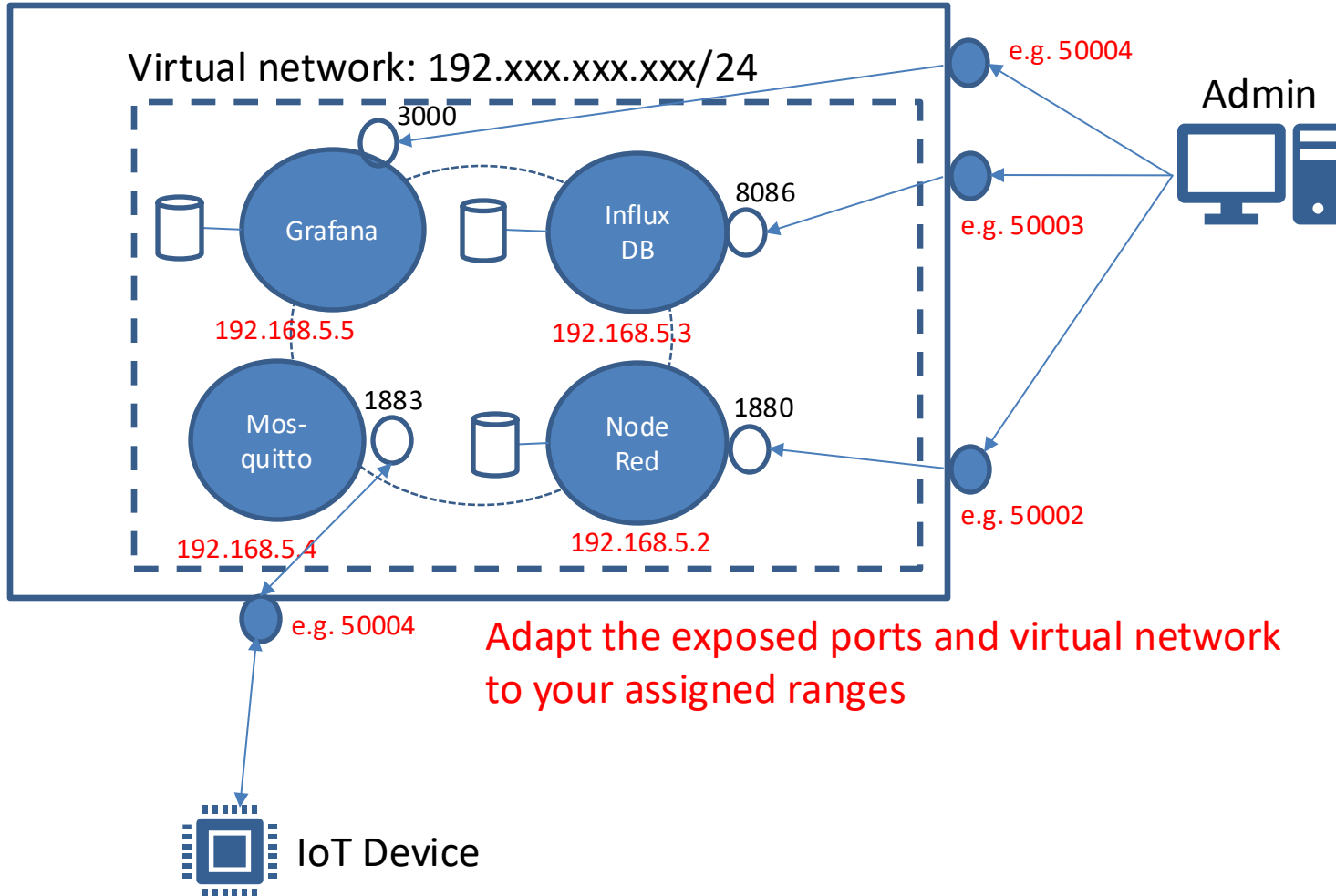



Endpoint on THM EI Cloud: e.g. `iot-lab01.ei.thm.de`

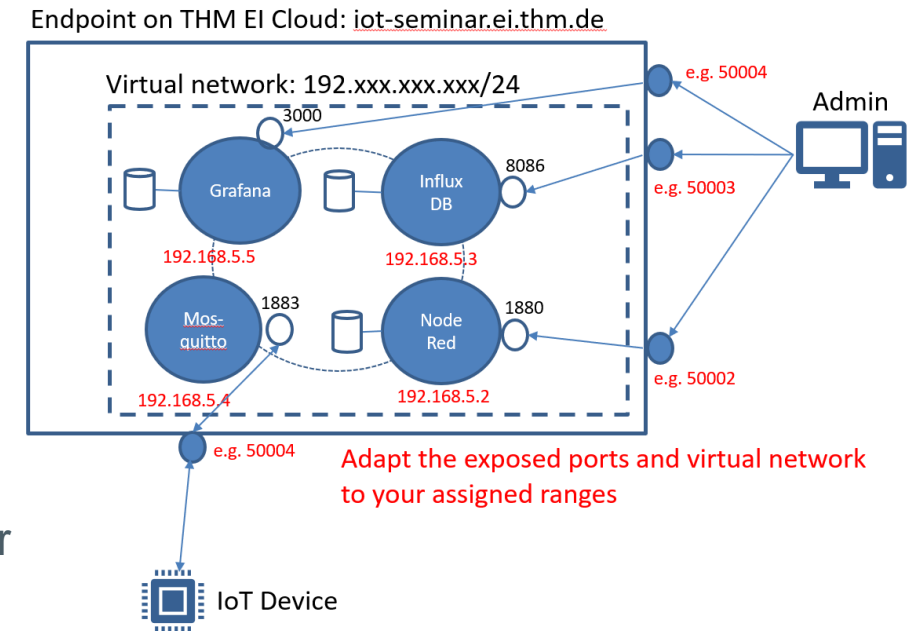


IoT Seminar Lab 2:

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

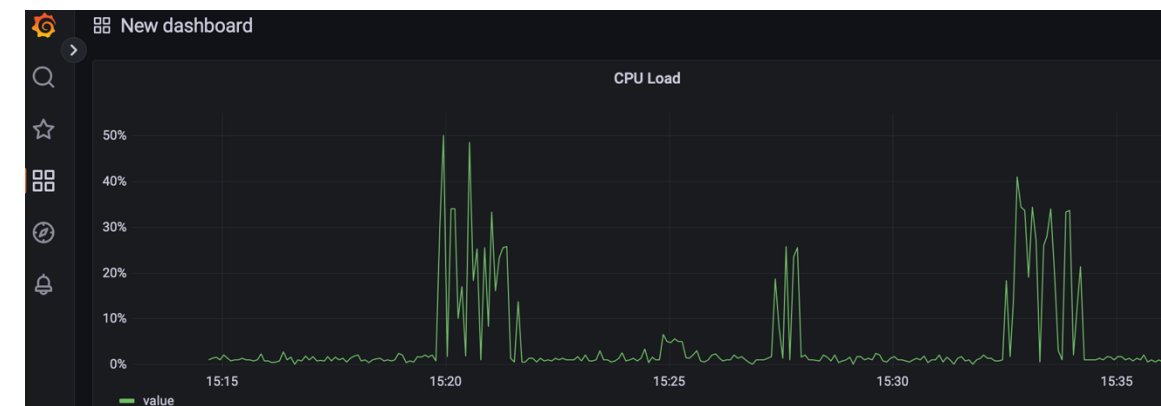
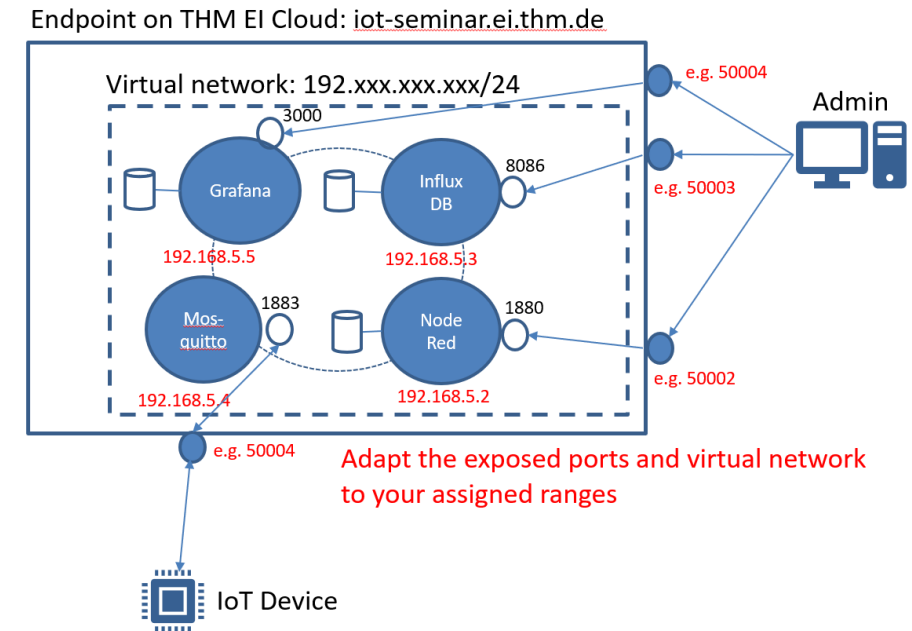


Steps in this Lab

- A.) Portainer GUI
 - 1.) Deploy NodeRed Container
 - 2.) Deploy Mosquitto Container
 - 3.) Configure and Test NodeRed and MQTT
 - 4.) Deploy 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)**

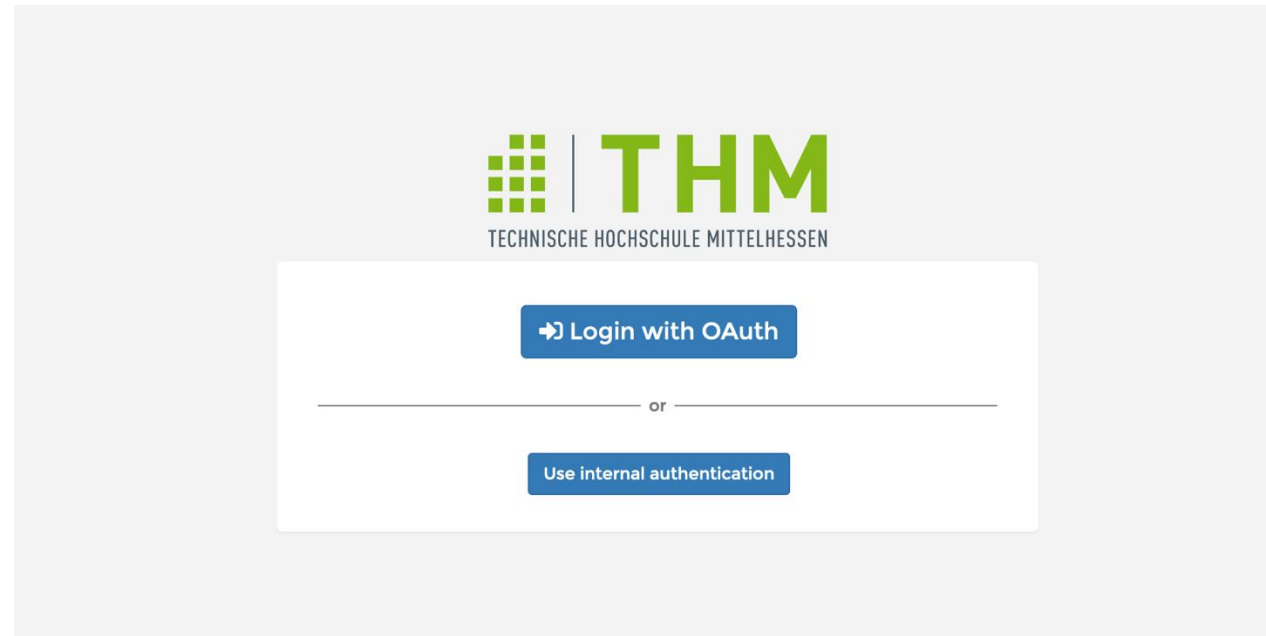
- C.) Use a YAML File (Docker compose)




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

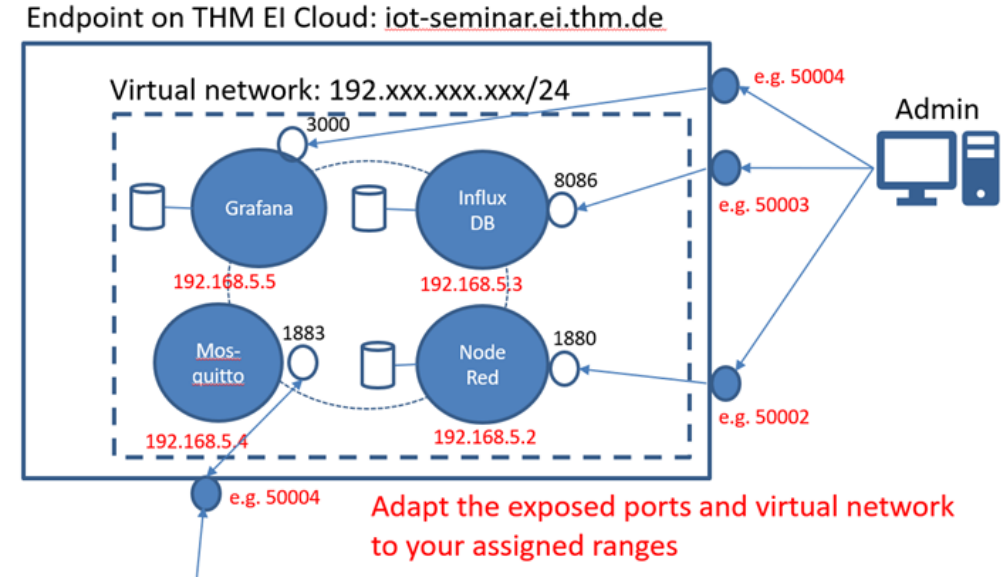
- Perform the steps above using
 - A.) Portainer GUI 
 - B.) Docker Command Line Interface (**done**)
 - C.) Docker manifest file: YAML

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!!!

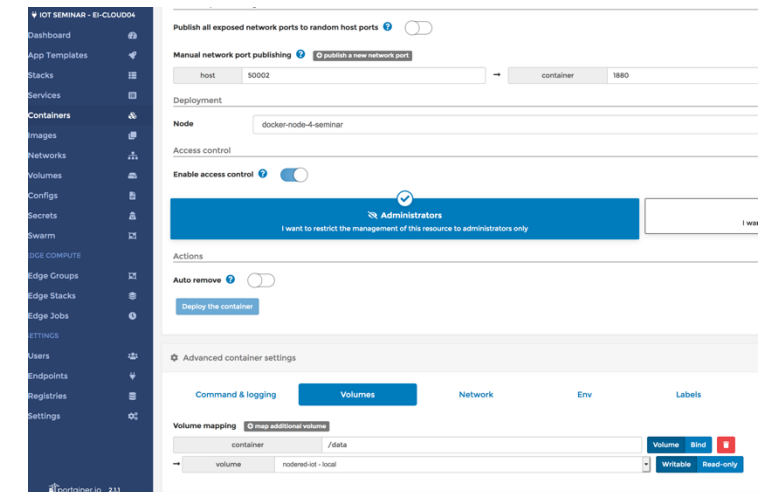
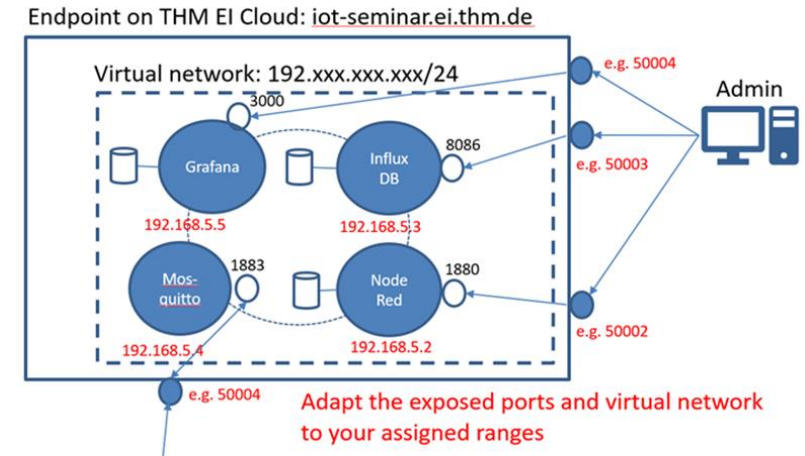
- name networks and volumes unique:
e.g. <yourname>_net and <yourname>_nodered_vol

IP Range	Port Range



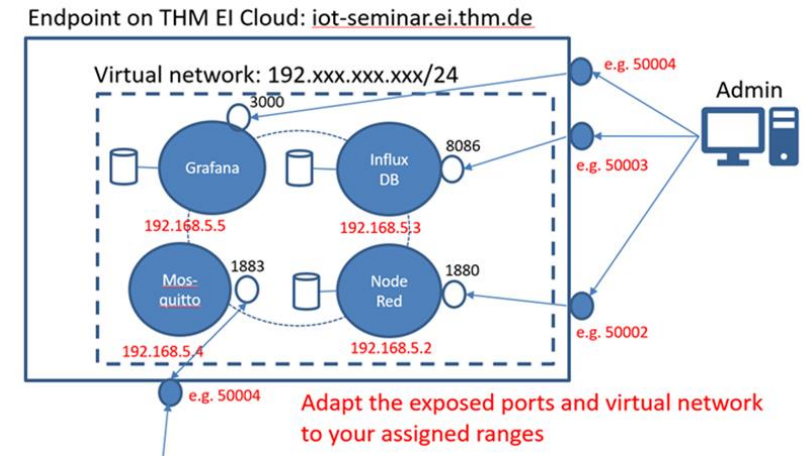
1.) Create your NodeRed Container


- **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.



Name ↓↑	State ↓↑	Filter	Quick Actions	Stack ↓↑	Image ↓↑	Created ↓↑	IP Address ↓↑	Host ↓↑	GPUs	Published Ports	Ownership ↓↑
<input type="checkbox"/> nodered_weber	starting		   	-	 nodered/node-red:latest	2022-09-14 10:36:36	192.168.9.2	iot-seminar	none	30300:1880	private

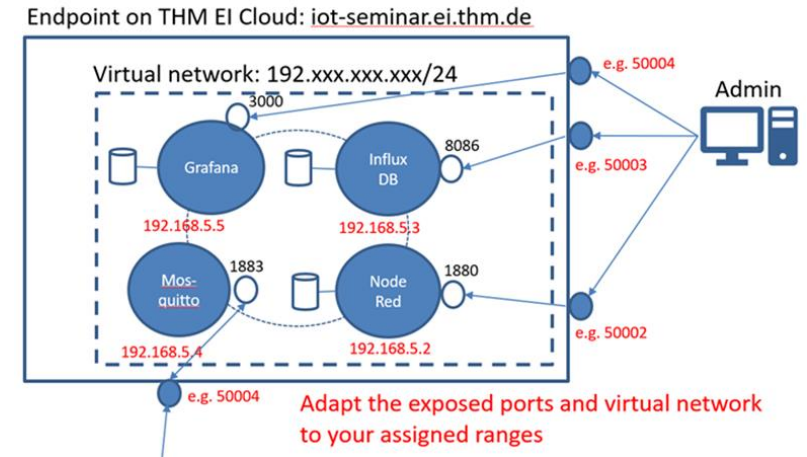
Logs Inspect Stats Console

Image

Access
NodeRed
Editor

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 container in the container list:



 mosquito-iotc

running



eclipse-mosquitto:latest

Name: Mosquitto

Image configuration

Registry: DockerHub

Image: docker.io eclipse-mosquitto:1.6.13

Advanced mode

Always pull the image: ☒

You are currently using a free account to pull images from DockerHub and will be limited to 200 pulls every 6 hours. Remaining pulls: 197/200

Network ports configuration

Publish all exposed network ports to random host ports: ☐

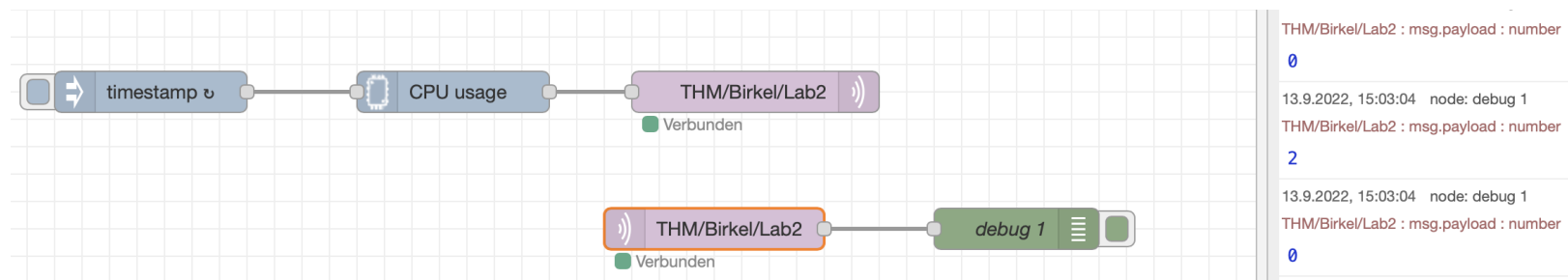
Manual network port publishing:

host	container	protocol
50004	1883	TCP

3.) Configure Node Red & MQTT

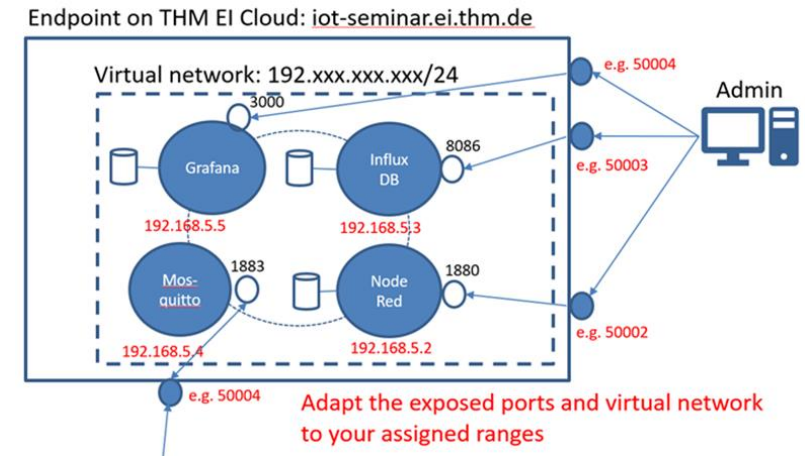
■ Open Node Red

- 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

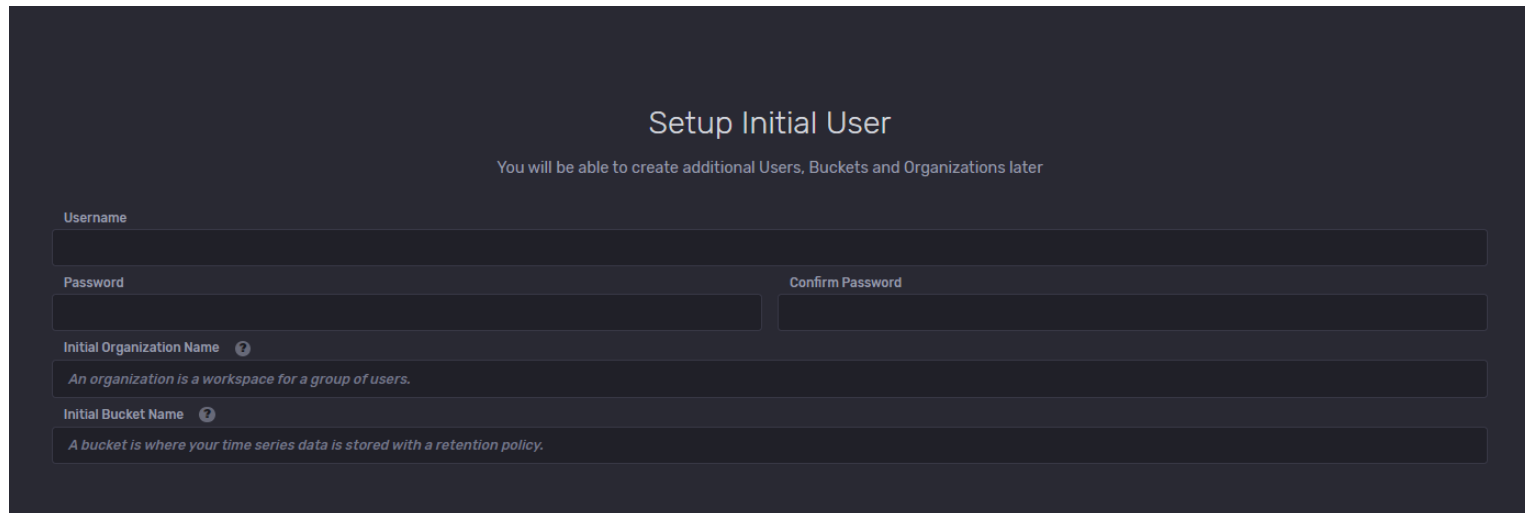
- **Deploy InfluxDB container in subnet and attach volume:**
Container: +Add container and configure the following items:
name, image `influxdb:latest` (Registry DockerHub),
+publish new network port: host 50xxx and container 8086
go down to Volumes: +map additional volume:
container path according to 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:



<input type="checkbox"/> InfluxDB_Birkel_Lab2	running		-	● influxdb:latest	2022-09-13 14:49:17	192.168.5.3	iot-seminar	none	50003:8086
---	---	---	---	--	---------------------	-------------	-------------	------	----------------------------

5.) Configure InfluxDB

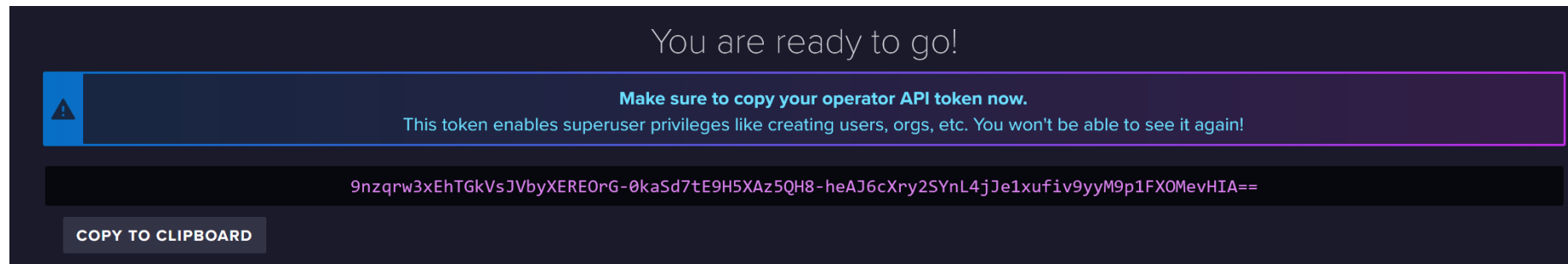
- **Open InfluxDB**
 - Walk through the Get Started guide and setup initial user
 - Choose a username and password and note it down
 - Choose the Organization Name “EI”
 - Name your data bucket



The screenshot shows the 'Setup Initial User' interface of InfluxDB. At the top, it says 'Setup Initial User' and 'You will be able to create additional Users, Buckets and Organizations later'. Below this, there are four input fields: 'Username', 'Password', 'Confirm Password', 'Initial Organization Name', and 'Initial Bucket Name'. Each field has a small question mark icon to its right. Below the 'Initial Organization Name' field, there is a hint: 'An organization is a workspace for a group of users.' Below the 'Initial Bucket Name' field, there is a hint: 'A bucket is where your time series data is stored with a retention policy.'

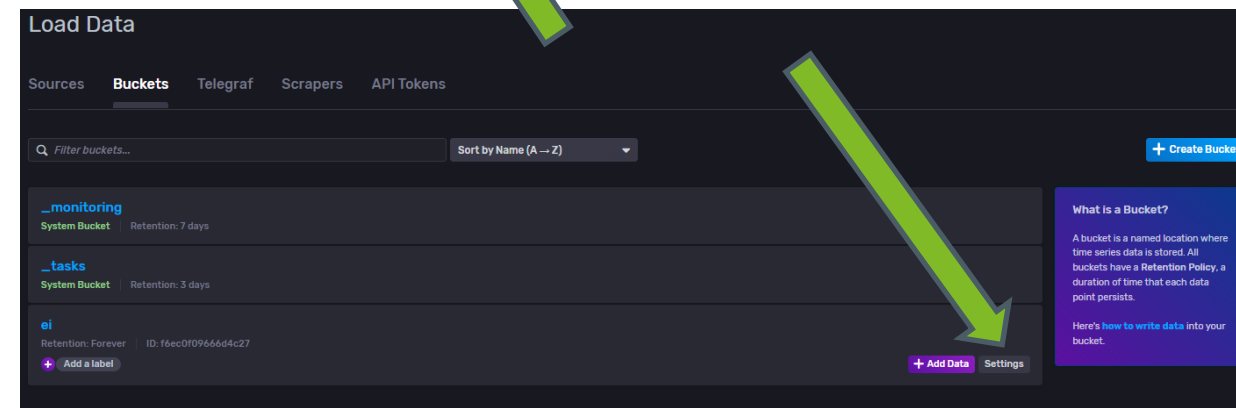
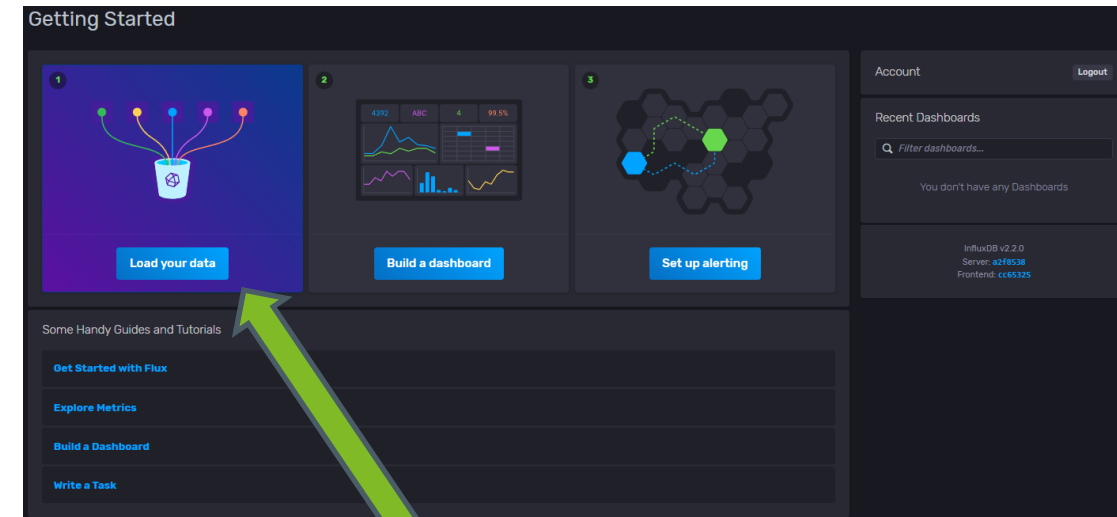
5.) Configure InfluxDB

- **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:



Node 'influxdb out' bearbeiten > Neuen Konfigurations-Node 'influxdb' hinzufügen

Abbrechen Hinzufügen

Eigenschaften

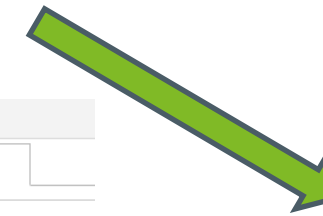
Name InfluxDB

Version 2.0

URL `http://iot-seminar.ei.thm.de:5xxxxx`

Token

☒ Verify server certificate



Node 'influxdb out' bearbeiten

Löschen Abbrechen Fertig

Eigenschaften

Name mymeasurement

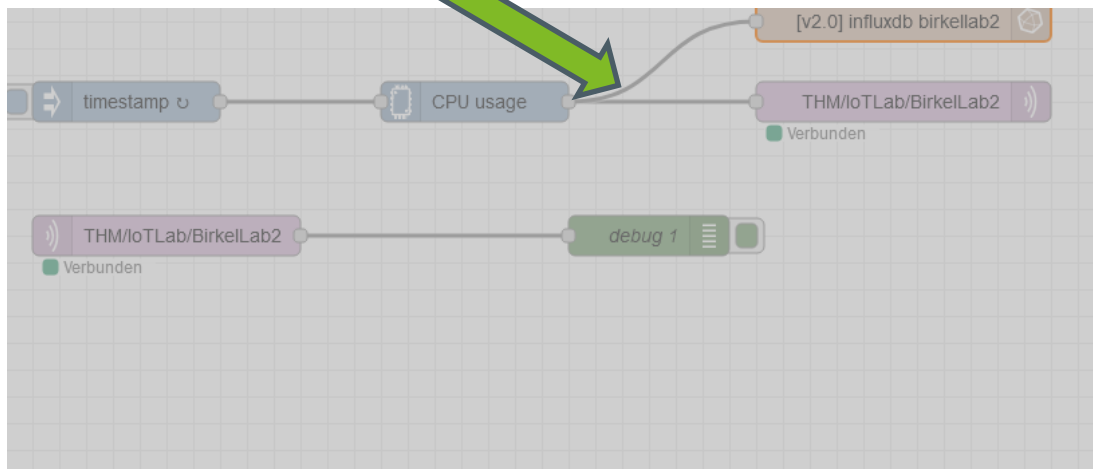
Server [v2.0] InfluxDB

Organization ei

Bucket ei

Measurement mymeasurement

Time Precision Milliseconds (ms)



Löschen

Eigenschaften

Name Name

Server [v2.0] influxdb

Organization ei

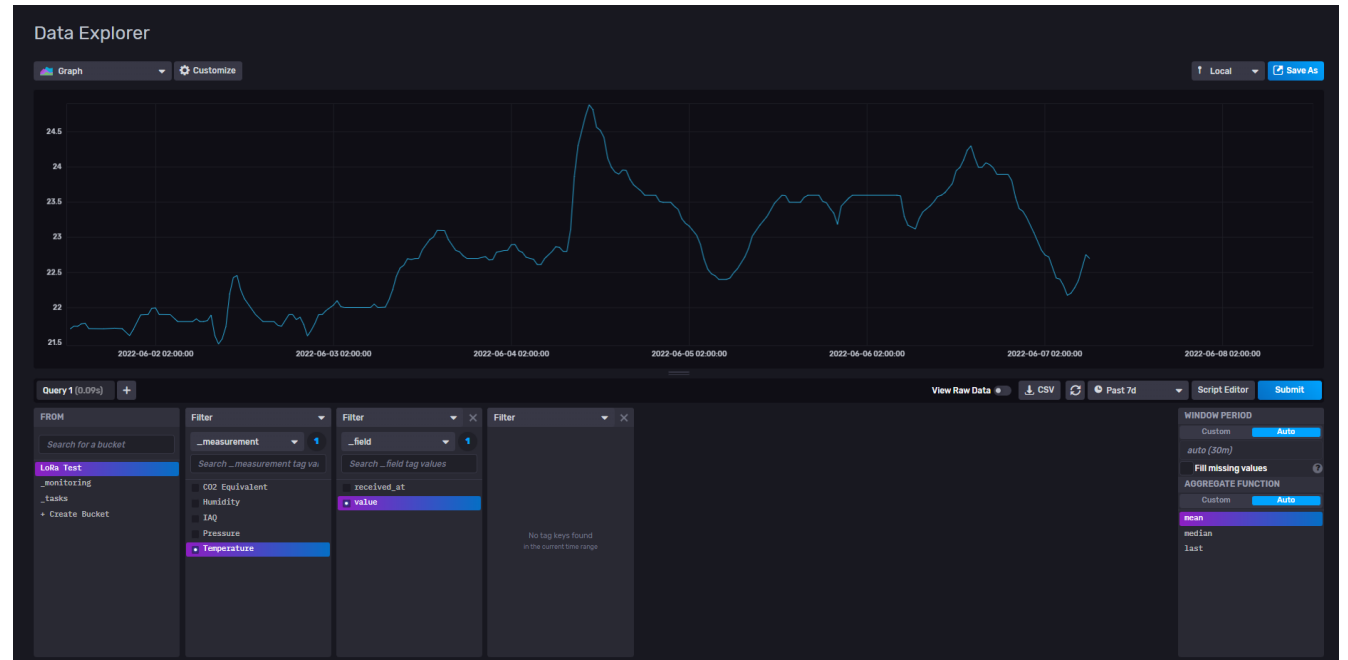
Bucket bucket

Measurement birkellab2

Time Precision Millisecond

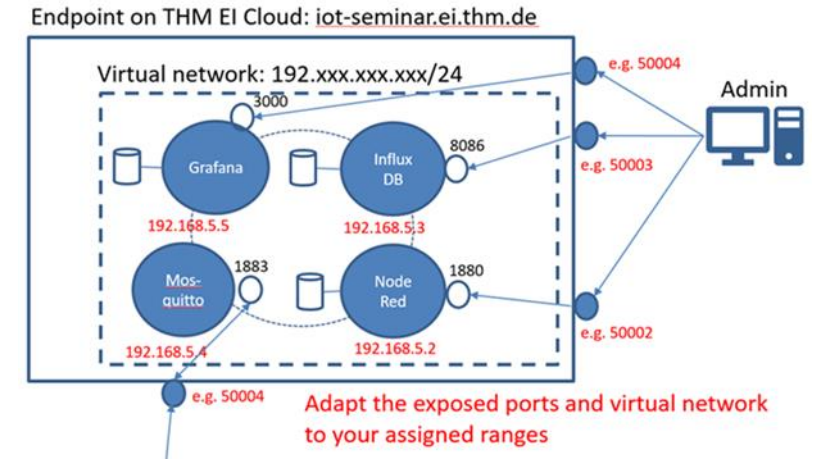
5.) Configure InfluxDB

- **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 following items:
name, image grafana/grafana:latest (Registry DockerHub),
+publish new network port: host 50xxx and container 3000
go down to Volumes: +map additional 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

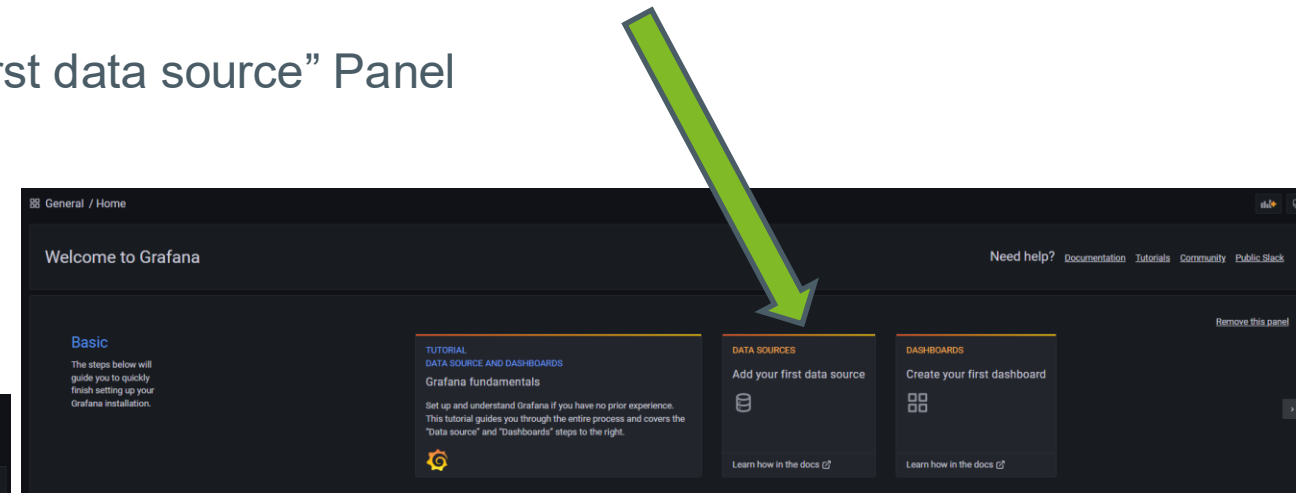
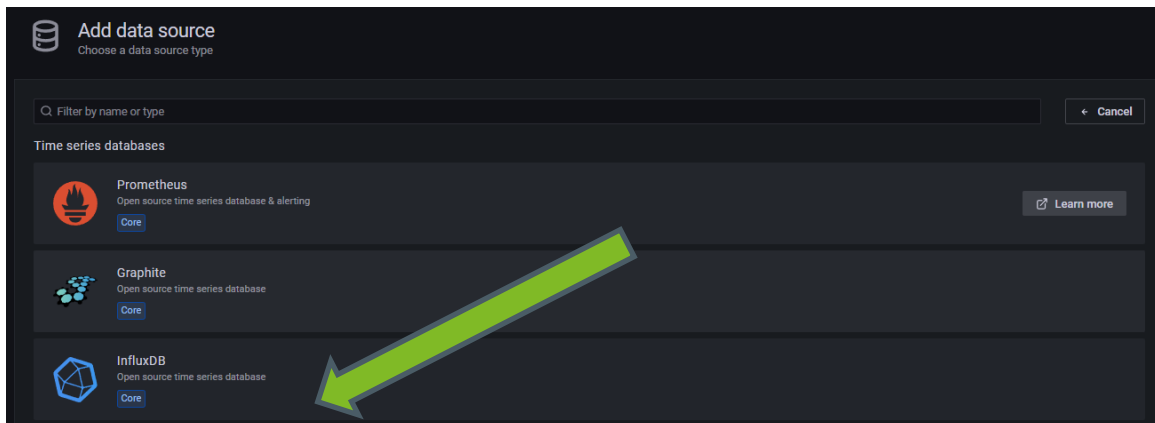
none

 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

InfluxDB Details

Organization	EI	
Token	configured	Reset
Default Bucket	Lab2	
Min time interval	10s	
Max series	1000	

Data Sources / InfluxDB
Type: InfluxDB

Settings

Name: InfluxDB Default ☒

Query Language
Flux

Support for Flux in Grafana is currently in beta
Please report any issues to:
<https://github.com/grafana/grafana/issues>

HTTP

URL: `http://iot-seminar.ei.thm.de:5xxxxx`

Access: Server (default) Help

Allowed cookies: New tag (enter key to add)

Timeout: Timeout in seconds

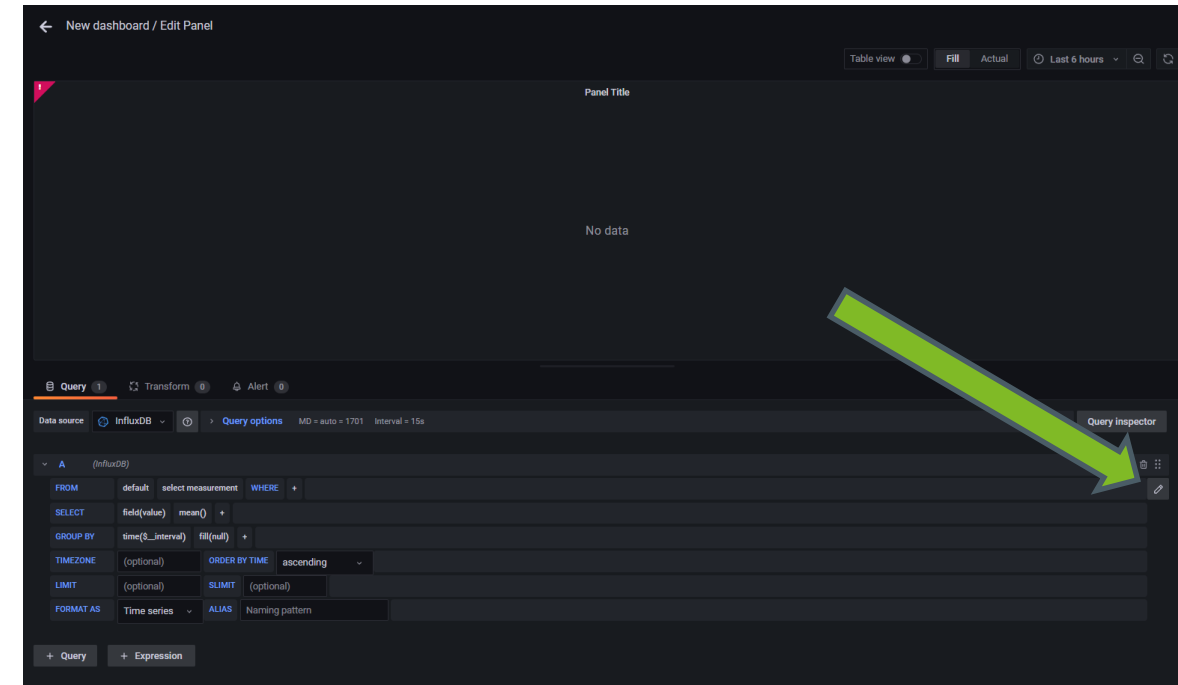
Basic Auth Details

User: user

Password: Password

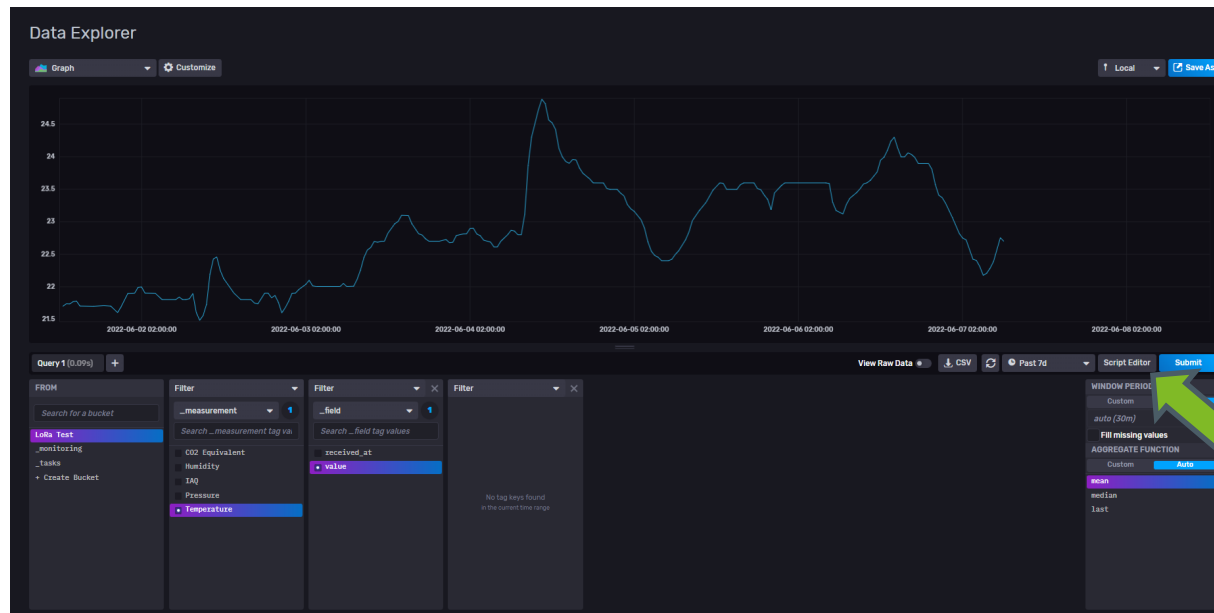
7.) Configure Grafana

- **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

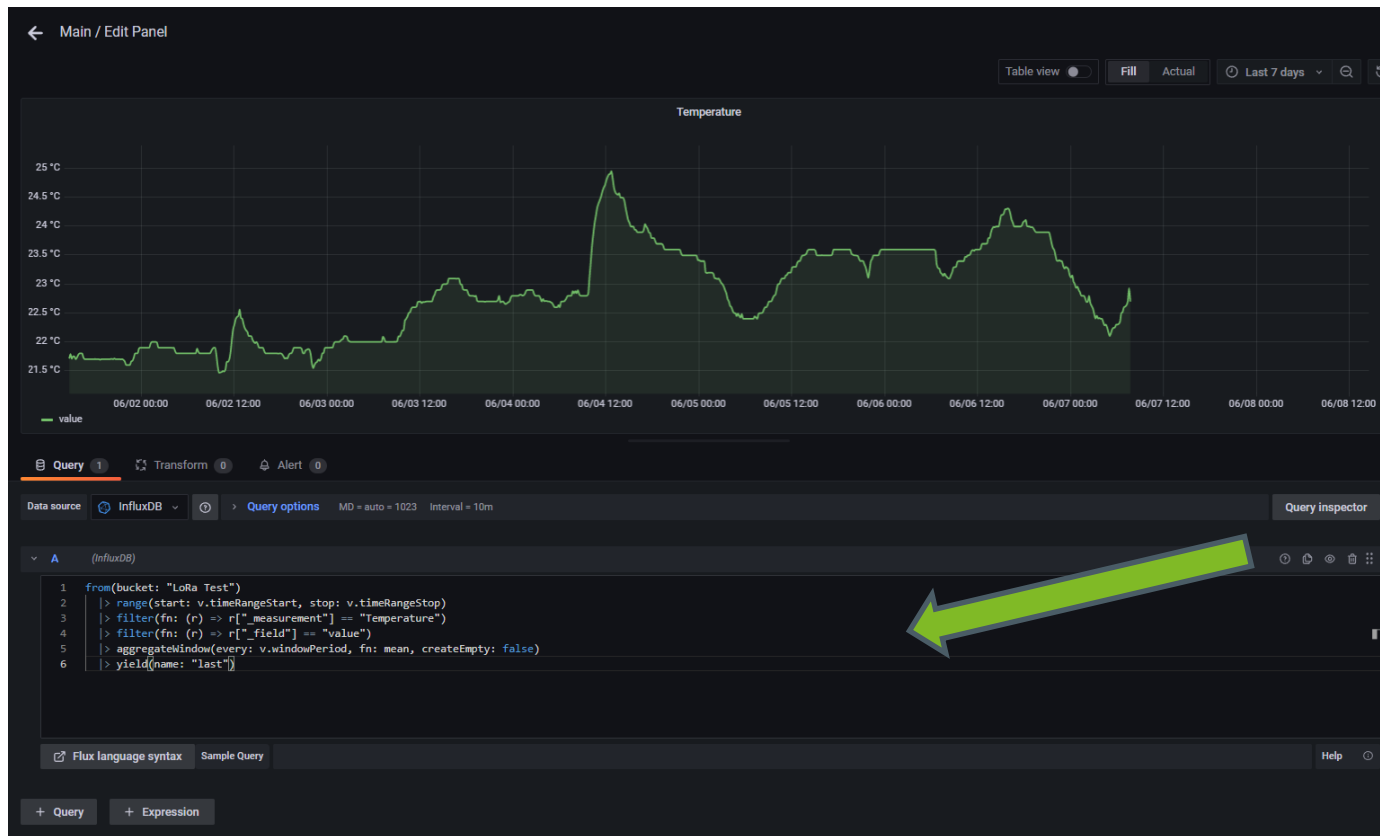
- **Go back to your InfluxDB Explorer**
 - Choose your bucket and a measurement you want to visualize (value)
 - Click on the script editor button
 - Copy the query text



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

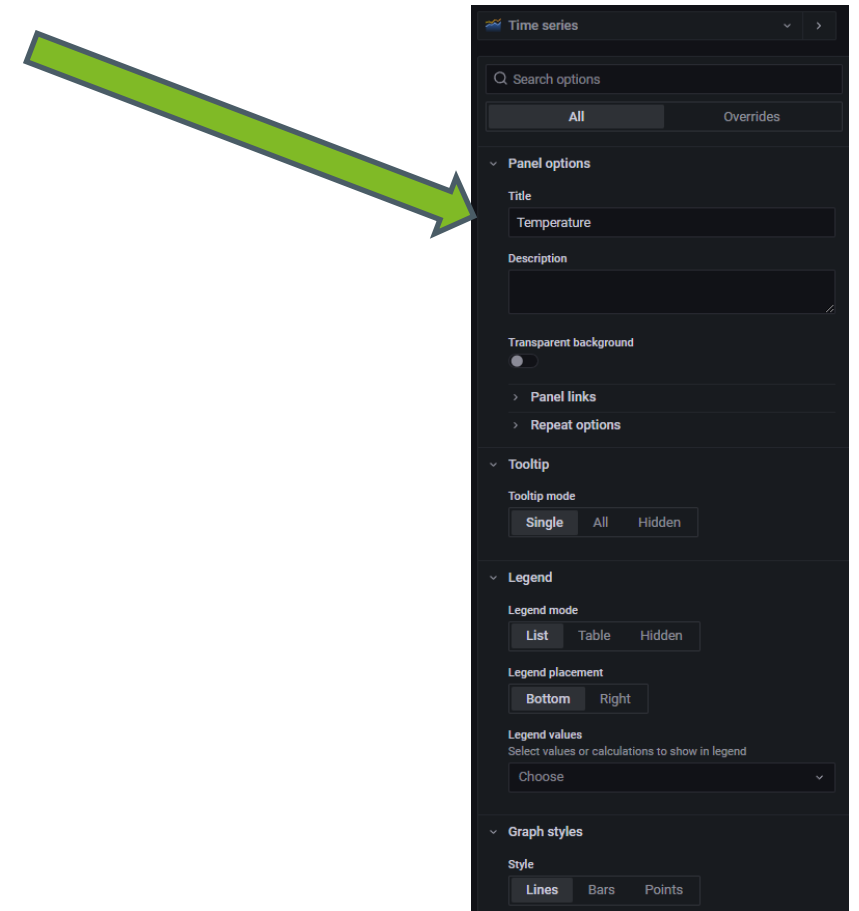
7.) Configure Grafana

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



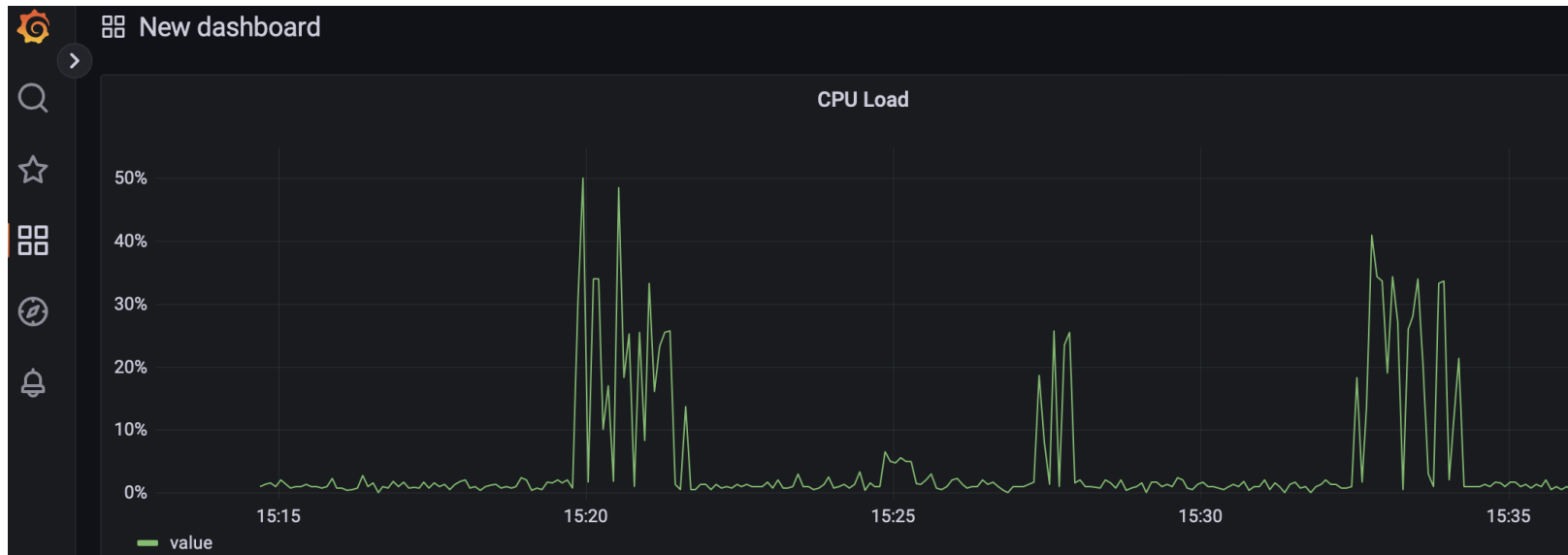
7.) Configure Grafana

- 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!



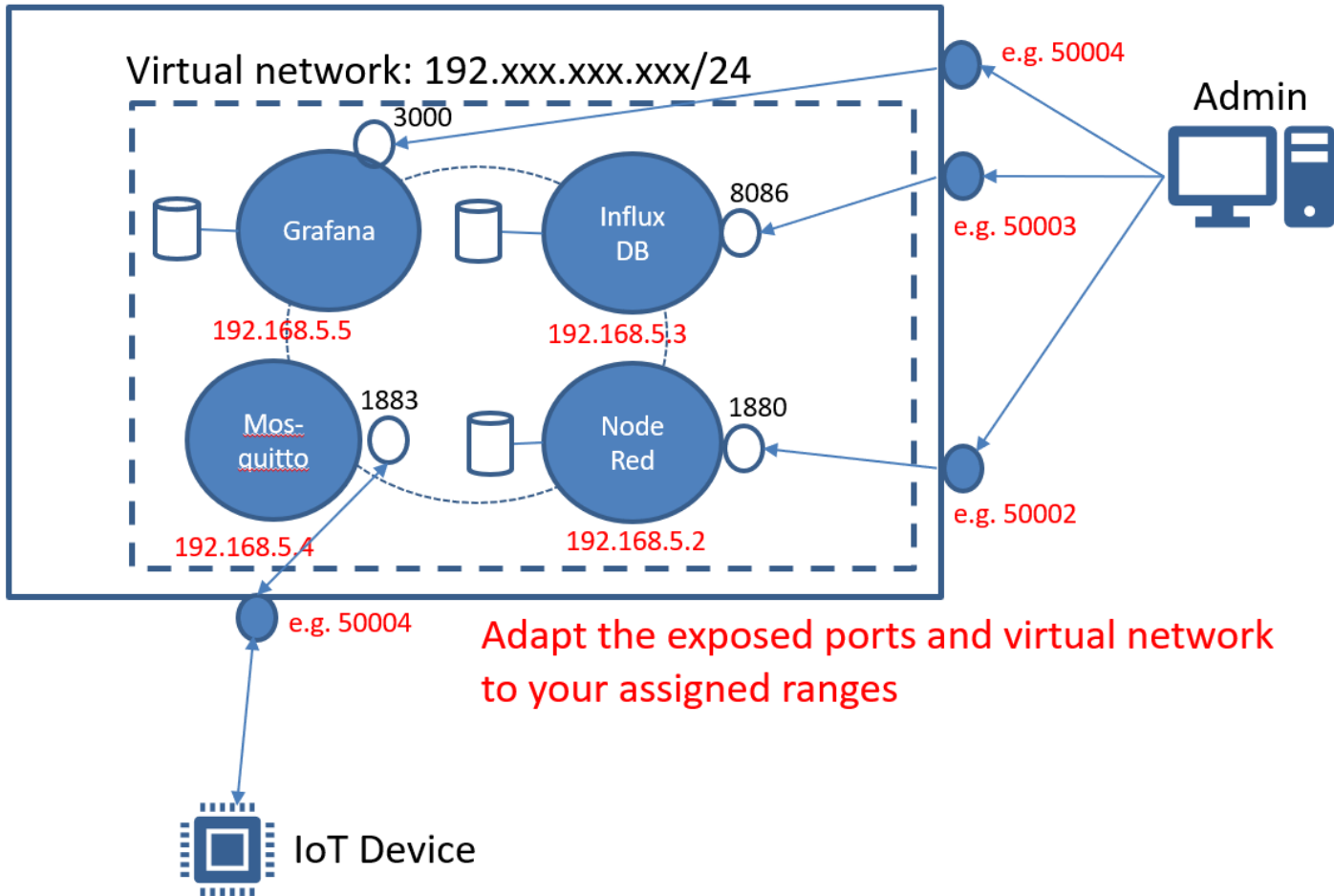
7.) Configure Grafana

- This is how your dashboard could look like:



8.) Send CPU Load from local docker client

Endpoint on THM EI Cloud: iot-seminar.ei.thm.de



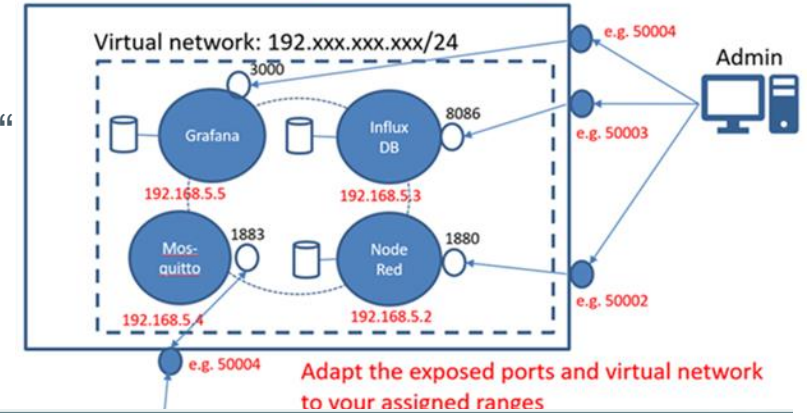
Check if your data is arriving using the MQTT Explorer or MQTTx



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**

Endpoint on THM EI Cloud: iot-seminar.ei.thm.de



+

Connections

Mosquitto in EI Cloud
mqtt://iot-seminar.ei.thm.de:50004/

test.mosquitto.org
mqtt://test.mosquitto.org:1883/

MQTT Connection

mqtt://iot-seminar.ei.thm.de:50004/

Name

Mosquitto in EI Cloud

Validate certificate

Encryption (tls)

Protocol

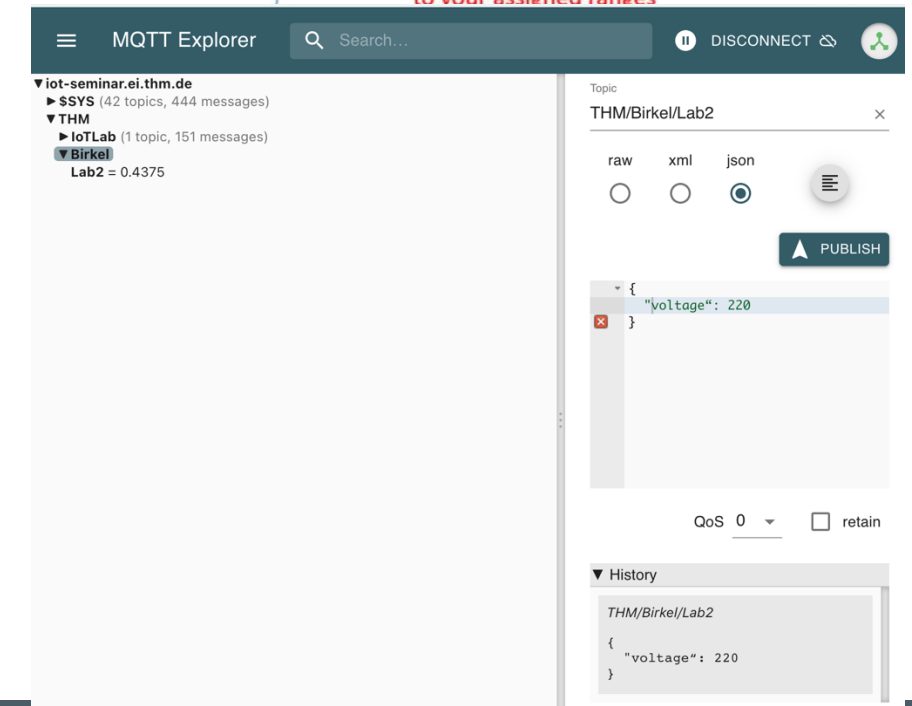
Host

Port

mqtt://

iot-seminar.ei.thm.de

50004



10.) Use Container Names instead of IP Address

- 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:

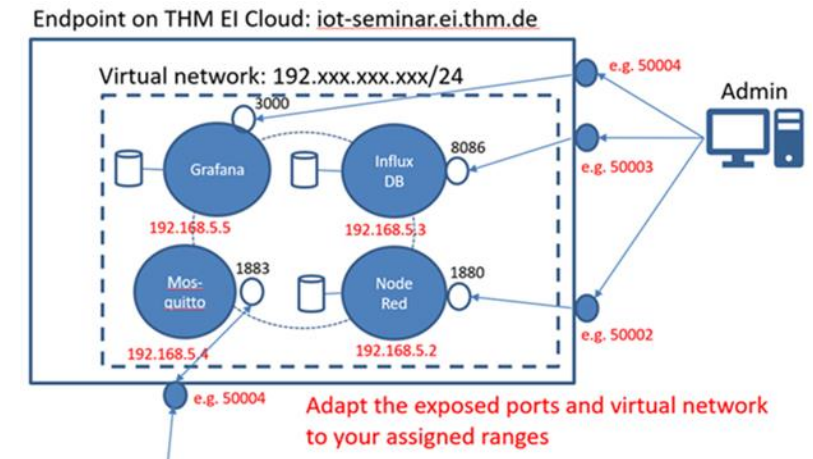
Container console

>_ Execute

Exec into container as default user using command bash

Disconnect

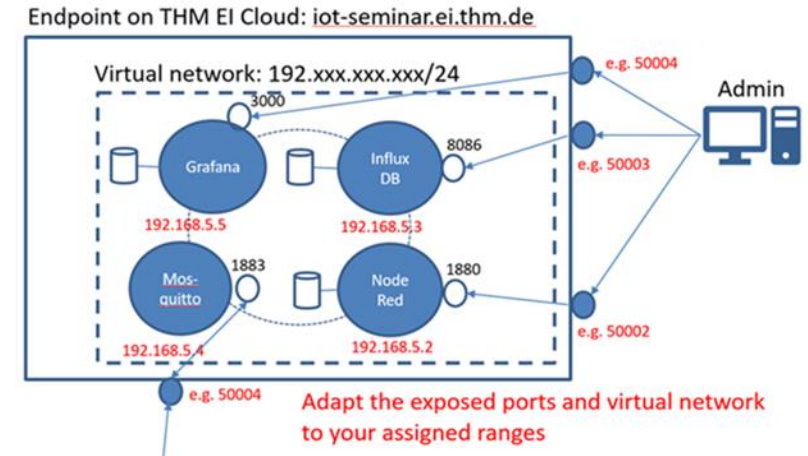
```
root@9e6170130f92:/# ping InfluxDB_Birkel_Lab2
PING InfluxDB_Birkel_Lab2 (192.168.5.3) 56(84) bytes of data.
64 bytes from 9e6170130f92 (192.168.5.3): icmp_seq=1 ttl=64 time=0.021 ms
64 bytes from 9e6170130f92 (192.168.5.3): icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 9e6170130f92 (192.168.5.3): icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 9e6170130f92 (192.168.5.3): icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 9e6170130f92 (192.168.5.3): icmp_seq=5 ttl=64 time=0.022 ms
^C
--- InfluxDB Birkel Lab2 ping statistics ---
```



<input type="checkbox"/>	InfluxDB_Birkel_Lab2	running	📄 🔍 📊 >_ 🔗
<input type="checkbox"/>	Lab2_Birkel_Grafana	running	📄 🔍 📊 >_ 🔗
<input type="checkbox"/>	Lab2_Birkel_Mosquitto	running	📄 🔍 📊 >_ 🔗

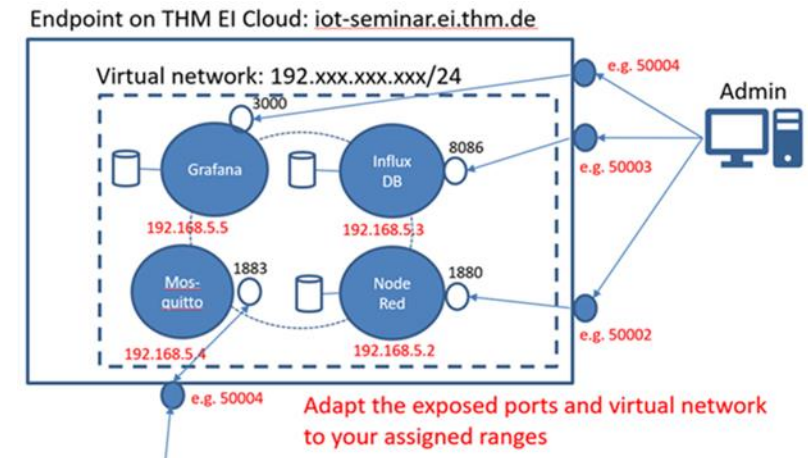
c.) Use Docker manifest file (YAML format)

- 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

Web editor

You can get more information about Compose file format in the [official documentation](#).

Remember **first level keywords**

```
1 version: "3.7"
2
3 services:
4
5
6
7
8
9 networks:
10
11 volumes:
12
13 configs:
14
15 secrets:
16
17
```


YAML File Format



How to write a YAML file:

- No Spaces
- No Upper Case Letters
- Use either Tab key or 2 spaces

Tab key or 2
spaces

1st level
2nd level
3rd level

```
1 versions: "3.8"
2
3 services:
4   nodered:
5     image: nodered/node-red:latest
6     ports:
7       "6008:1880"
8   database:
9     image: bitnami/mongodb:latest
10
11
12 networks:
```




Create a new swarm stack

Docker engine release

Endpoints

Refresh

Search by name, group, tag, status, URL...

	FlexQuartier - ei-cloud01 up 2020-09-08 16:16:20 12 stacks 28 services 92 containers - 30 62 / 4 cloud	Group: Unassigned Swarm 19.03.5 + Agent tasks.agent:9001
	IoT Seminar - ei-cloud04 up 2020-09-08 16:16:20 2 stacks 4 services 7 containers - 4 3 / 1 0 6 volumes 5 images cloud	Group: Unassigned Swarm 19.03.8 + Agent ei-cloud04.ei.thm.de:9001
	Industrie4.0 - ei-cloud03 up 2020-09-08 16:16:21 4 stacks 9 services 9 containers - 9 0 / 1 0 13 volumes 51 images cloud	Group: Unassigned Swarm 19.03.8 + Agent ei-cloud03.ei.thm.de:9001

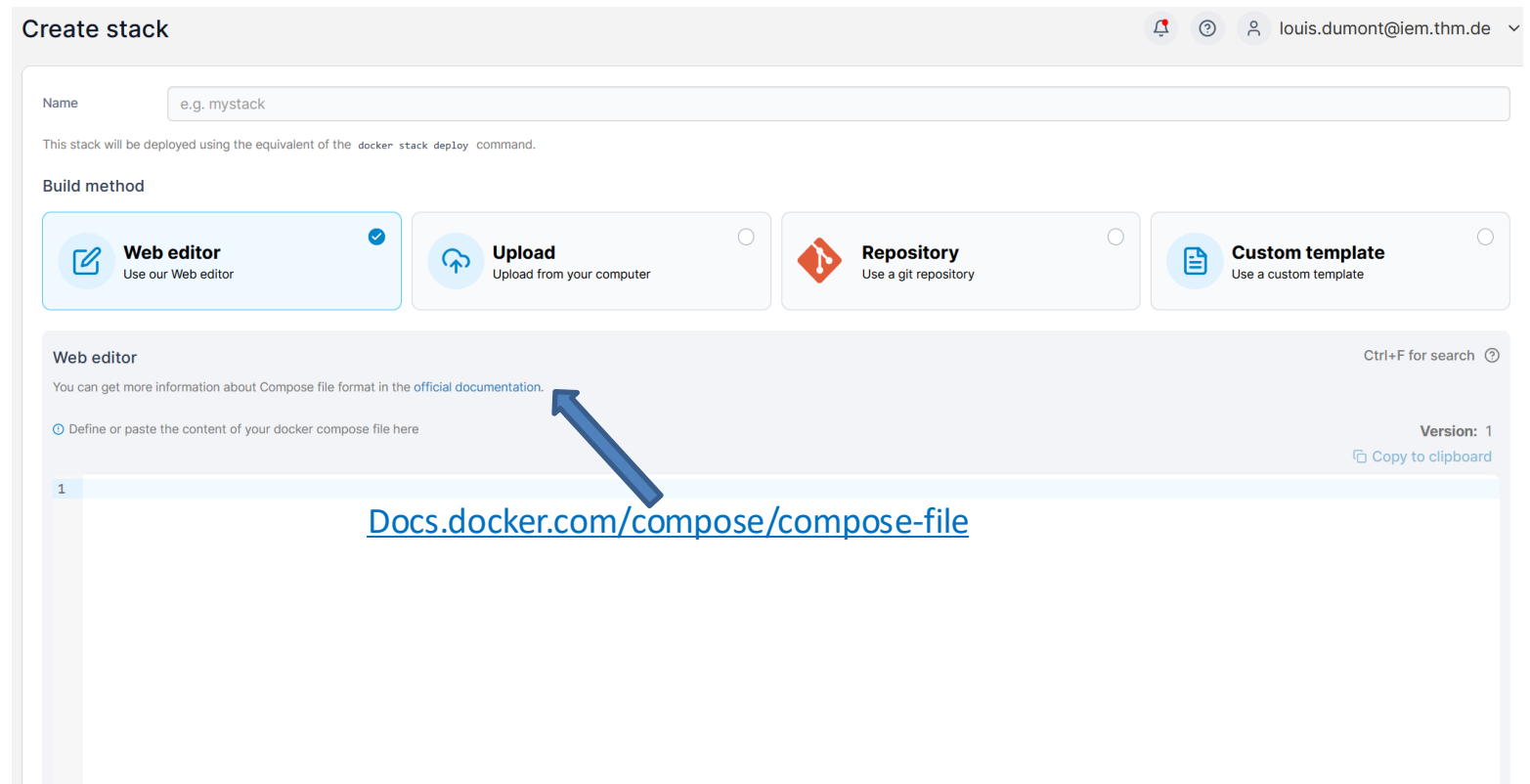
Go to the "IoT seminar" endpoint (should only see that endpoint)



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 documentation
- Example :

Compose File Format 3.8 supports
Docker Engine release 19.03.0+



Create stack

Name: e.g. mystack

This stack will be deployed using the equivalent of the `docker stack deploy` command.

Build method

- Web editor** (selected) - Use our Web editor
- Upload - Upload from your computer
- Repository - Use a git repository
- Custom template - Use a custom template

Web editor

You can get more information about Compose file format in the [official documentation](https://docs.docker.com/compose/compose-file).

Define or paste the content of your docker compose file here

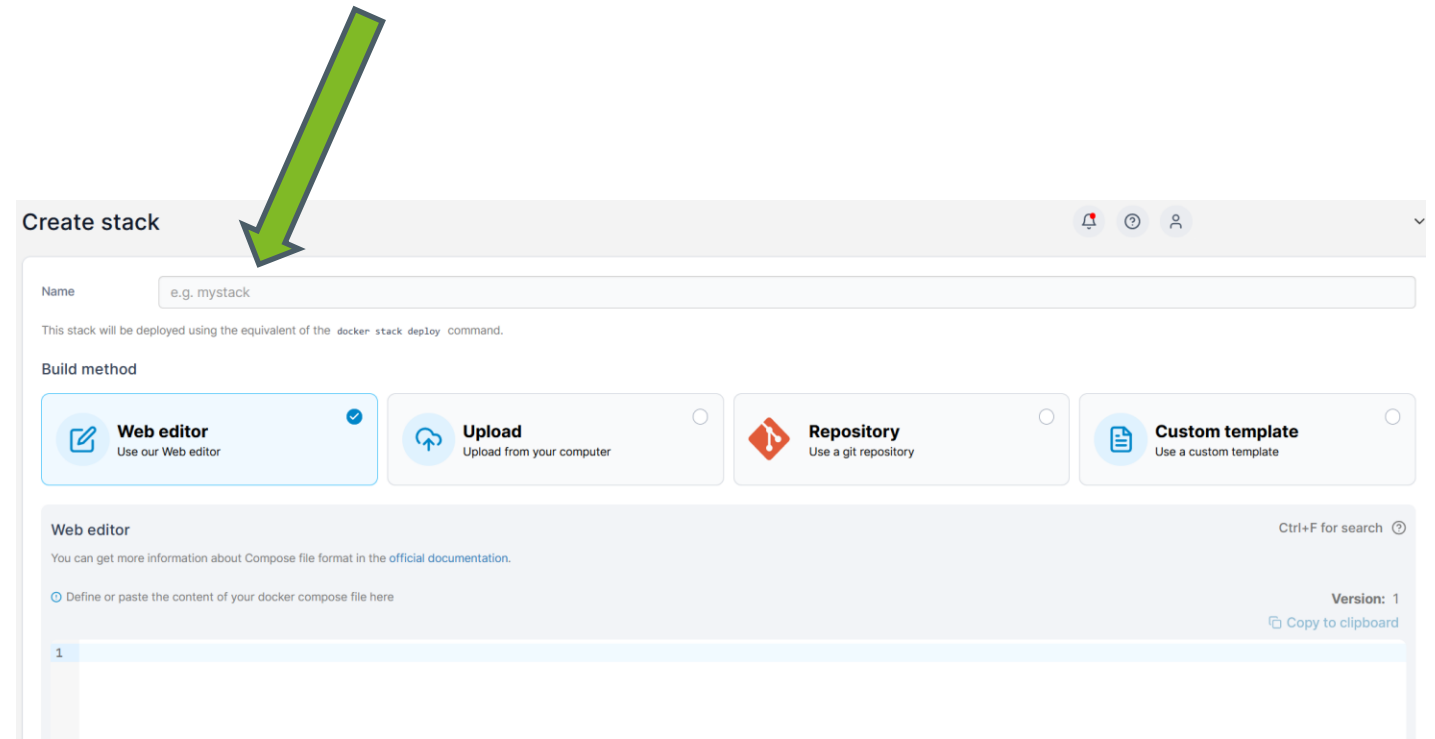
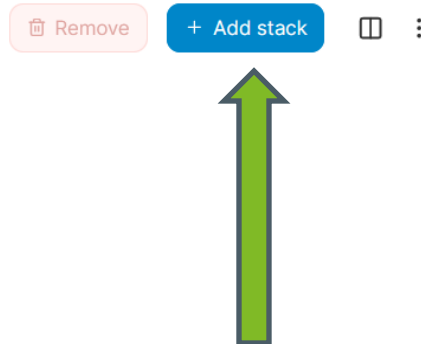
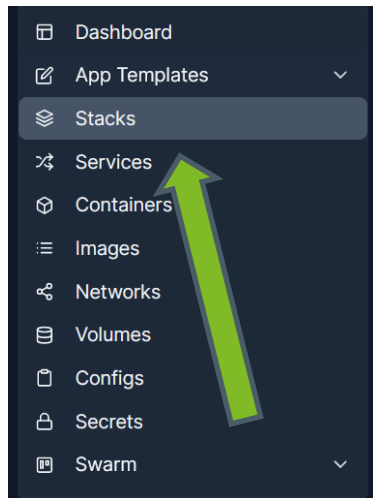
Version: 1

[Docs.docker.com/compose/compose-file](https://docs.docker.com/compose/compose-file)

Copy to clipboard

b.) Create a new stack

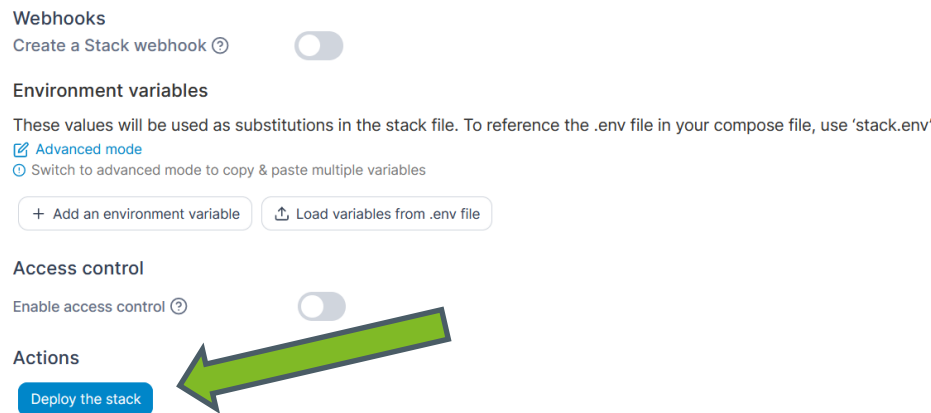
- **Login** to your endpoint
 - Head to “Stacks” and click “Add stack”
 - Choose <yourname>_lab2_ws2223 as the name of your stack
 - Use the web editor to configure your services



b.) Create a new stack

Docker-compose:

- **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 !!!



```
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



THM

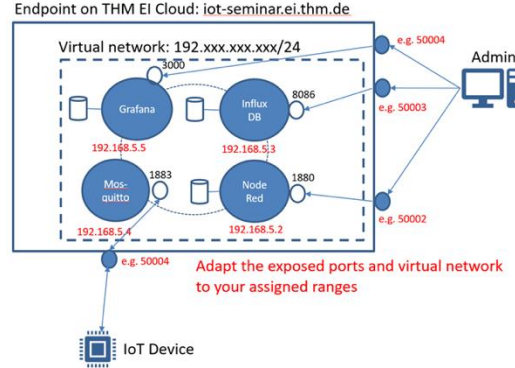
CAMPUS
GIESSEN

EI

Elektro- und
Informationstechnik

You can get more information about Compose file format in the [official documentation](#).

```
1 version: "3.8"
2
3 services:
4   nodered1:
5     image: nodered/node-red:latest
6     ports:
7       - "50002:1880"
8     networks:
9       network1:
10        ipv4_address: 192.168.5.2
11     volumes:
12       - nodered1-v:/data
13   mosquitto1:
14     image: eclipse-mosquitto:1.6.3
15     ports:
16       - "50004:1883"
17     networks:
18       network1:
19        ipv4_address: 192.168.5.4
20   mongodb1:
21     image: bitnami/mongodb:latest
22     ports:
23       - "50003:27017"
24     networks:
25       network1:
26        ipv4_address: 192.168.5.3
27
28 networks:
29   network1:
30     attachable: true # define if containers can communicate
31     ipam:
32       config:
33         - subnet: 192.168.5.0/24
34
35 volumes:
36   nodered1-v:
37   mongodb1-v:
```



Modify your stack:

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

```
version: "3.8"
services:
  node-red-weber:
    image: nodered/node-red:latest
    ports:
      - "30300:1880"
    networks:
      - weber_net
    volumes:
      - weber_nodered_vol:/data
  mosquitto-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:

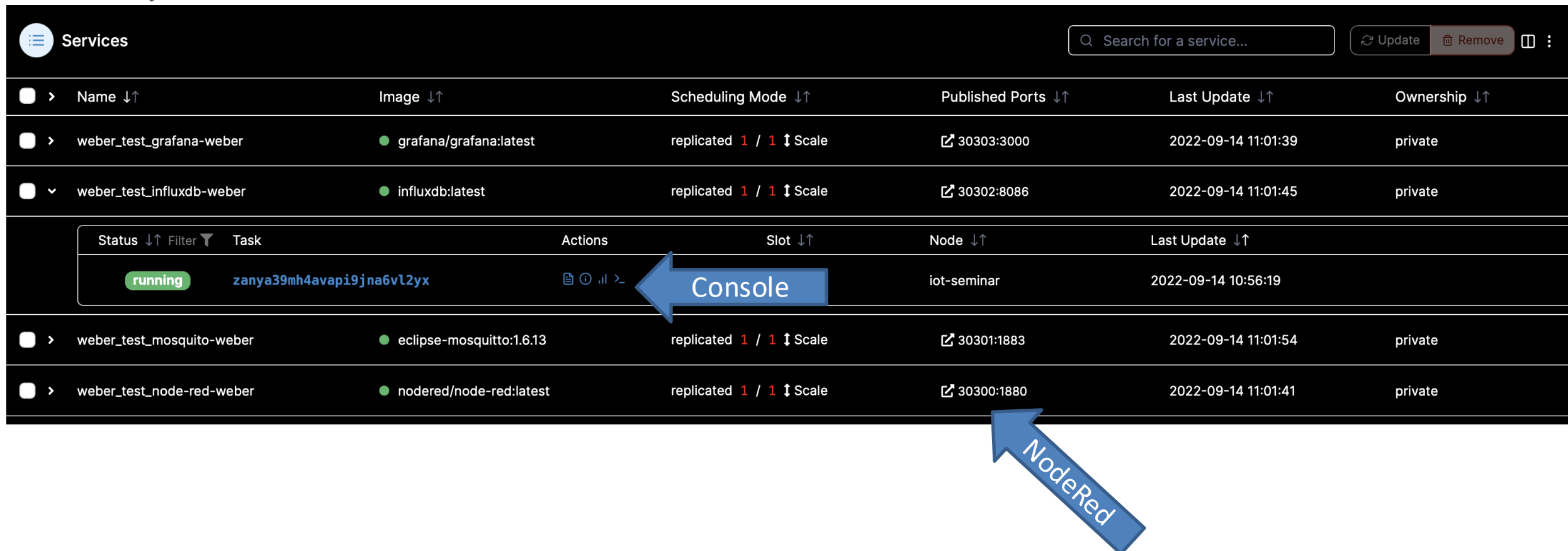
networks:
  weber_net:
```

Portainer GUI: Container inspect










```
▼ Networks:
  ► ingress: { Aliases: 6e0e29e8c4b8, DriverOpt
  ▼ my_cloud_app_ub_network2:
    ► Aliases: [ 6e0e29e8c4b8 ]
    DriverOpts:
      EndpointID: 5fafcb5f818e02936c2c
      Gateway:
        GlobalIPv6Address:
        GlobalIPv6PrefixLen: 0
    ► IPAMConfig: { IPv4Address: 10.0.59.8 }
      IPAddress: 10.0.59.8
      IPPrefixLen: 24
      IPv6Gateway:
      Links:
      MacAddress: 02:42:0a:00:3b:08
      NetworkID: 48gfp9tob668u2zn6ufj
```

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



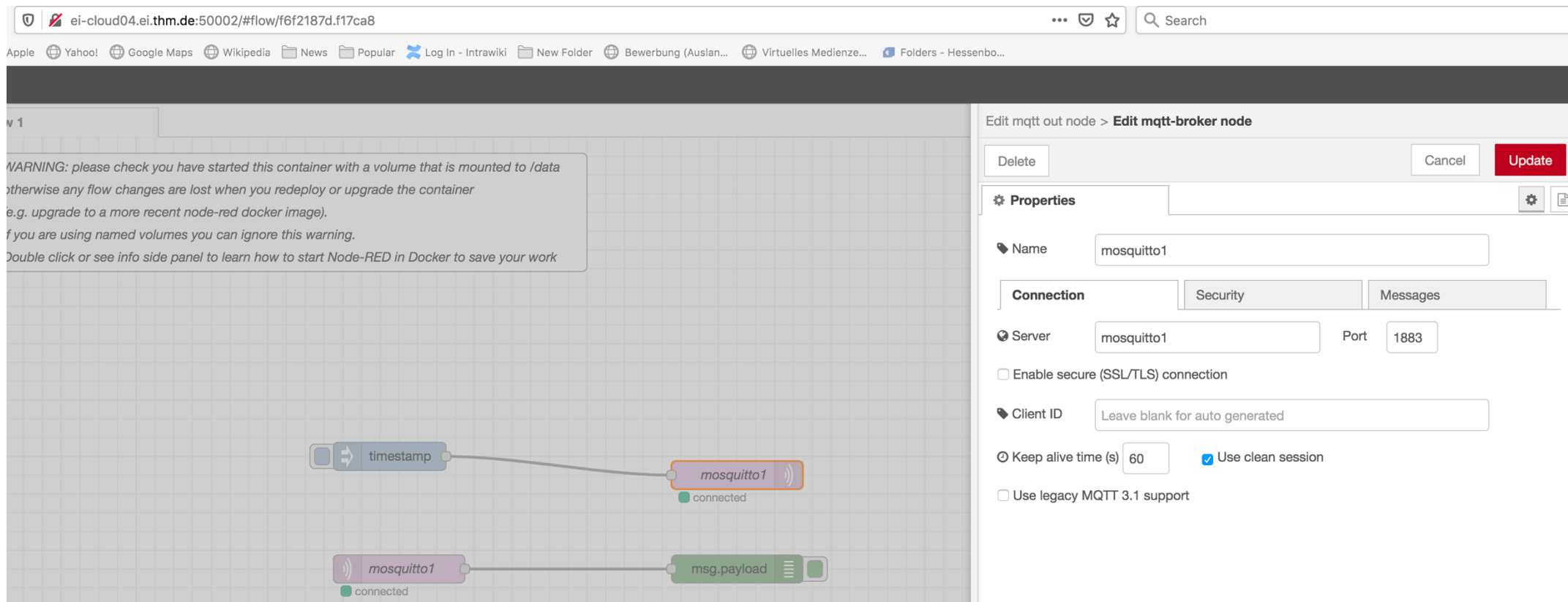
The screenshot shows the Docker Swarm 'Services' page. It lists several services: 'weber_test_grafana-weber', 'weber_test_influxdb-weber', 'weber_test_mosquito-weber', and 'weber_test_node-red-weber'. A task for 'weber_test_influxdb-weber' is expanded, showing a 'running' status and a task ID 'zanya39mh4avapi9jna6vl2yx'. A blue arrow labeled 'Console' points to the task's actions menu. Another blue arrow labeled 'NodeRed' points to the 'weber_test_node-red-weber' service's published ports.

Name ↓↑	Image ↓↑	Scheduling Mode ↓↑	Published Ports ↓↑	Last Update ↓↑	Ownership ↓↑												
weber_test_grafana-weber	grafana/grafana:latest	replicated 1 / 1 ↑ Scale	30303:3000	2022-09-14 11:01:39	private												
weber_test_influxdb-weber	influxdb:latest	replicated 1 / 1 ↑ Scale	30302:8086	2022-09-14 11:01:45	private												
<div>Status ↓↑ Filter ▾ Task</div> <table border="1"><thead><tr><th>Status ↓↑</th><th>Task</th><th>Actions</th><th>Slot ↓↑</th><th>Node ↓↑</th><th>Last Update ↓↑</th></tr></thead><tbody><tr><td>running</td><td>zanya39mh4avapi9jna6vl2yx</td><td>  </td><td></td><td>iot-seminar</td><td>2022-09-14 10:56:19</td></tr></tbody></table>						Status ↓↑	Task	Actions	Slot ↓↑	Node ↓↑	Last Update ↓↑	running	zanya39mh4avapi9jna6vl2yx	  		iot-seminar	2022-09-14 10:56:19
Status ↓↑	Task	Actions	Slot ↓↑	Node ↓↑	Last Update ↓↑												
running	zanya39mh4avapi9jna6vl2yx	  		iot-seminar	2022-09-14 10:56:19												
weber_test_mosquito-weber	eclipse-mosquitto:1.6.13	replicated 1 / 1 ↑ Scale	30301:1883	2022-09-14 11:01:54	private												
weber_test_node-red-weber	nodered/node-red:latest	replicated 1 / 1 ↑ Scale	30300:1880	2022-09-14 11:01:41	private												

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



ei-cloud04.ei.thm.de:50002/#flow/f6f2187d.f17ca8

Apple Yahoo! Google Maps Wikipedia News Popular Log In - IntraWiki New Folder Bewerbung (Auslan... Virtuelles Medienze... Folders - Hessenbo...

WARNING: please check you have started this container with a volume that is mounted to /data otherwise any flow changes are lost when you redeploy or upgrade the container (e.g. upgrade to a more recent node-red docker image). If you are using named volumes you can ignore this warning. Double click or see info side panel to learn how to start Node-RED in Docker to save your work

1

Edit mqtt out node > Edit mqtt-broker node

Delete Cancel Update

Properties

Name mosquitto1

Connection Security Messages

Server mosquitto1 Port 1883

☐ Enable secure (SSL/TLS) connection

Client ID Leave blank for auto generated

Keep alive time (s) 60 ☒ Use clean session

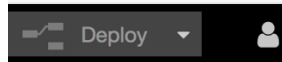
☐ Use legacy MQTT 3.1 support

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")`
Library 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:
      - "50002:1000"
    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: [{
```

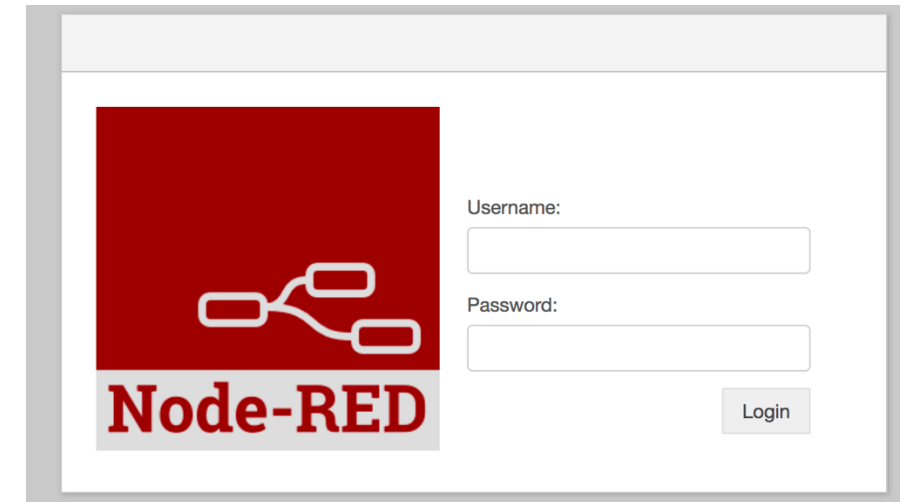
```
        username: my_function()['username'],
```

```
        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/>