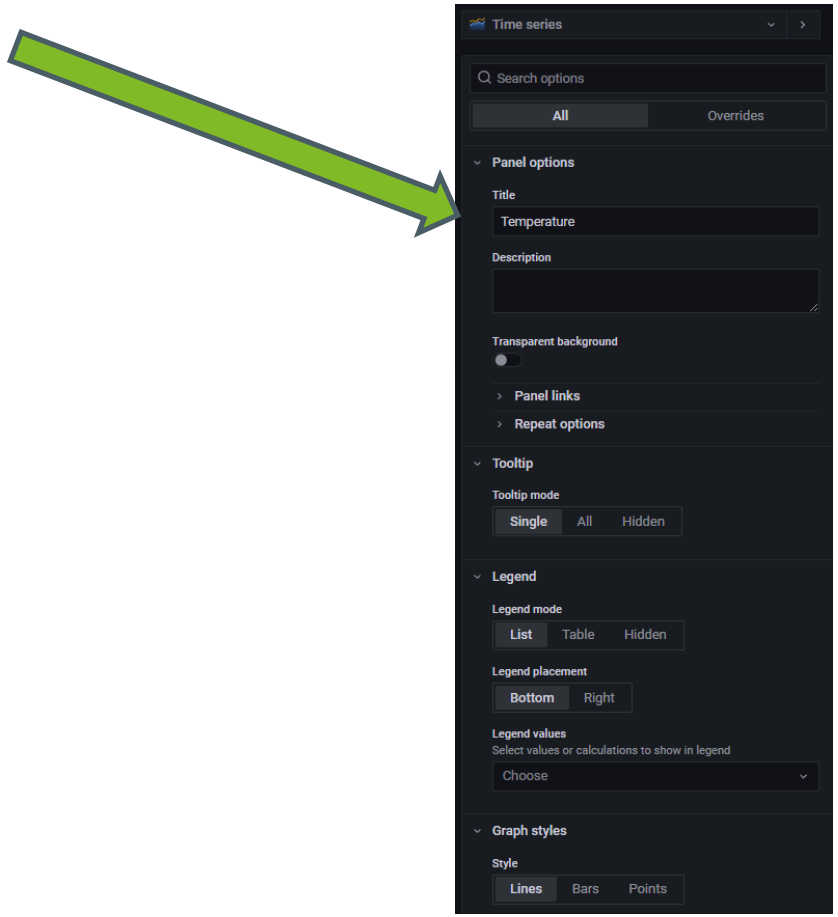
7.) Configure Grafana

- **Insert the copied Query text in your Grafana raw query editor**



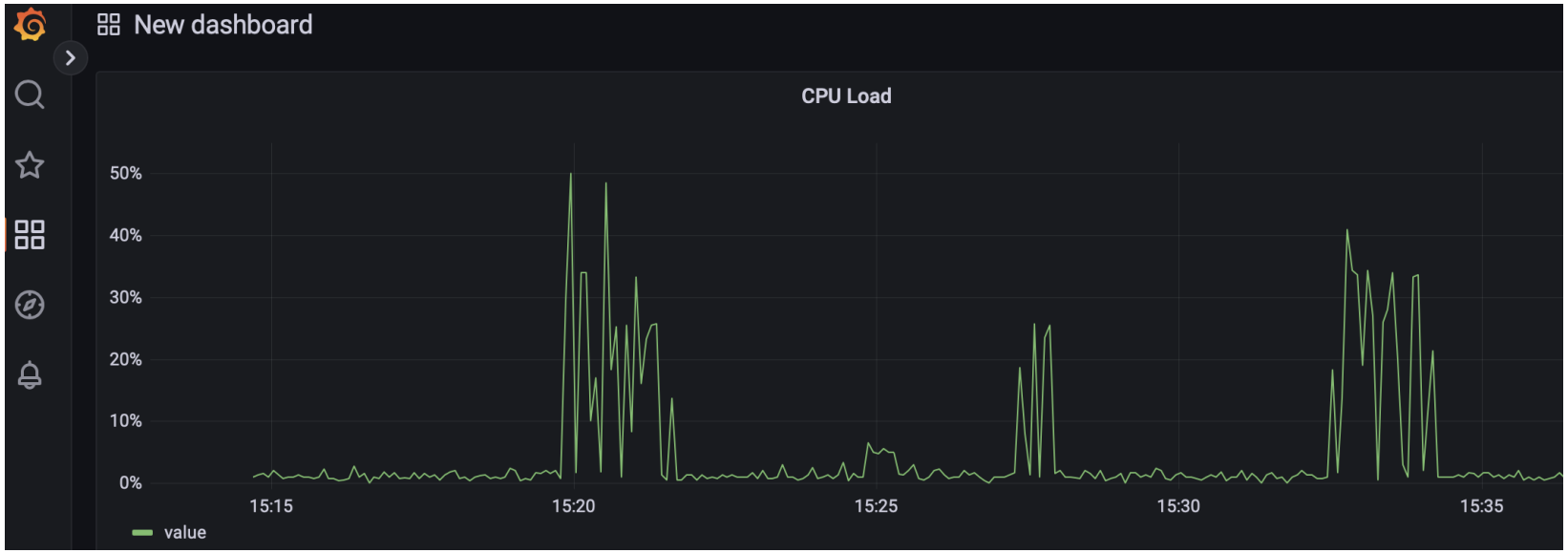
7.) Configure Grafana

- **Give it a title, description and the correct unit**
  - Fine tune your graph if you want to
  - Click apply if you are done and SAVE your dashboard!
    It will not be saved automatically!

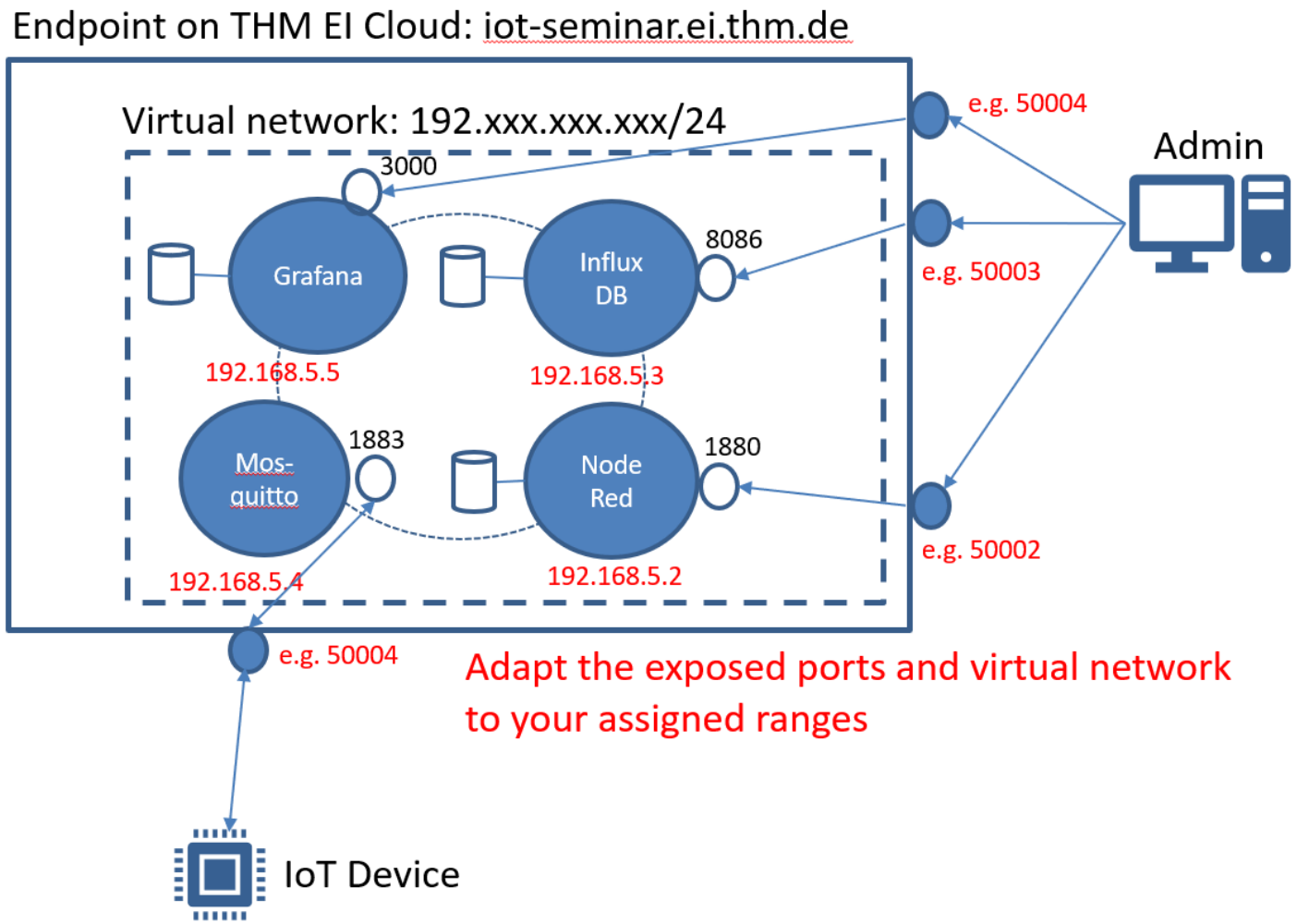


7.) Configure Grafana

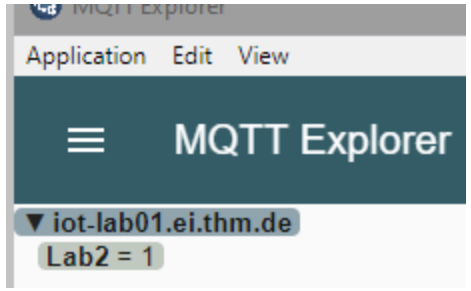
- This is how your dashboard could look like:



8.) Send CPU Load from local docker client

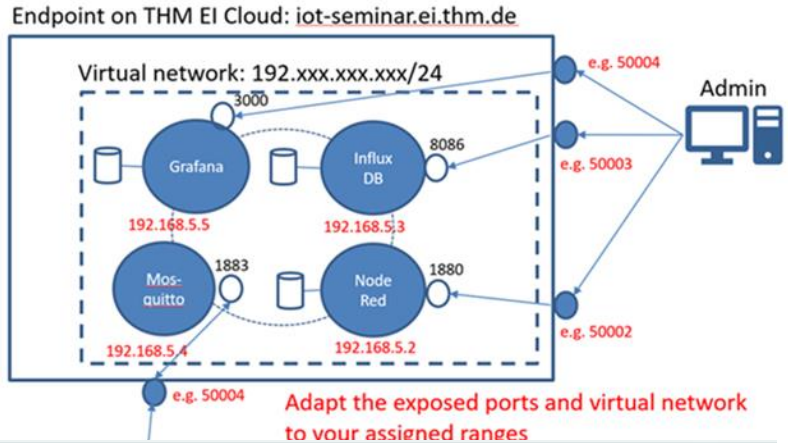


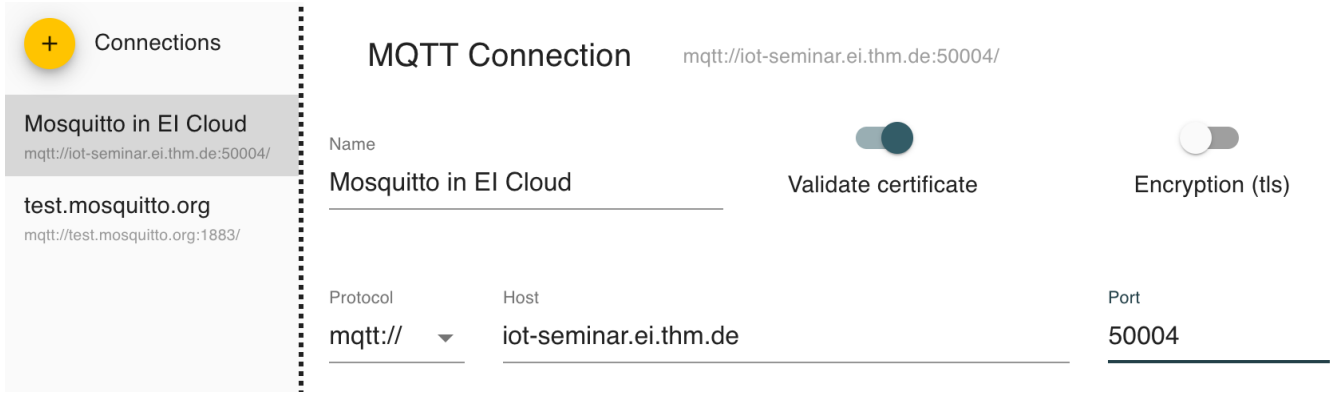
Check if your data
is arriving using
the MQTT Explorer or MQTTx

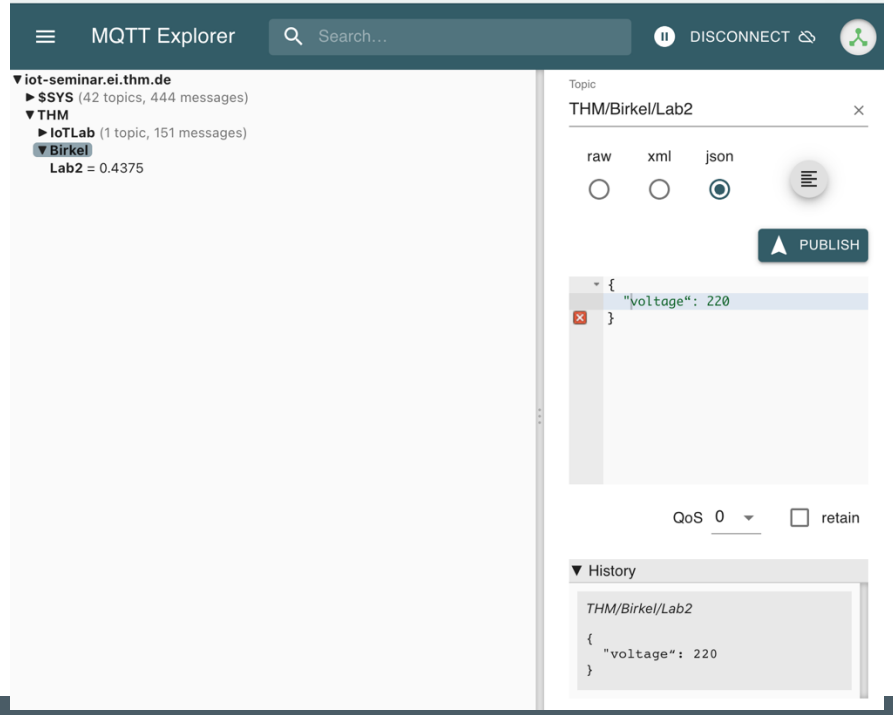


9.) Experiment with MQTT Client

- **Download&Install MQTT Explorer (already installed in Lab)**
  - Add a new connection Server „iot-lab0x.ei.thm.de:<your public port>“
  - Connect to your MQTT Broker
  - Publish JSON object: { „voltage“: 220} & insert it into your database using „iot-seminar.ei.thm.de:<your port>“
  - **Check if the message is in the InfluxDB database**

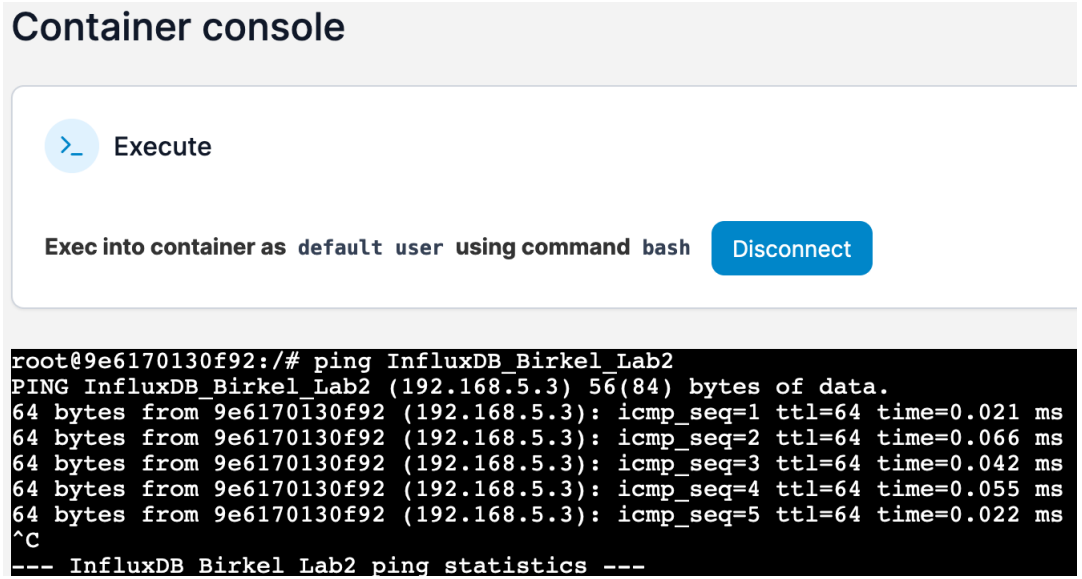


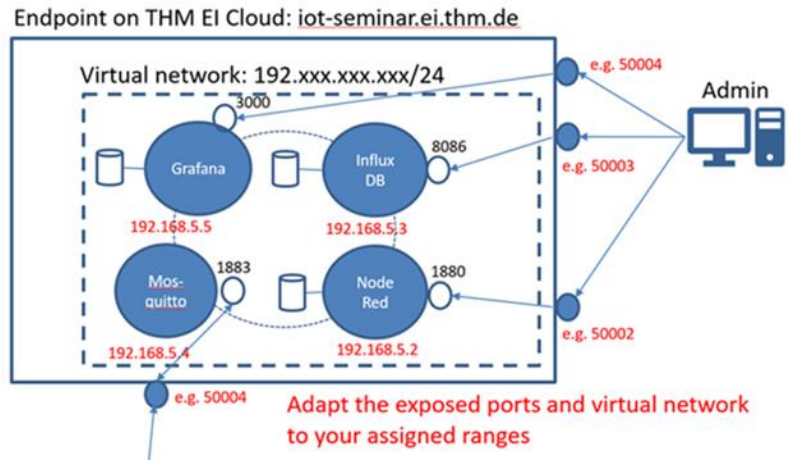


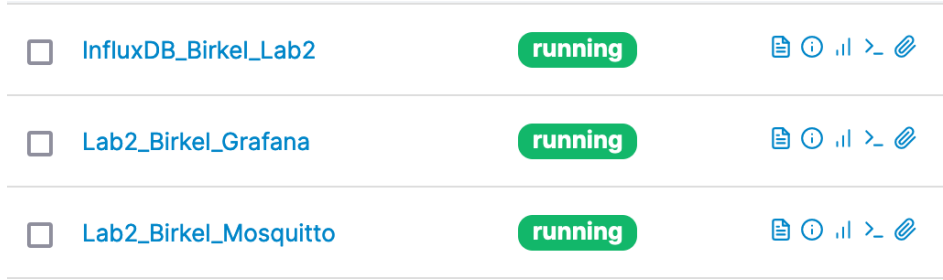


10.) Use Container Names instead of IP Adress

- It is strongly recommended to **use the container name instead of the IP addresses**, since the IP address might change during orchestration. Note that when using the container name, you must also use its container port (e.g. 8086) and not the published port.
- Go to the NodeRed console and ping your Container via its container name:



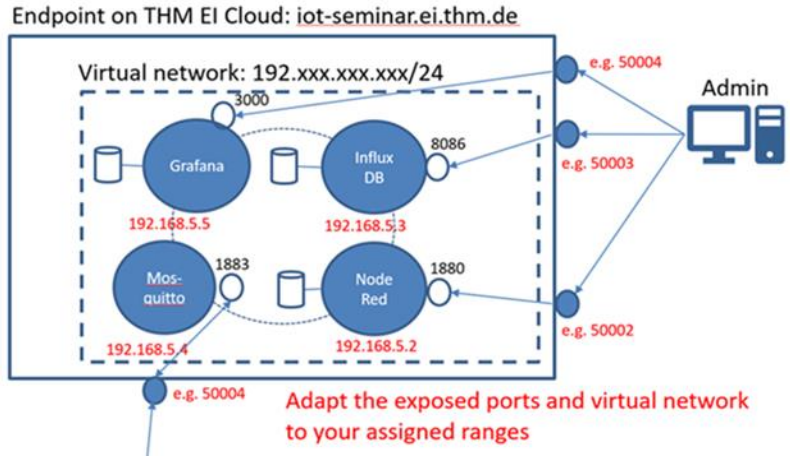




c.) Use Docker manifest file (YAML format)

- Perform the steps above using
  - a.) Portainer GUI
  - b.) Docker Command Line Interface
  - c.) Docker manifest file (YAML format)

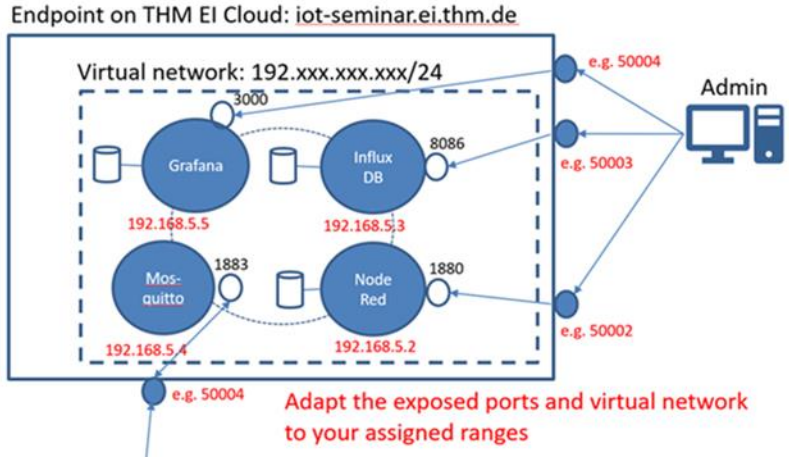
- **Note: Before you start you need to delete the application you previously created in Part a.) using the Portainer GUI**



Manifest File and Orchestration

- We will give the orchestrator „Docker Swarm“ a deployment/manifest File in **YAML** format, which will define our cloud application
- YAML is a human-readable data-serialization language, which is commonly used for configuration files in applications where data is being stored or transmitted
- Container orchestration automates the deployment, management, scaling and networking of containers

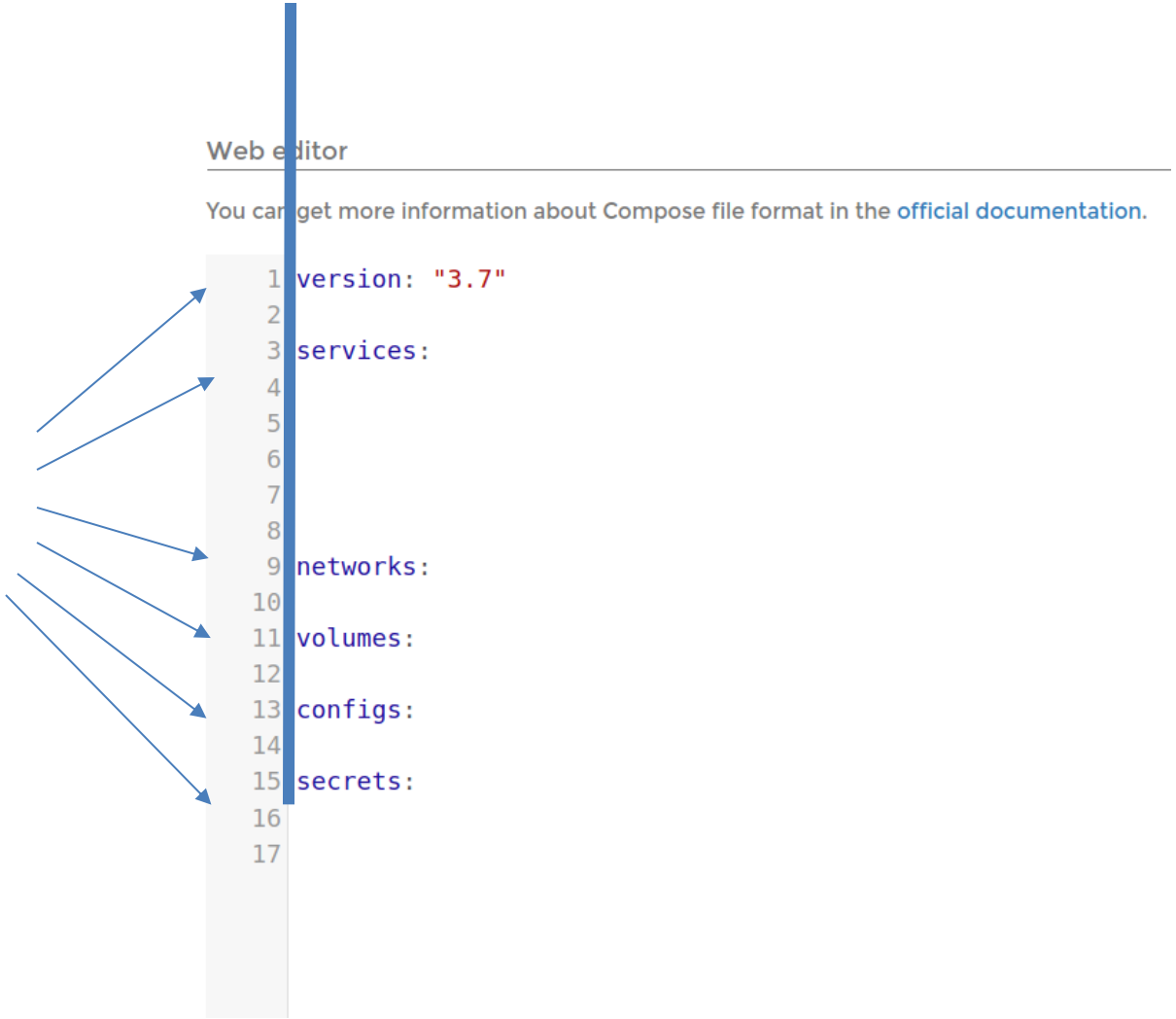
- We will
  - Edit and upload the deployment file
  - Deploy the stack
  - Run and modify our cloud application



YAML File Format

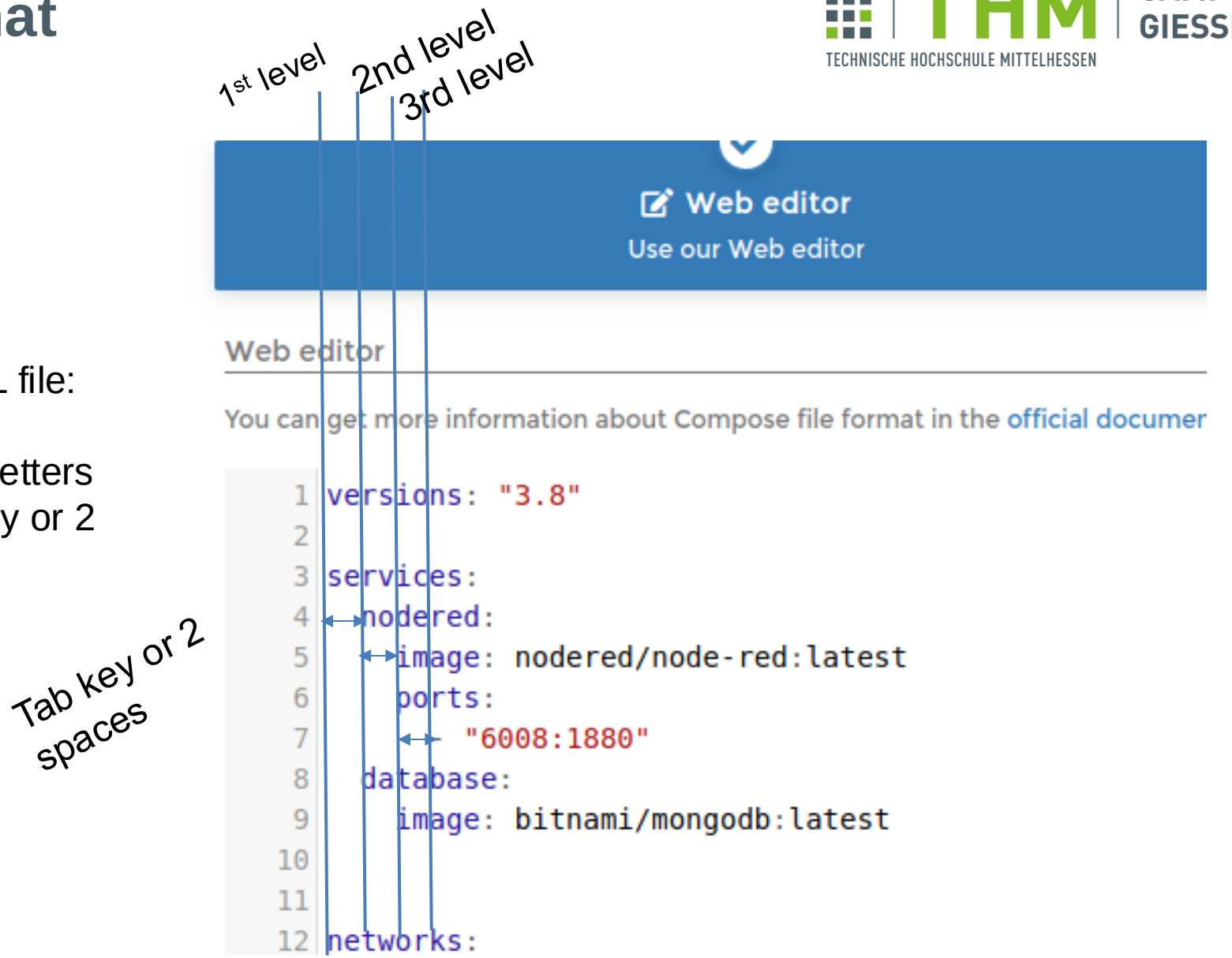
First level

Remember first level keywords



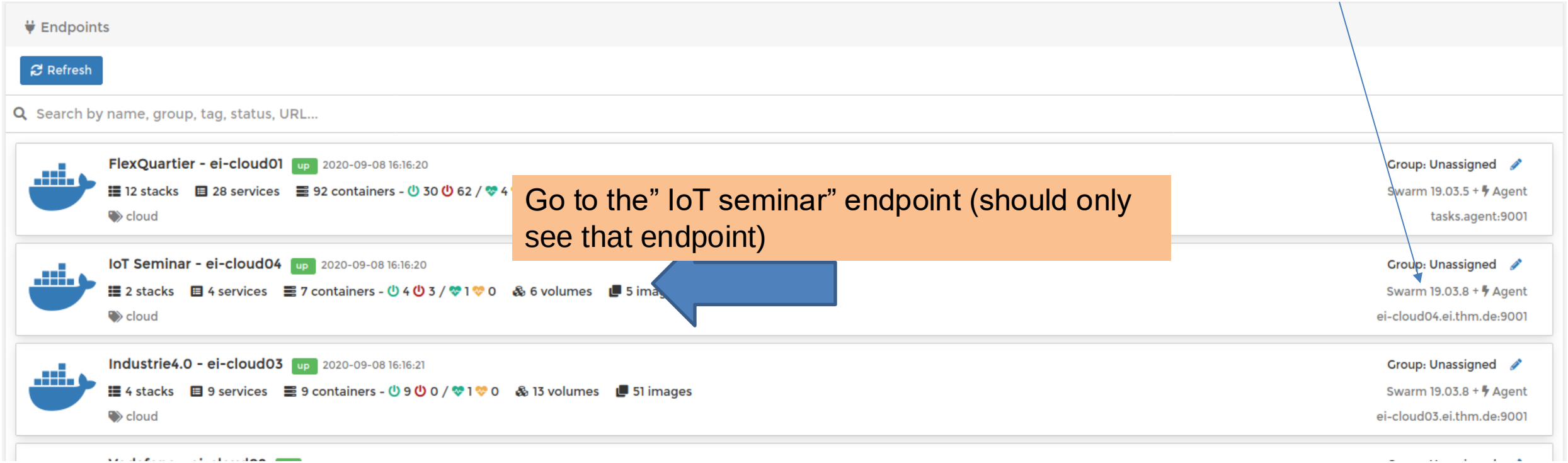
YAML File Format

How to write a YAML file:
- No Spaces
- No Upper Case Letters
- Use either Tab key or 2 spaces



Create a new swarm stack

Docker engine release



Go to the" IoT seminar" endpoint (should only see that endpoint)

Create a new swarm stack

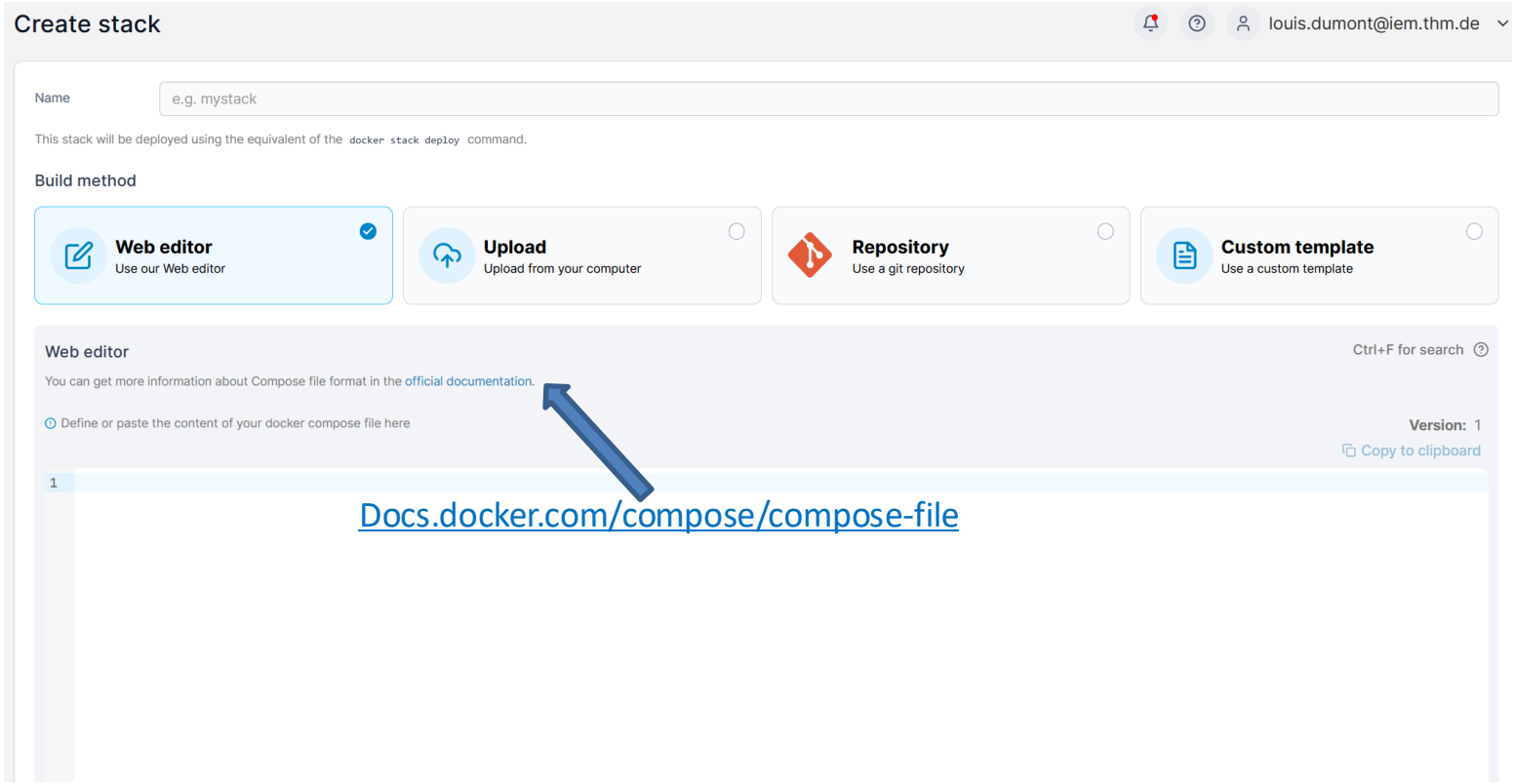
- Go to your endpoint / Stacks: +Add stack and give it a unique name without spaces or upper letters

- Choose among 4 options
  - **Web editor:** Directly edit your YAML File
  - **Upload:** Upload your YAML File from your Host
  - **Repository:** Upload your File from a git Repo
  - **Custom template**: Deploy and modify an existing template

- Check Version depends of your Docker endpoint (HOME) and compare to official doucmentation

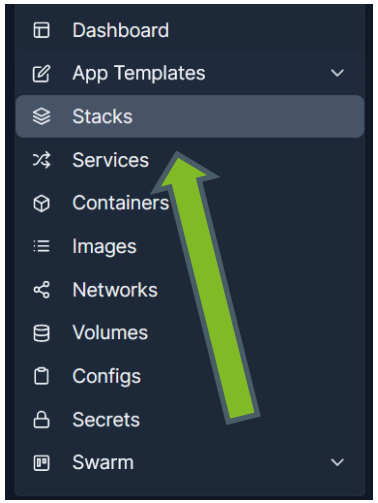
- Example :

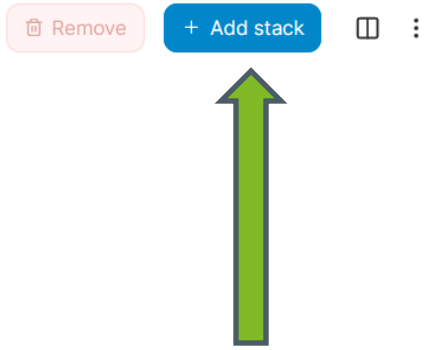
  Compose File Format 3.8 supports
  Docker Engine release 19.03.0+

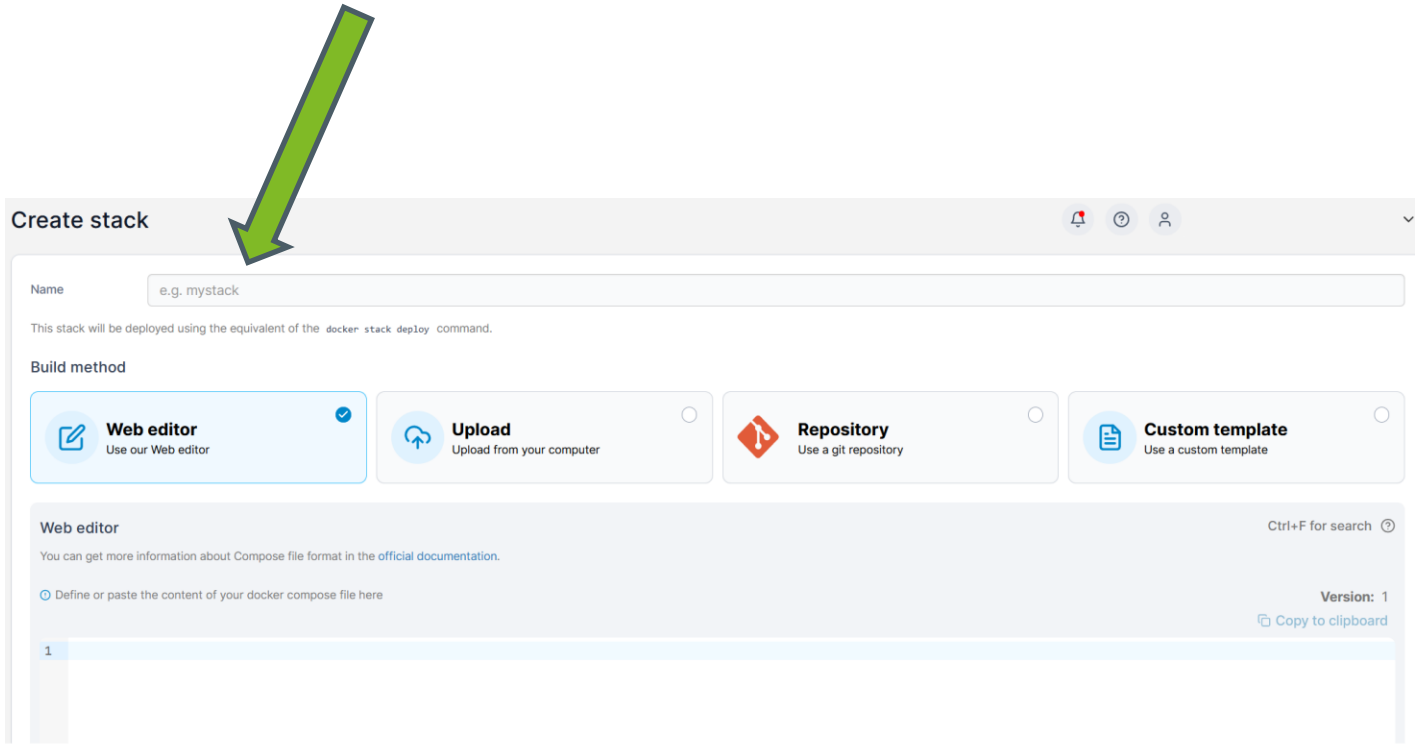


b.) Create a new stack

- **Login** to your endpoint
  - Head to “Stacks” and click “Add stack”
  - Choose <yourname>_lab2_ws2223 as the name of your stack
  - Use the web editor to configure your services



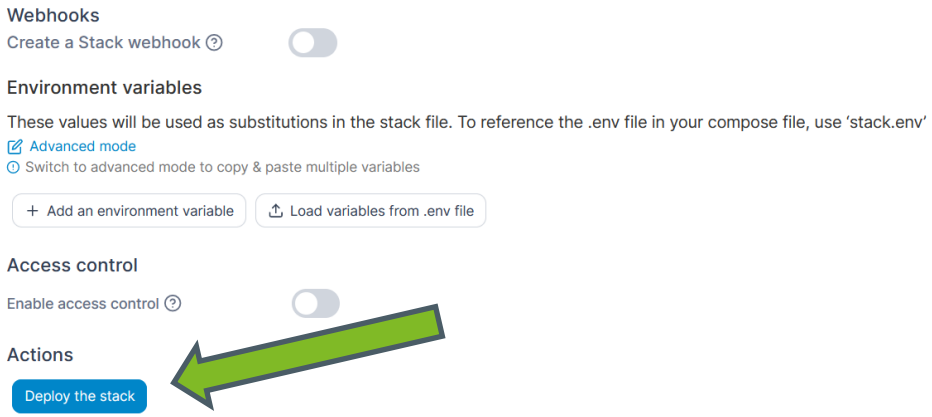




b.) Create a new stack

THM | CAMPUS GIESSEN | EI Elektro- und Informationstechnik
TECHNISCHE HOCHSCHULE MITTELHESSEN

- **Deploy Stack using the docker-compose web editor:**
  - Add the services Mosquitto, NodeRed, InfluxDB and Grafana
  - Change <yourname> to your name (lowercase only)
  - Change the 5xxxx ports to your own ports assigned by the tutor
  - Ensure that each service has its own unique port; do not reuse ports
  - Check your configuration and deploy the stack
    don't get nervous, it takes about a minute to pull the images ☺
  - DO NOT copy from slides !!!



Docker-compose:

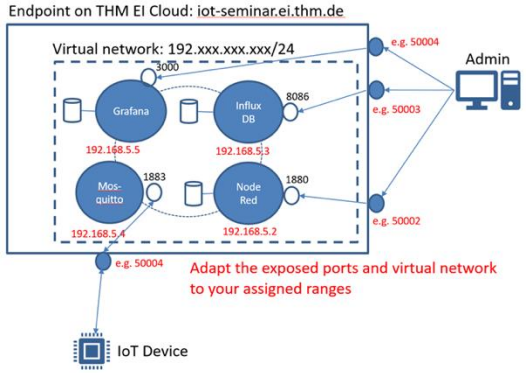
```
version: "3.8"
services:
  node-red-<yourname>:
    image: nodered/node-red:latest
    ports:
      - "50000:1880"
    networks:
      - <yourname>_net
    volumes:
      - <yourname>_nodered_vol:/data

  mosquitto-<yourname>:
    image: eclipse-mosquitto:1.6.13
    ports:
      - "50004:1883"
    networks:
      - <yourname>_net

  influxdb-<yourname>:
    image: influxdb:latest
    ports:
      - "50001:8086"
    networks:
      - <yourname>_net
    volumes:
      - <yourname>_influx_vol:/var/lib/influxdb2:rw

  grafana-<yourname>:
    image: grafana/grafana:latest
    ports:
      - "50002:3000"
    networks:
      - <yourname>_net
    volumes:
      - <yourname>_grafana_vol:/var/lib/grafana:rw

volumes:
  <yourname>_nodered_vol:
  <yourname>_influx_vol:
  <yourname>_grafana_vol:

networks:
  <yourname>_net:
```

Specify your CloudApp/Stack with Webeditor

Web editor

You can get more information about Compose file format in the official documentation.

```yaml
version: "3.8"

services:
  nodered1:
    image: nodered/node-red:latest
    ports:
      - "50002:1880"
    networks:
      network1:
        ipv4_address: 192.168.5.2
    volumes:
      - nodered1-v:/data
  mosquitto1:
    image: eclipse-mosquitto:1.6.3
    ports:
      - "50004:1883"
    networks:
      network1:
        ipv4_address: 192.168.5.4
  mongodb1:
    image: bitnami/mongodb:latest
    ports:
      - "50003:27017"
    networks:
      network1:
        ipv4_address: 192.168.5.3


networks:
  network1:
    attachable: true # define if containers can communicat
    ipam:
      config:
        - subnet: 192.168.5.0/24


volumes:
  nodered1-v:
  mongodb1-v:
```

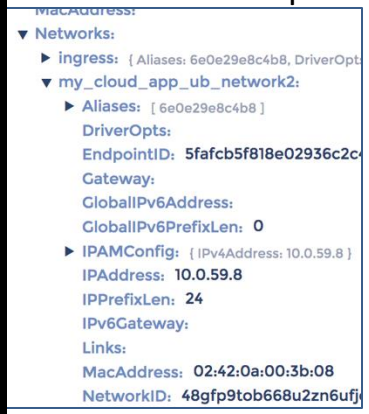


Modify your stack:

- no fix IP address!
- use DHCP: Docker will specify subnet & IP adressing
- Use service name to ping via CLI
- Validate Container IP via ifconfig in CLI and Portainer GUI

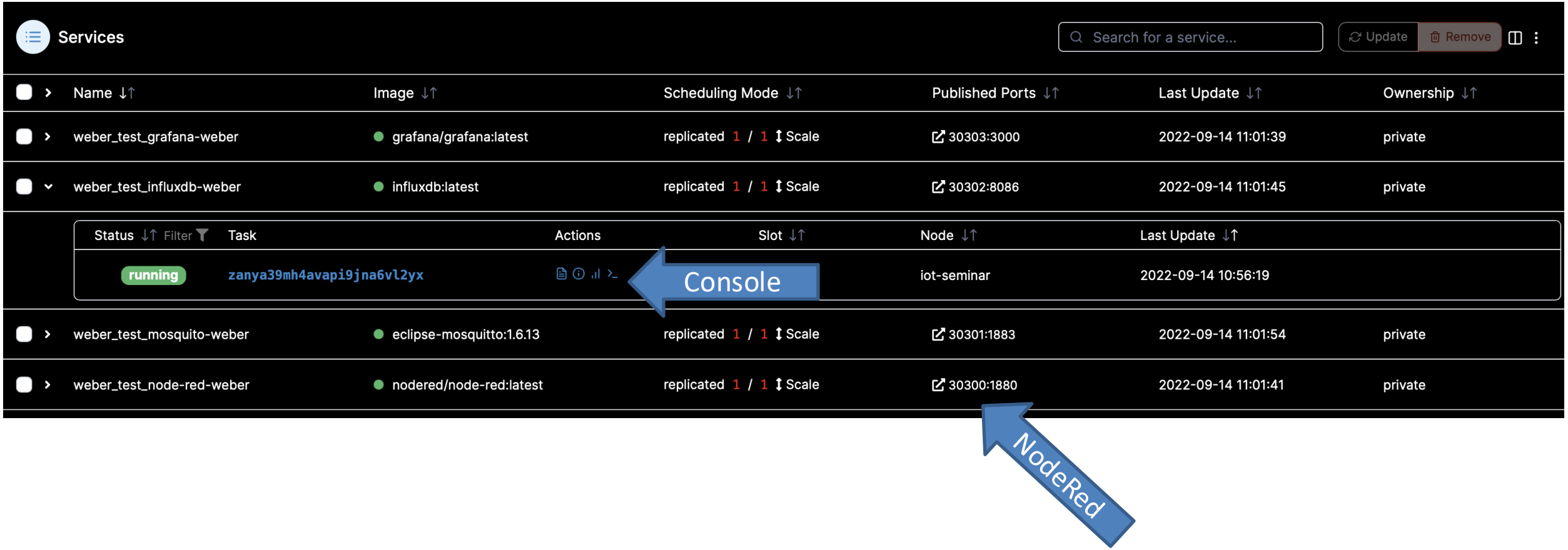
```yaml
version: "3.8"
services:
  node-red-weber:
    image: nodered/node-red:latest
    ports:
      - "30300:1880"
    networks:
      - weber_net
    volumes:
      - weber_nodered_vol:/data
  mosquito-weber:
    image: eclipse-mosquitto:1.6.13
    ports:
      - "30301:1883"
    networks:
      - weber_net
  influxdb-weber:
    image: influxdb:latest
    ports:
      - "30302:8086"
    networks:
      - weber_net
    volumes:
      - weber_influxdb_vol:/var/lib/influxdb2:rw
  grafana-weber:
    image: grafana/grafana:latest
    ports:
      - "30303:3000"
    networks:
      - weber_net
    volumes:
      - weber_grafana_vol:/var/lib/grafana:rw

volumes:
  weber_nodered_vol:
  weber_influxdb_vol:
  weber_grafana_vol:

networks:
  weber_net:
```

Portainer GUI: Container inspect



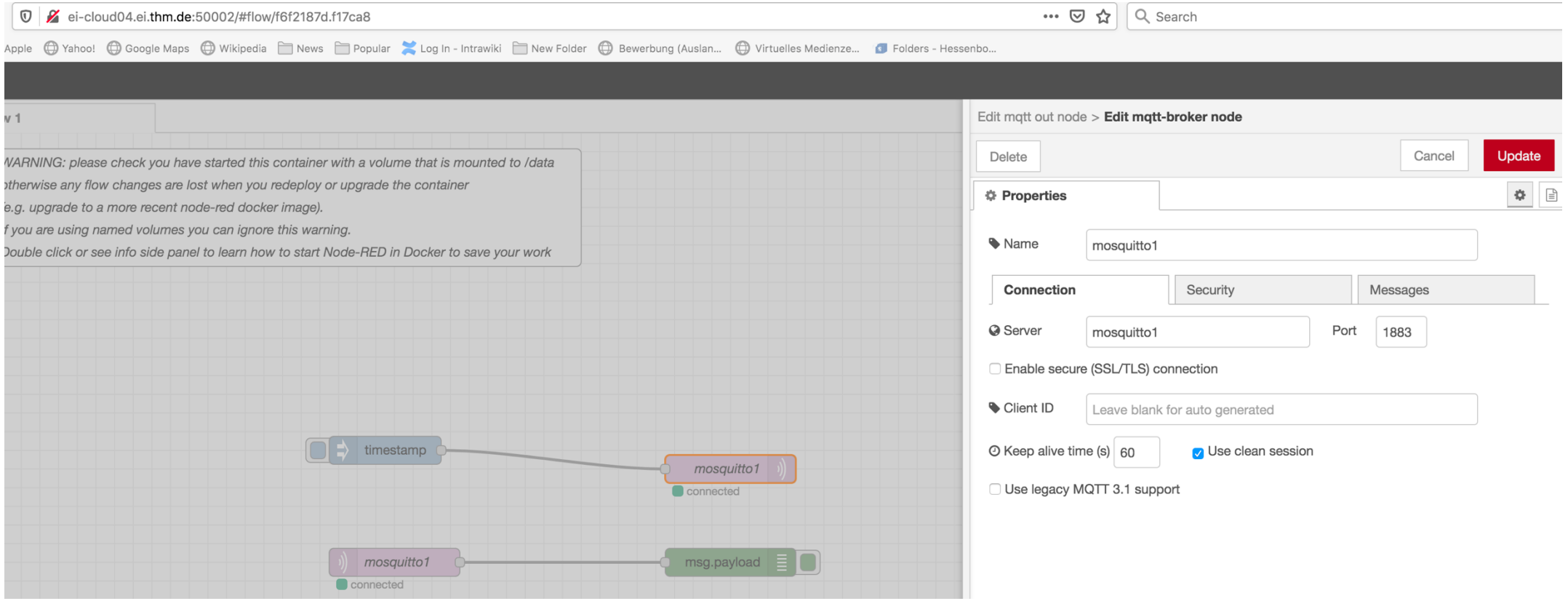
Access the console of the container in the swarm stack

- We have three containers running in the stack called “my_cloud_app_ub”
- Access the Mosquitto and InfluxDB console and check IP address via CLI and ping other services locally
- Start your NodeRed editor



Access the console of the container in the swarm stack

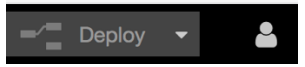
Start NodeRed and publish data & optionally store it again into your InfluxDB (define user etc.)
Note: In NodeRed use service name for addressing not the IP address



Add Authentication to node red

- ***Option 1*: Modify setting.js file to define username&password**
- Go to your NodeRed volume and download setting.js, go to // securing node red section in this file
- Uncomment this string including the users key value which has one item defined as a JSON Object
  - JSON Objects defines: Username, password and permissions
  - We can generate the hash of the password via a hashfunction:
    `password: require('bcryptjs').hashSync("cloud2021")`
    Libary bccrpyt includes the hashSync function which returns the hash value of the password
  - Rename old setting.js to setting.js.backup and upload modified file including the password
  - Go to your stack, choose NodeRed and update it => NodeRed will be restartet and reads the settings.js file
  - Login with your username and password
  - **Don‘t forget to logout in NodeRed on the upper right corner!**

```
// Securing Node-RED
// ----------------
adminAuth: {
   type: "credentials",
   users: [{
      username: "admin",
      password: require('bcryptjs').hashSync("cloud2021"),
       permissions: "*“
    }]
},
```

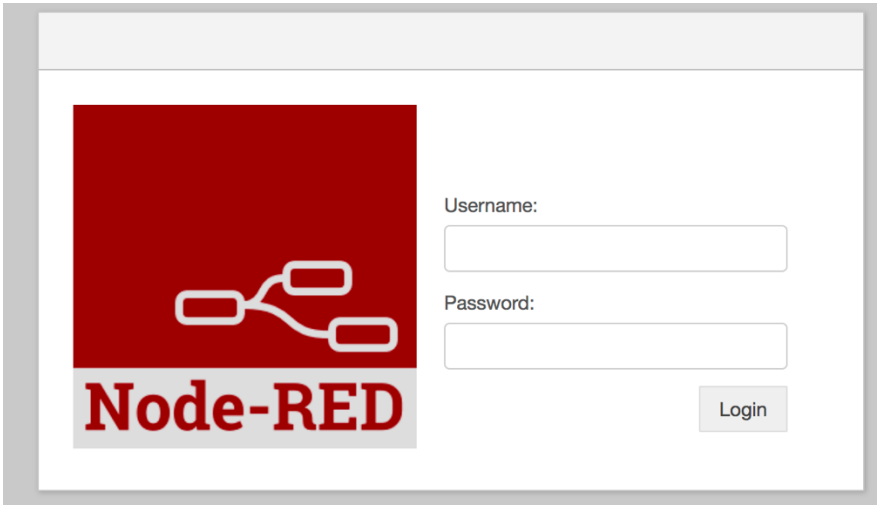


- **Option 2: Using an environment Variables to define username & pw**
  - Environment variables exist at the runtime of the container
  - Modify your deployment stack file (YAML) and define username and password
  - Update your settings.js file to read the new environment variable:
    - Go to your nodered volume
    - Download settings.js file
    - Rename the original setting.js file into setting.js.backup
    - Modify it according to the slide on the next page & upload it again
    - Restart the NodeRed app and login with the username and password to node red
    - Modify the password in the deployment file and test if it works!

```
version: "3.8"

services:
  nodered1:
    image: nodered/node-red:latest
    ports:
      - "50002:1880"
    environment:
      - NODERED_MY_PASSWORD=cloud2021
      - NODERED_MY_USERNAME=me
    networks:
      network2:
    volumes:
      - nodered1-v:/data
```

Modifying the file settings.js on volume


```
//All the way on the to in settings.js enter the following function:
my_function = ()=> {

    username = process.env.NODERED_MY_USERNAME
    password = process.env.NODERED_MY_PASSWORD

    return {
        username,
        password
    }
}

// Securing Node-RED
    // ----------------
    // To password protect the Node-RED editor and admin API, the following
    // property can be used. See http://nodered.org/docs/security.html for details.
    adminAuth: {
        type: "credentials",
        users: [{
            username: my_function()['username'],
            password: require('bcryptjs').hashSync(my_function()['password']),
            permissions: "*"
        }]
    },
```



References

- NodeRED: https://nodered.org/docs/getting-started/docker
- Mosquitto: https://hub.docker.com/_/eclipse-mosquitto
- InfluxDB: https://docs.influxdata.com/influxdb/v2/