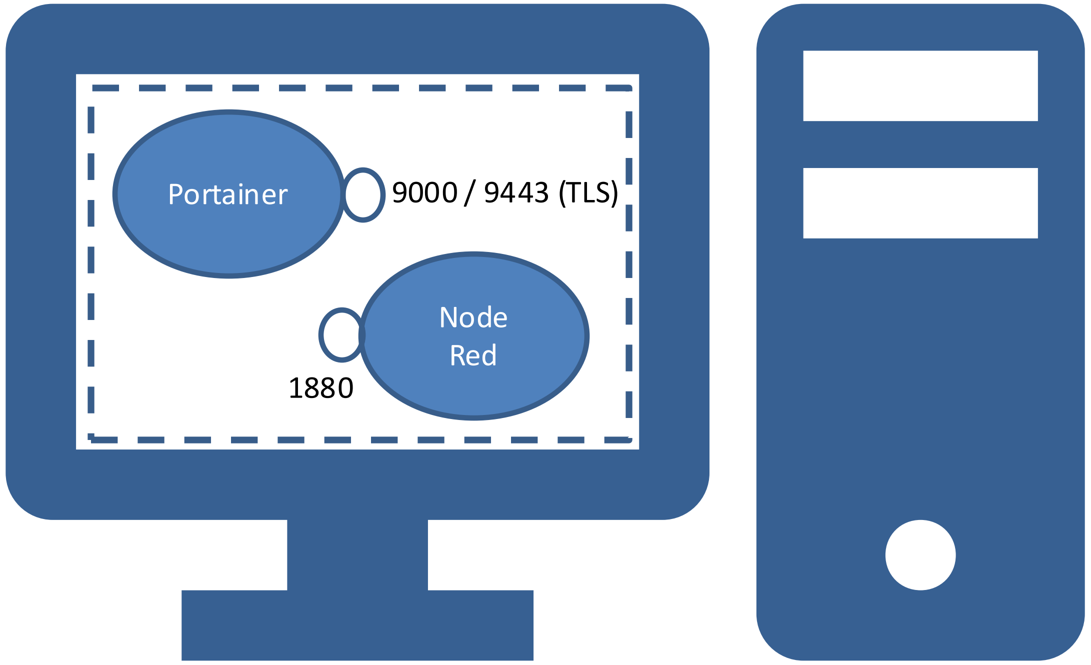
Docker Desktop (local)



IoT Lab 1:

Local Docker „Cloud“

Learning Objectives

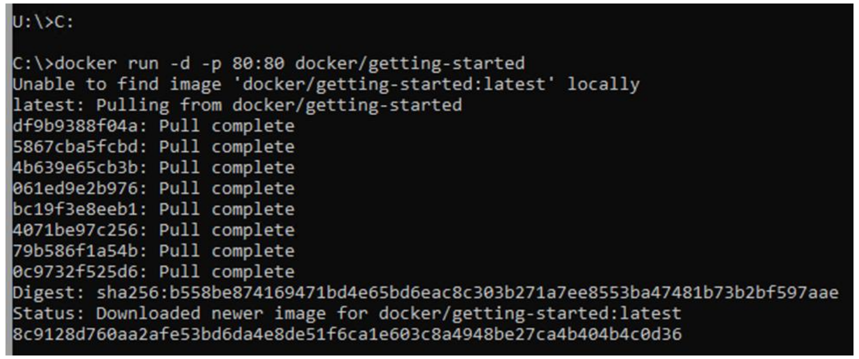
- deploy example containers in CLI
- learn basic commands / docker commands in CLI
- deploy Portainer CE and Node-Red as container in CLI
- learn basics in Portainer / networking / port-mapping
- create first flows in Node-Red (debug, read CPU load, connect to MQTT Broker (publish / subscribe), local dashboard)

- Some recommendations:
- Always have a look into the official documentation to get more detailed information and try out (especially at home) as much as possible. There are awesome videos on Youtube as an additional source for practicing and gathering knowledge

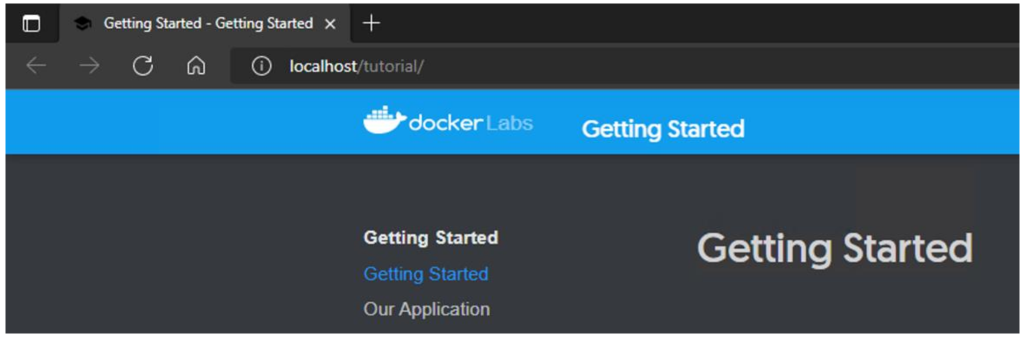
Docker Desktop - Getting started

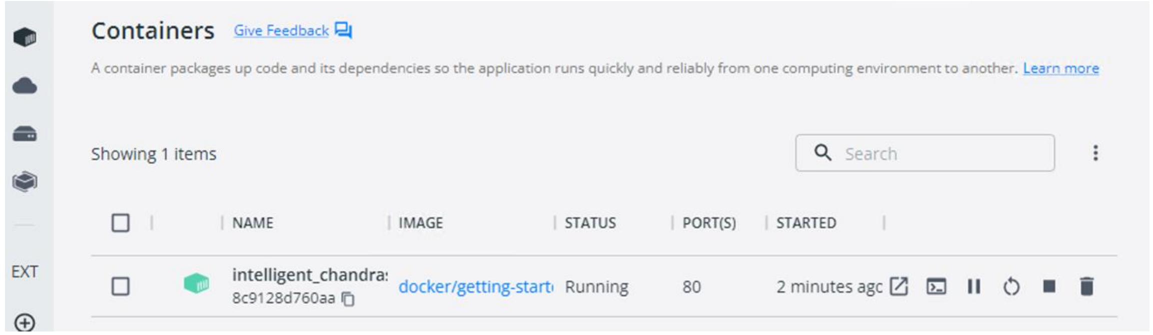
- Run once the following file: C:/zIT/zDocker before starting docker desktop, otherwise restart PC and run script first
- Start your local Docker Desktop and open cmd / terminal
  - Check with command *docker version* if docker is running correctly (see lab 0)
- If you like, start tutorial (@home)
- First we run a Sample Container as shown in (cmd)
  - *docker run -d -p 80:80 docker/getting-started*
- In Docker Desktop GUI you should see a running container now
- Open a browser with localhost or
  click on button "open with browser" at your container

HINT: this script deletes ALL your docker data, so do not run it during semester if not needed !!!



```
U:\>C:

C:\>docker run -d -p 80:80 docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
df9b9388f04a: Pull complete
5867cba5fcbd: Pull complete
4b639e65cb3b: Pull complete
061ed9e2b976: Pull complete
bc19f3e8eeb1: Pull complete
4071be97c256: Pull complete
79b586f1a54b: Pull complete
0c9732f525d6: Pull complete
Digest: sha256:b558be874169471bd4e65bd6eac8c303b271a7ee8553ba47481b73b2bf597aae
Status: Downloaded newer image for docker/getting-started:latest
8c9128d760aa2afe53bd6da4e8de51f6ca1e603c8a4948be27ca4b404b4c0d36
```





Containers Give Feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 1 items

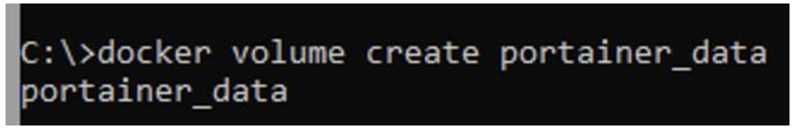
NAME	IMAGE	STATUS	PORT(S)	STARTED
intelligent_chandra: 8c9128d760aa	docker/getting-start	Running	80	2 minutes agc

- Congratulations: your first container is working ☺

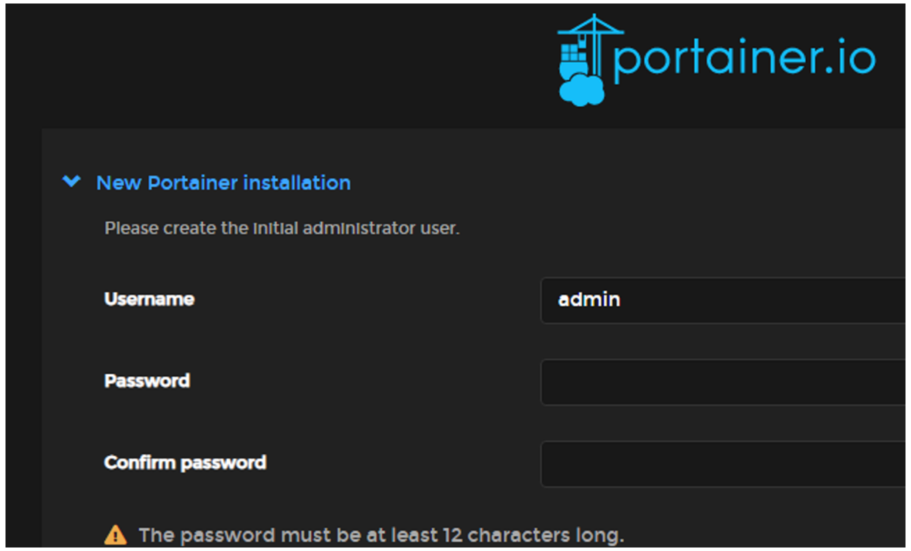
Docker Desktop - deploy Portainer (CLI)

- You can stop/delete the container of the last part if you like
- Get used to some commands: *docker run, ps, stop, rm and exec*
- Now we are going to deploy Portainer as Container Management Tool (this is later used in EI Cloud for deployments)
  - https://docs.portainer.io/start/install/server/docker/wsl
  - First we create a volume to store portainers database
    - *docker volume create portainer_data*
  - Then download and deploy portainer server (community edition)
    - *docker run -d -p 8000:8000 -p 9443:9443 -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest*
  - open portainer in browser with port 9000 (http) and set up user / password or 9443 with https

Port 8000: edge agent functionality
Port 9443: WebUI TLS Port (own certificate will be created while deployment)
Port 9000: WebUI non TLS Port



```
C:\>docker volume create portainer_data
portainer_data
```



New Portainer installation

Please create the initial administrator user.

Username: admin

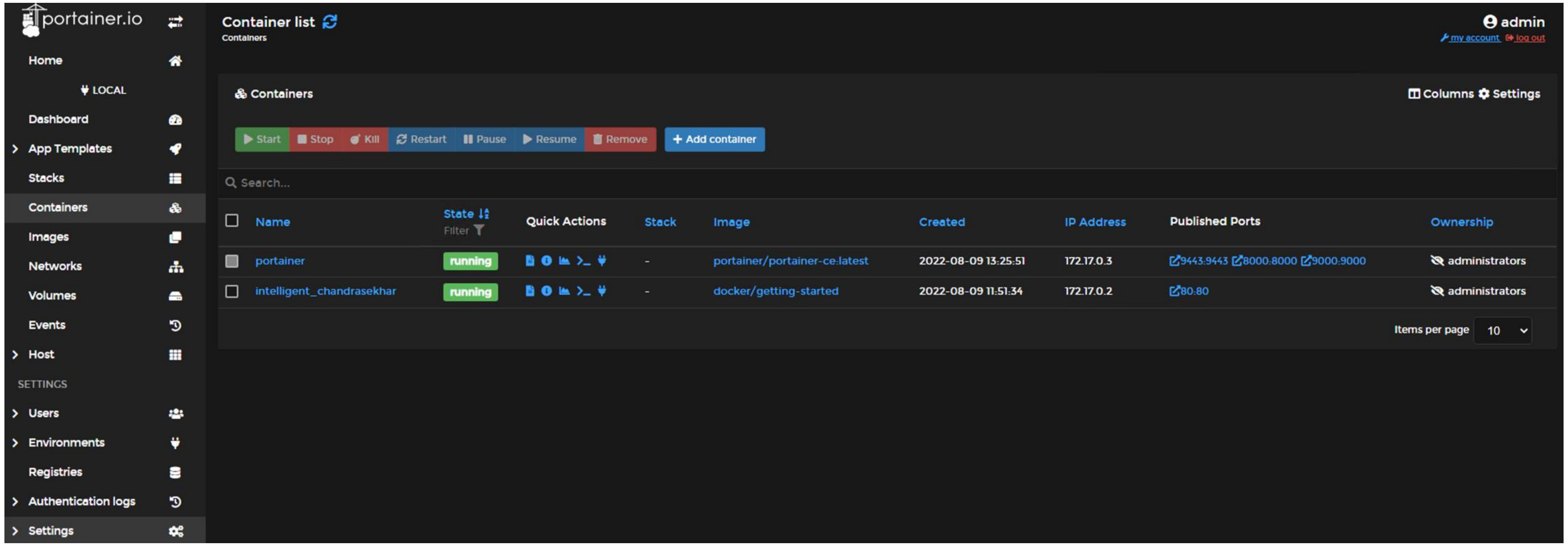
Password:

Confirm password:

The password must be at least 12 characters long.

Docker Desktop - deploy Portainer (CLI)

- After login into portainer you can find your deployed containers in Home → local (environment) → containers



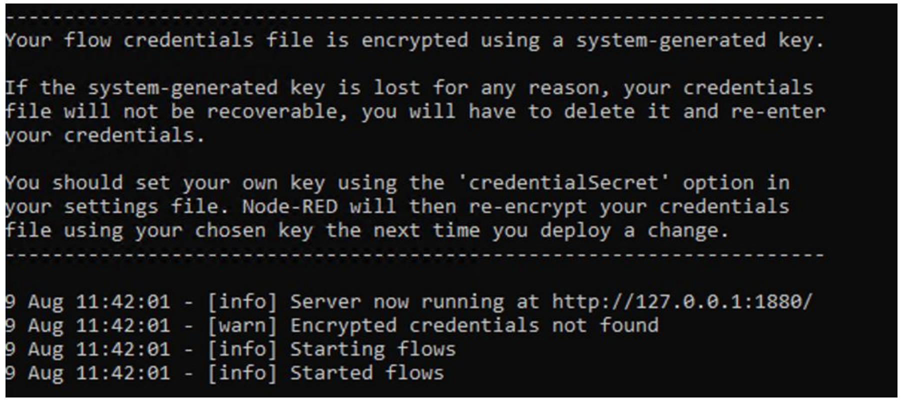
Name	State Filter	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
portainer	running		-	portainer/portainer-ce:latest	2022-08-09 13:25:51	172.17.0.3	9443:9443 8000:8000 9000:9000	administrators
intelligent_chandrasekhar	running		-	docker/getting-started	2022-08-09 11:51:34	172.17.0.2	80:80	administrators

Items per page 10

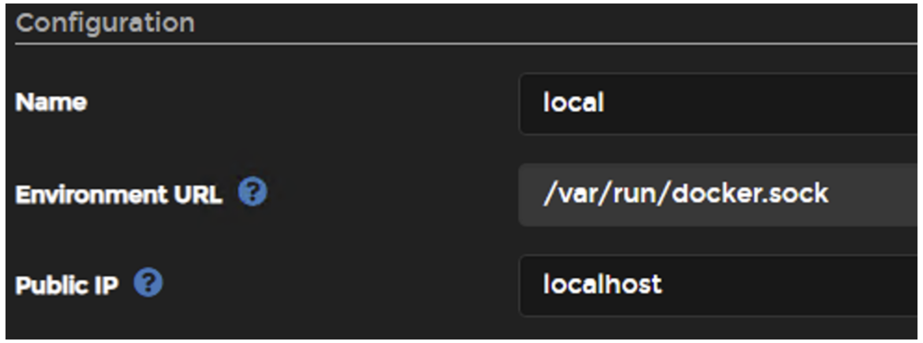
- Make yourself familiar with portainer environment, have a look at containers, networks, volumes, published ports, internal IPs etc...
- We will dive deeper into portainer and deployments in the next labs
- For now it is only a means to the end

Docker Desktop - deploy Node-Red (CLI)

- We are going to deploy Node-Red as webbased Application in CMD
  - *docker run -it -p 1880:1880 -v node_red_data:/data --name mynodered nodered/node-red*
- It is going to be started directly in your cmd window which causes a problem if we close that window
- So we are going to start the container in portainer
  - First terminate node-red in cmd with ctrl+c
  - Navigate in portainer to containers to find node-red (maybe you need to refreh the browser window)
  - "mynodered" should be shown as stopped
  - Select container and start it → starting → refresh → running
  - In Portainer GUI click on port 1880 in published port list
    - Browser opens with 0.0.0.0:1880 and an error message
    - To correct that, navigate to environments → local and put localhost in field Public IP → update environment
    - Open via port again → should be localhost:1880 now

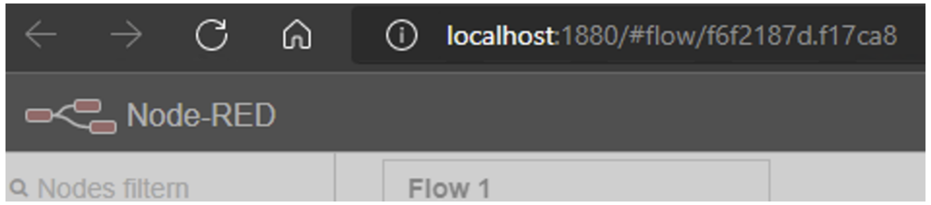


```
---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

9 Aug 11:42:01 - [info] Server now running at http://127.0.0.1:1880/
9 Aug 11:42:01 - [warn] Encrypted credentials not found
9 Aug 11:42:01 - [info] Starting flows
9 Aug 11:42:01 - [info] Started flows
```



Configuration

Name	local
Environment URL	/var/run/docker.sock
Public IP	localhost



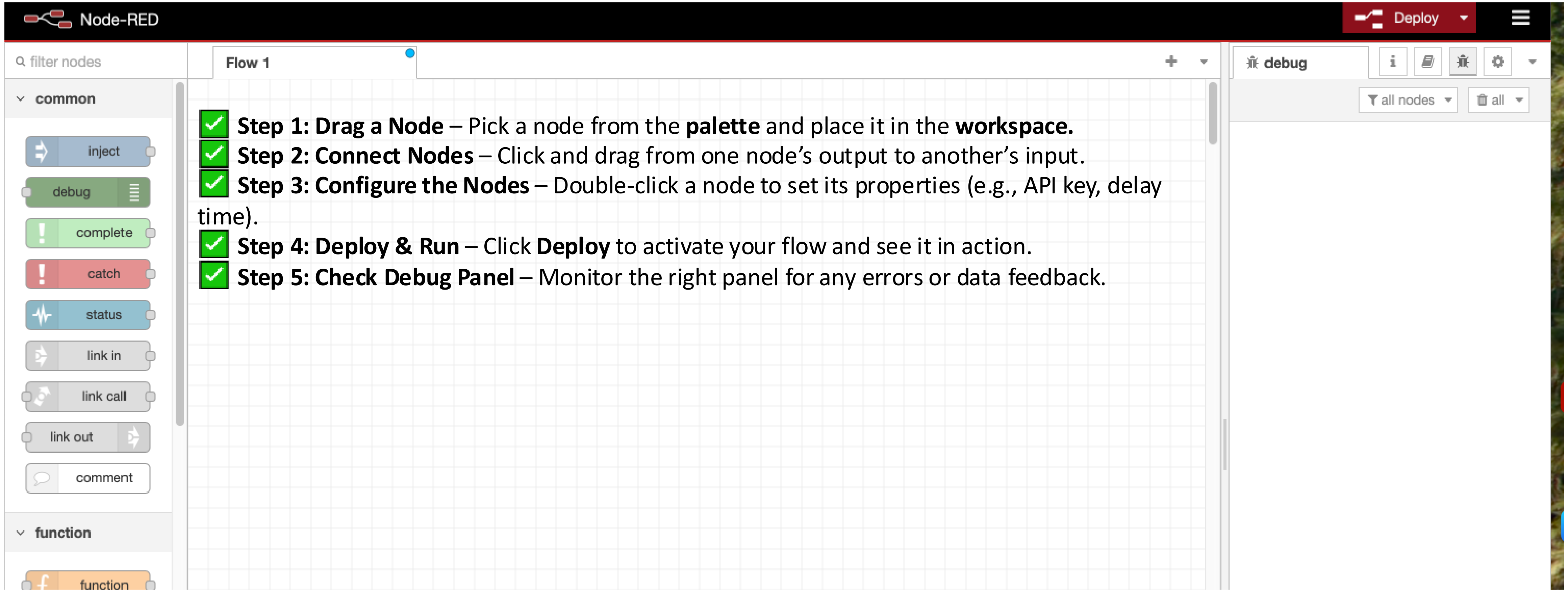
localhost:1880/#flow/f6f2187d.f17ca8

Node-Red – some details

- Node-RED is a **visual programming tool** that allows you to **connect devices, APIs, and online services** by simply **dragging and dropping blocks (nodes)**. It's widely used in IoT (Internet of Things), automation, and data processing.
- Think of Node-RED as a **big whiteboard where you can connect different building blocks (nodes) to create workflows (flows).** It has three main parts:
  - **Palette (Left Side)**
    - This is your **toolbox** with different **nodes** (blocks).
    - Each node performs a specific function, like getting data, processing it, or sending it somewhere.
  - **Flow Workspace (Middle Section)**
    - This is where you **drag and drop** nodes and connect them using **wires (connections).**
    - Each **flow** (sequence of nodes) represents a process
  - **Debug Panel (Right Side)**
    - shows messages, errors, and results of your flows.
    - Helps you **troubleshoot and test** what's happening.

✔ **No Coding Needed** – Drag & drop instead of writing long code.
✔ **Fast Development** – Build workflows quickly with ready-made nodes.
✔ **IoT & Automation Friendly** – Works with **Raspberry Pi, Arduino, smart devices, APIs, databases, and cloud services**.
✔ **Easy Debugging** – Built-in tools for testing and fixing errors.

Node-Red – some details



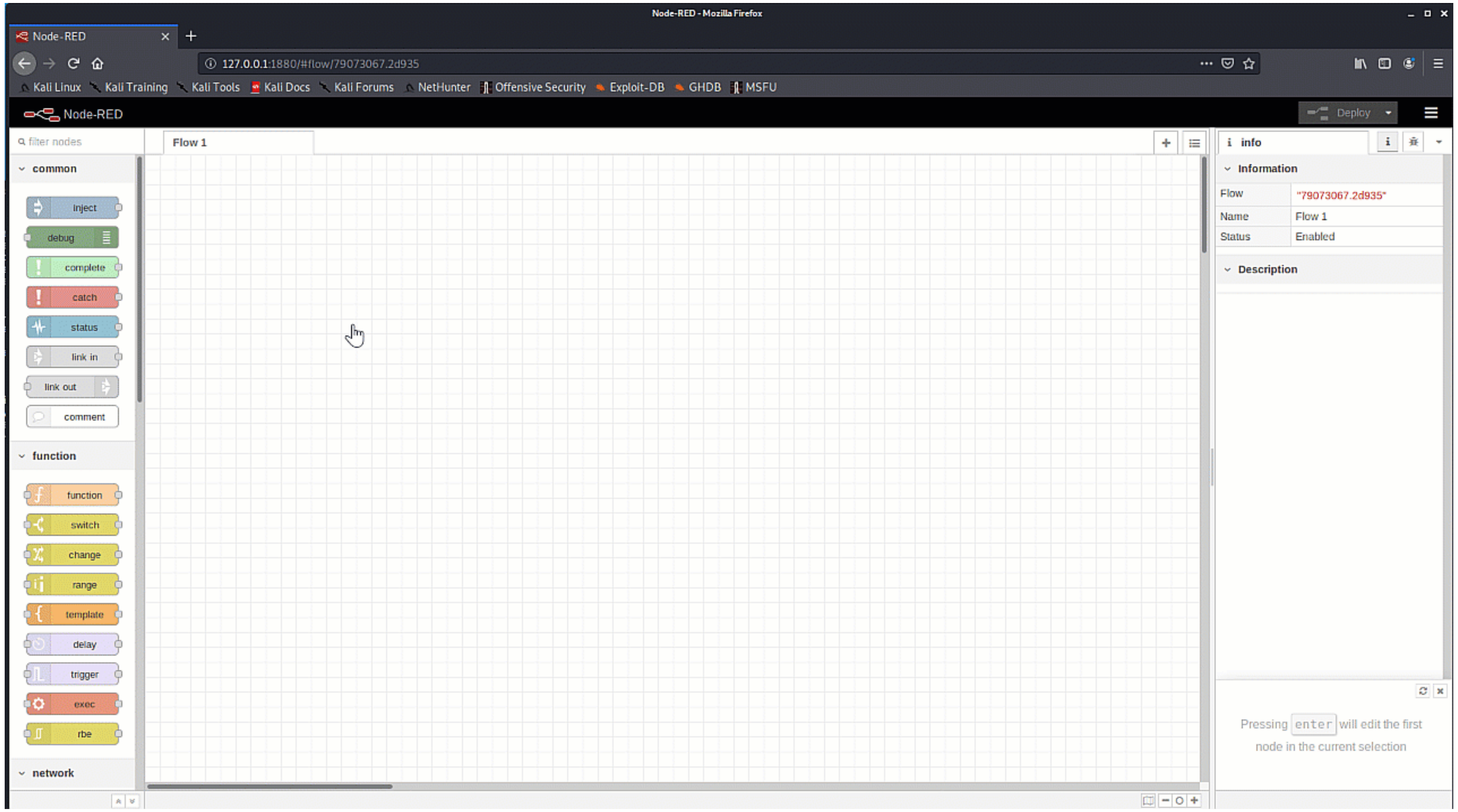
✓ **Step 1: Drag a Node** – Pick a node from the **palette** and place it in the **workspace.**
✓ **Step 2: Connect Nodes** – Click and drag from one node's output to another's input.
✓ **Step 3: Configure the Nodes** – Double-click a node to set its properties (e.g., API key, delay time).
✓ **Step 4: Deploy & Run** – Click **Deploy** to activate your flow and see it in action.
✓ **Step 5: Check Debug Panel** – Monitor the right panel for any errors or data feedback.

Palette **Flow Workspace** **Debug Panel**

Node-Red - Exercise 1

Node-Red introduction as simple flow

- rename your flow
- drag and drop 2 nodes:
  - **inject**
  - **debug**
- and **connect** them
- change inject node to **„string“ and put in text in payload field**
- switch window to debug
  - → **deploy!**
  - click inject **button**
- text should be in right debug window

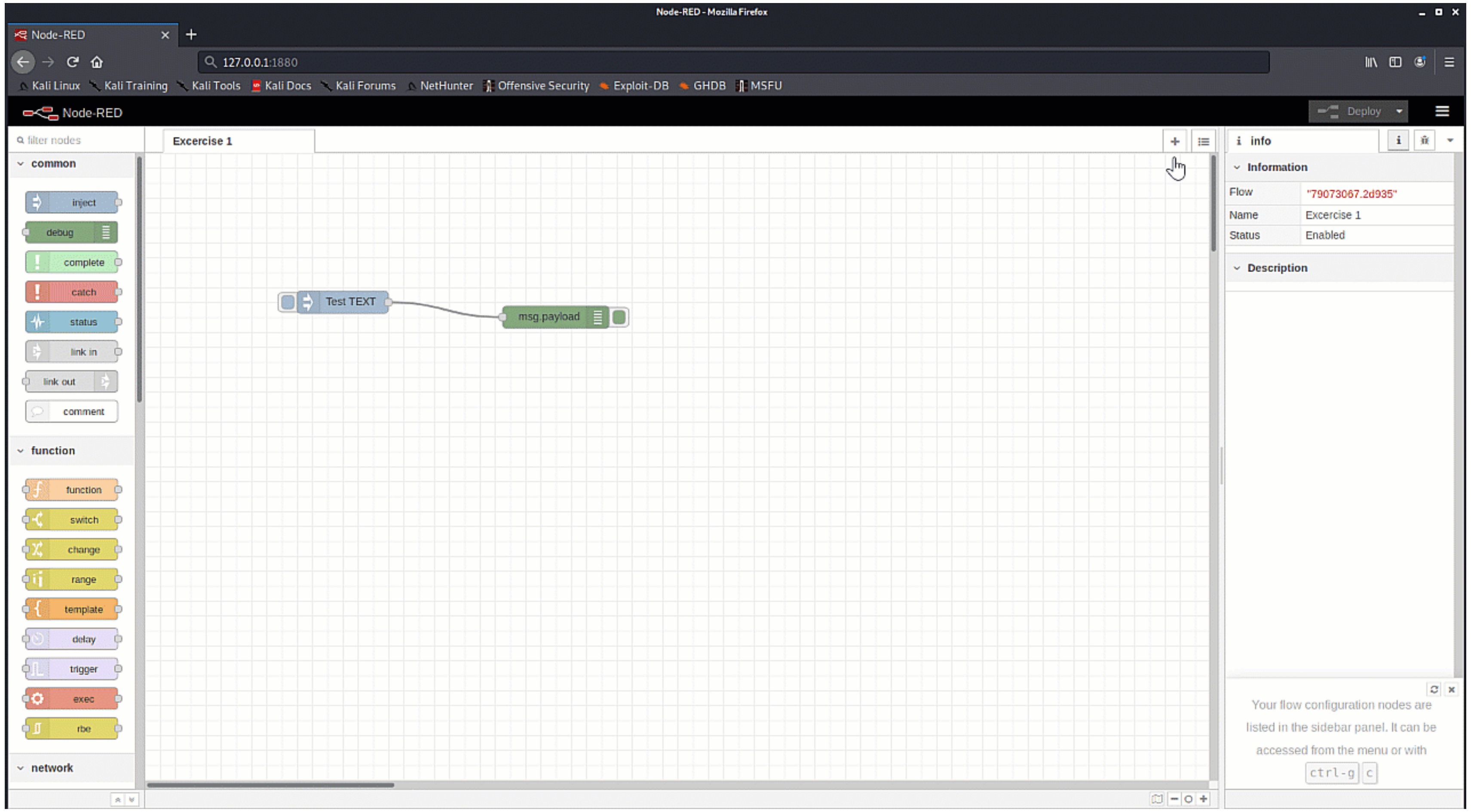


Flow	"79073067.2d935"
Name	Flow 1
Status	Enabled

Node-Red - Exercise 2

Simple chat with MQTT Broker

- **Create 2nd flow** and **rename** it
- send a string (inject node) to a public **MQTT** Broker (**publish**) with an unique topic: ***THM/IoTLab/yourname***
- read it (**subscribe**) and show it as text in **debug** window
- example MQTT Broker:
  - ***test.mosquitto.org***
  - → simple chat through MQTT Broker
- Check MQTT Messages in MQTT Explorer



Flow	"79073067.2d935"
Name	Excercise 1
Status	Enabled

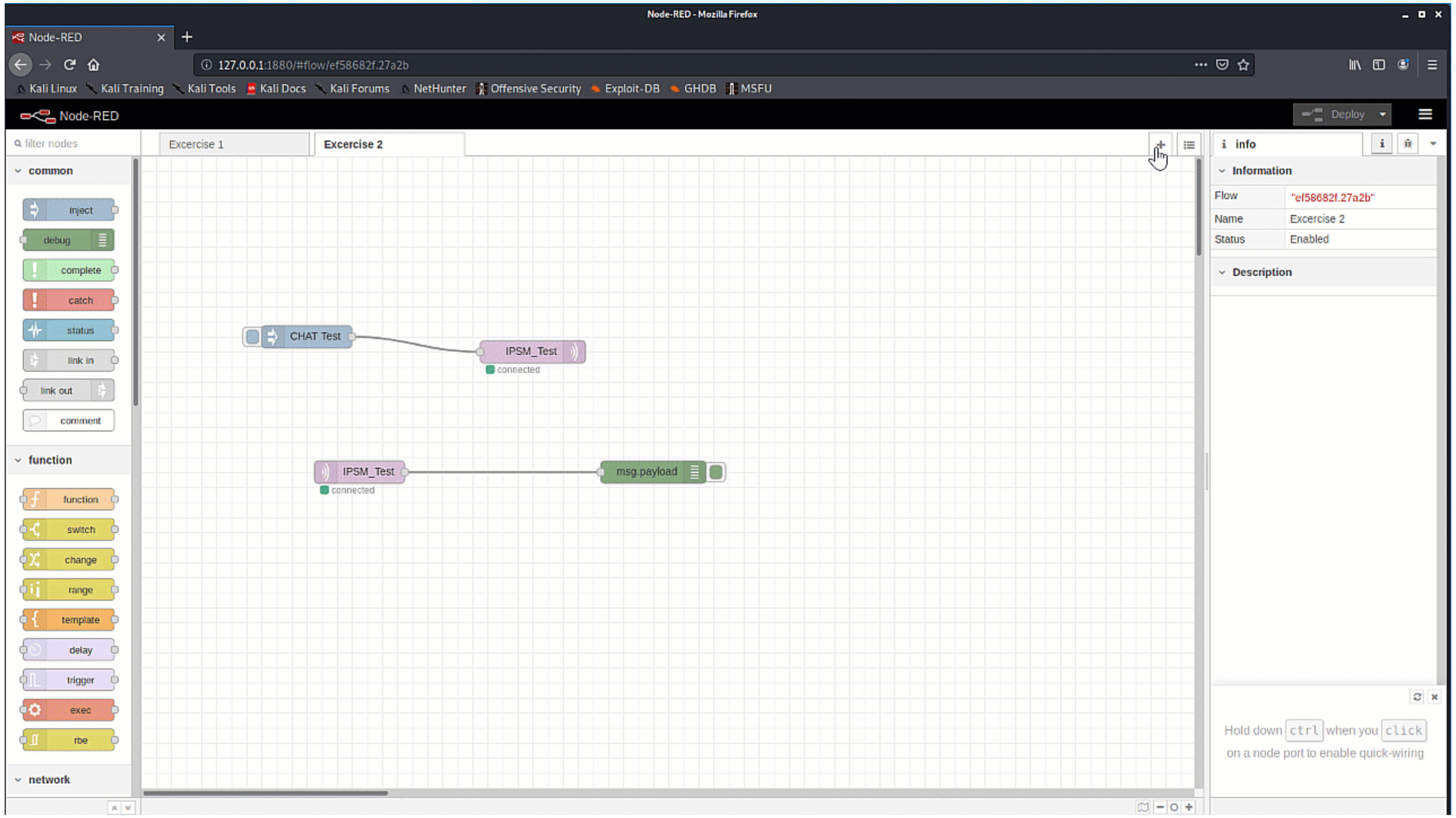
MQTT – some details

- **MQTT = “WhatsApp for Machines”** 📲🤖
  ✅ **M** – *Message*, ✅ **Q** – *Queuing*, ✅ **T** – *Telemetry*, ✅ **T** – *Transport*
- 👉 In simple terms, **MQTT is a lightweight protocol used to send and receive messages between devices**. It is super useful for **IoT (Internet of Things)**, where many small devices need to talk to each other using minimal power and internet data.
- MQTT follows a **Publisher-Subscriber Model** (like YouTube! 🎥).
  - 📡 **Step 1: A device (publisher) sends a message**
  - 📢 **Step 2: A central MQTT broker (server) receives it**
  - 👂 **Step 3: Devices (subscribers) get the message if they are subscribed to that topic**
- **Real-Life Example: our Lab today**
  - we have a **environment sensor (BME680 in Germany)** (**publisher**) that measures room temperature and other data and sends data every second.
  - Our NodeRed in Spain (**subscriber**) listens for all data updates and shows them in Debug windows and later: stores it in database and visualizes in Grafana
  - The **MQTT broker** (like a post office) ensures that messages go from the sensor to the NodeRed **without interruptions**.

Node-Red - Exercise 3

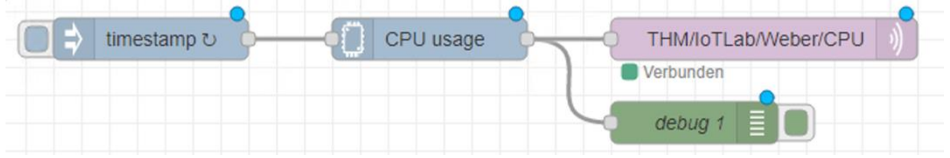
Read CPU overall load

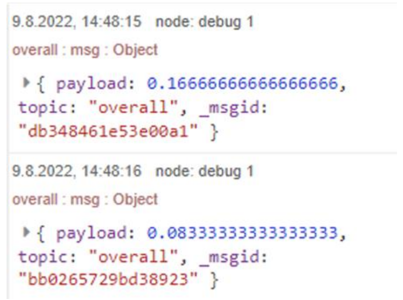
- **Create 3rd flow** and **rename** it
- Install a new node:
  - **node-red-contrib-cpu**
- use new node and display CPU load in debug window with following **settings**:
  - **inject-node with interval of 1 second**
  - **CPU Node: overall usage**
  - **debug node: complete msg object**



Node-Red - Exercise 4

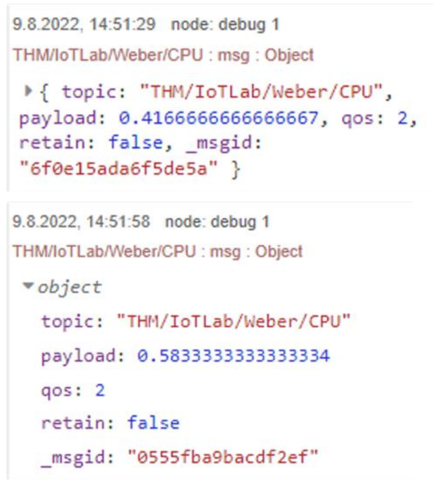
- We are going to send the CPU Data through MQTT back to our flow
  - Use the following MQTT topic: THM/IoTLab/<yourname>/CPU



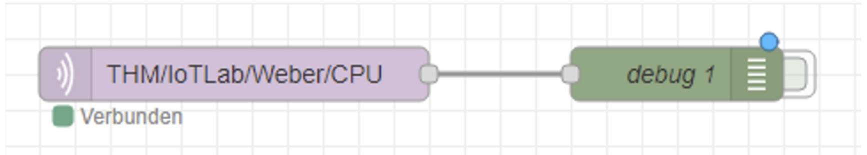


```
9.8.2022, 14:48:15   node: debug 1
overall : msg : Object
▸{ payload: 0.16666666666666666,
topic: "overall", _msgid:
"db348461e53e00a1" }

9.8.2022, 14:48:16   node: debug 1
overall : msg : Object
▸{ payload: 0.08333333333333333,
topic: "overall", _msgid:
"bb0265729bd38923" }
```

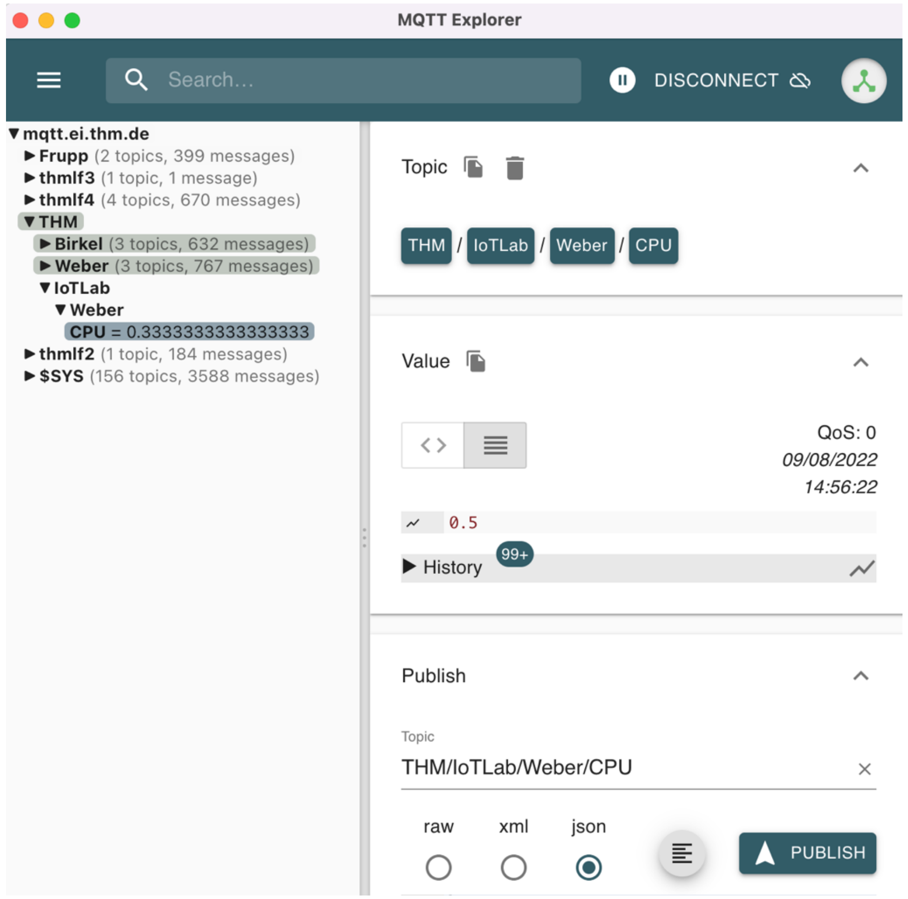
  - Subscribe again and show values in debug
  - Have a look at the complete msg object and compare to the object before sending with MQTT → can you find a difference?



```
9.8.2022, 14:51:29   node: debug 1
THM/IoTLab/Weber/CPU : msg : Object
▸{ topic: "THM/IoTLab/Weber/CPU",
payload: 0.41666666666666667, qos: 2,
retain: false, _msgid:
"6f0e15ada6f5de5a" }

9.8.2022, 14:51:58   node: debug 1
THM/IoTLab/Weber/CPU : msg : Object
▾object
  topic: "THM/IoTLab/Weber/CPU"
  payload: 0.5833333333333334
  qos: 2
  retain: false
  _msgid: "0555fba9bacdf2ef"
```



  - Analyse the MQTT Traffic as well with the MQTT Explorer



▼mqtt.ei.thm.de
▶Frupp (2 topics, 399 messages)
▶thmIf3 (1 topic, 1 message)
▶thmIf4 (4 topics, 670 messages)
▼THM
▶Birkel (3 topics, 632 messages)
▶Weber (3 topics, 767 messages)
▼IoTLab
▼Weber
CPU = 0.3333333333333333
▶thmIf2 (1 topic, 184 messages)
▶$SYS (156 topics, 3588 messages)

Topic
THM / IoTLab / Weber / CPU

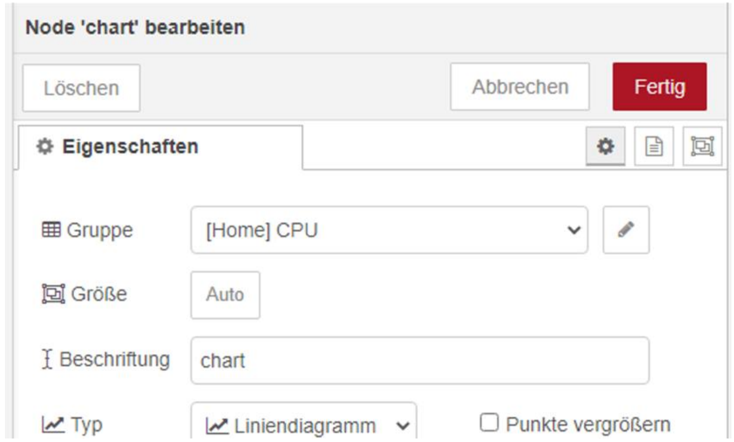
Value
QoS: 0
09/08/2022
14:56:22
0.5
History 99+

Publish
Topic
THM/IoTLab/Weber/CPU
raw xml [✓ json]
PUBLISH

Node-Red - Exercise 5

Create Dashboard and show values

- **Install** dashboard nodes:
  - **node-red-dashboard**
- Use chart node
- Create **new dashboard group (name it CPU) and new ui tab (can be left Home)**
- Connect the subscribing MQTT Node to your chart node



Node 'chart' bearbeiten

Löschen | Abbrechen | Fertig

Eigenschaften

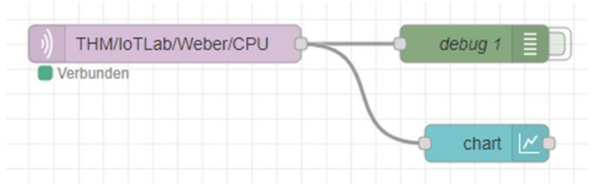
Gruppe: [Home] CPU

Größe: Auto

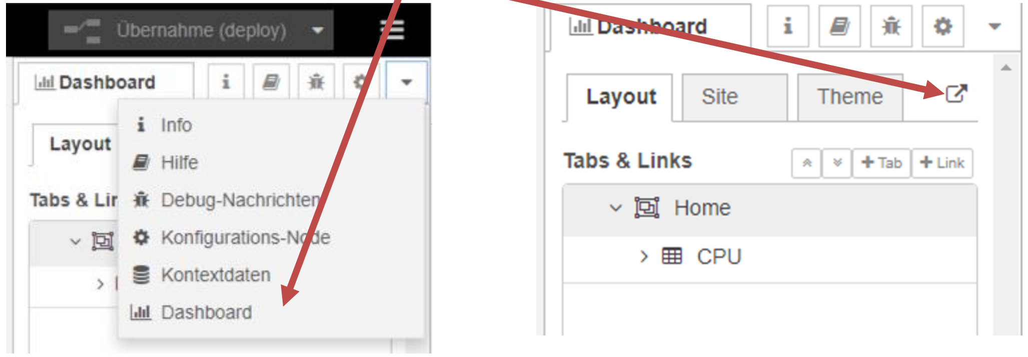
Beschriftung: chart

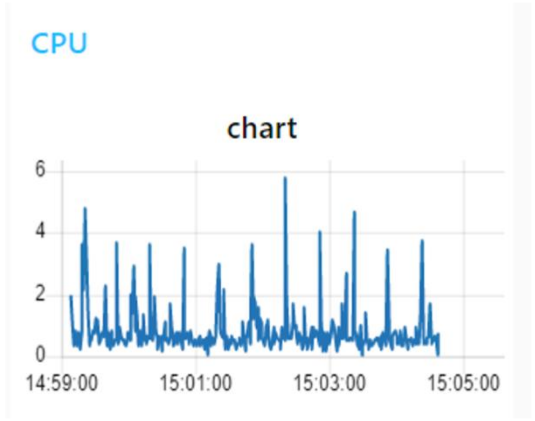
Typ: Liniendiagramm

☐ Punkte vergrößern



- Open **dashboard** → CPU values should be shown as simple chart





Lab 1 Summary

Docker CLI, Portainer & Node-RED

In this lab, you learned to:

- **Docker CLI Basics** 🐳
  - Use commands like docker run, ps, stop, rm and exec
  - Deploy and manage containers from the terminal
- **Deploy Core Tools** 📦
  - Launch **Portainer CE** and **Node-RED** via CLI
  - Understand their roles in container and flow management
- **Work with Portainer** 🖥
  - Navigate the UI to manage Docker containers
  - Inspect logs, start/stop containers, and deploy images
- **Create Flows in Node-RED** 🔄
  - Build your first flows using inject, debug, and function nodes
  - Monitor system performance (e.g., CPU load)
  - Connect to an MQTT broker (publish/subscribe)
  - Design a simple dashboard to display live data

- **You now have the foundational skills to** 🎯
  - Automate data flows
  - Manage containerized applications
  - Visualize sensor or system data in Node-Red

Lab 1 Review & Reflection

- What is the purpose of containers, and how do they differ from virtual machines?
- How do you start and stop a Docker container using the CLI?
- Which command shows all running containers?
- What is Portainer CE and why is it useful?
- What are the basic components of a Node-RED flow?

Lab 1 Review & Reflection

- What is the difference between **publish** and **subscribe** in MQTT?
- How do you deploy a container via CLI?
- Create a topology for this lab (template is provided)