sysWORXX IO Docker Container

Application Note

Edition May 2025

Document No.: L-2949e-02

SYS TEC electronic AG   Am Windrad 2   D-08468 Heinsdorfergrund
Phone: +49 (3765) 38600-0   Fax: +49 (3765) 38600-4100
Web: https://www.systec-electronic.com   Mail: info@systec-electronic.com

Changes

Version	Date	Section	Changes
01	03.2025	All	Initial document
02	05.2025	all	Figures added

<table>
<tr><th>Contact</th><th>Direct</th><th>Your Local Distributor</th></tr>
<tr><td>Address:</td><td>SYS TEC electronic AG
Am Windrad 2
D-08468 Heinsdorfergrund
GERMANY</td><td rowspan="5">Please contact us for information about your local distributor.

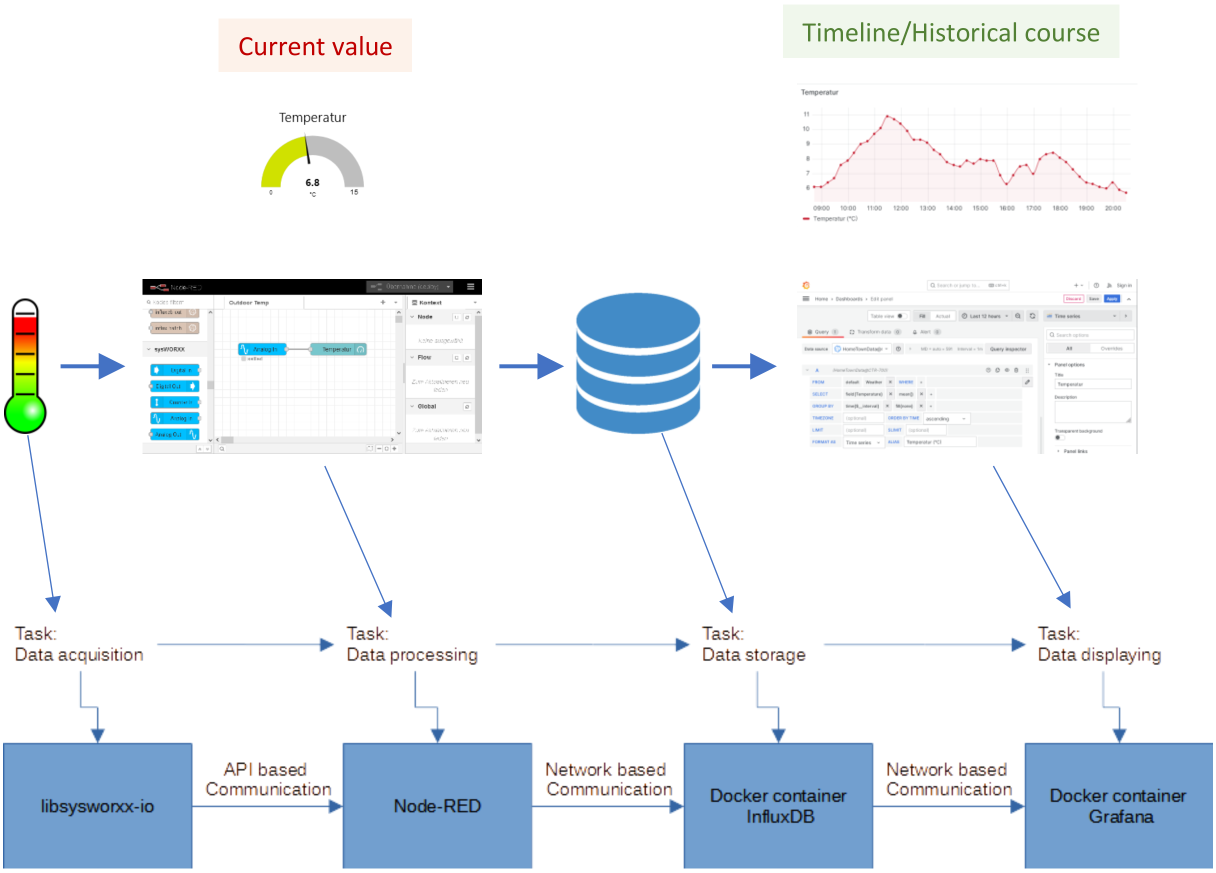
info@systec-electronic.com</td></tr>
<tr><td>Ordering Information:</td><td>+49 (0) 37 65 / 38 600-0
info@systec-electronic.com</td></tr>
<tr><td>Technical Support:</td><td>+49 (0) 37 65 / 38 600-0
support@systec-electronic.com</td></tr>
<tr><td>Fax:</td><td>+49 (0) 37 65 / 38 600 4100</td></tr>
<tr><td>Web Site:</td><td>https://www.systec-electronic.com</td></tr>
</table>

Table of Contents

List of Figures

1 Introduction

The main goal of this demo is to demonstrate the integration of various tools such as Node-RED, InfluxDB, Grafana, and Docker for visualizing and storing sensor data. The following aspects will be demonstrated:
Yocto Linux provides the base operating system. If additional applications are required that are not included in the base Yocto image, they can be added using Docker containers.
Grafana is an open-source software application primarily used for data visualization and analysis. It allows data from various sources to be presented in clear dashboards. These dashboards consist of various visualization elements such as charts, graphs, gauges, and tables that help quickly identify patterns, trends, and anomalies.
Grafana is particularly popular for monitoring systems, networks, IoT devices, and other measurement data because of its flexibility and support for a wide variety of data sources, such as InfluxDB, Prometheus, MySQL, and many more. It's ideal for presenting complex data in a clear and interactive way, allowing you to grasp important information at a glance.
The demo will show how to use Grafana in kiosk mode to present a clear and appealing view of the data.
InfluxDB is a special database that specializes in storing and analysing time series data. This means it's particularly good at managing data that is recorded at regular intervals, such as temperature values, CPU usage, weather data, or other measurements collected over time. What's special about InfluxDB is that it can process large amounts of time series data very quickly and efficiently.
InfluxDB offers functions for storing, querying, and analysing data, allowing you to identify trends or filter out specific events. It also supports data aggregation and filtering, which is very helpful when monitoring systems or analysing measurement data. The demo demonstrates how measured values are stored in an InfluxDB and displayed for visualization in a web browser as well as on a local display (kiosk mode).
The demo uses the CPU temperature as the specific data source. The libsysworxx-io library is used to cyclically read the data in the background. Node-RED is used to read this data from libsysworxx-io, format it, and write it to the InfluxDB.



2 Installation

The Yocto image on each sysWORXX device already includes tools such as Docker and Node-RED, as well as the libsysworxx-io library. In contrast, the required Docker containers for InfluxDB and Grafana are not included. For these, Docker images must be created and then run as a Docker container instance in the next step. Communication between the Docker containers takes place using Docker networks. To use Node-RED, the corresponding service must be started. The creation and configuration of the required tools is described in the following chapters.

2.1 Install Docker Container

Docker containers are lightweight, portable, and self-sufficient units that package an application and all its dependencies, allowing it to run consistently across different computing environments. They utilize the host system's kernel, making them more efficient than traditional virtual machines. Docker simplifies the deployment process, enabling developers to create, deploy, and manage applications easily.

An example of using Docker containers is setting up a monitoring stack with Grafana (https://hub.docker.com/r/grafana/grafana) and InfluxDB (https://hub.docker.com/_/influxdb). InfluxDB is a time-series database that stores metrics and events, while Grafana is a visualization tool that allows users to create dashboards and graphs from the data stored in InfluxDB. By running both applications in separate Docker containers, once can quickly deploy a robust monitoring solution. For instance, once can collect performance metrics from the applications, store them in InfluxDB, and visualize them in real-time using Grafana, all while ensuring that the setup is consistent and easily replicable across different environments.

2.1.1 Customize User Account

If the following examples are not run with the root user, the user who creates containers must be added to the Docker group as a new user.

Add the user to the docker group:

```
root@sysworxx:/# usermod -aG docker <user name>
```

Log out (command: logout) and log back in so that the group membership is re-evaluated. Or run the following command to activate the changes to groups:

newgrp docker

2.1.2 Start Docker-Daemon

To use Docker, the Docker daemon must be running in the background. The daemon is started by executing the first Docker command. However, to start the daemon automatically upon reboot, the service must be configured as follows:

```
$  systemctl enable docker
$  reboot
```

The command

```
$  systemctl status docker
```

can be used to determine whether the service is running.

2.1.3 Create a Docker Network

First, create a Docker network to allow the containers to communicate with each other.

Three different network types (network drivers) are created for a Docker container: Bridge, Host, and None. The Bridge network is the default mode and provides a virtual network for communication between containers. In a Bridge network, individual ports can be mapped to other ports on the host.

The host driver uses the host's network stack. This makes all of the host's ports and protocols available to the container. However, this interface is not isolated from the host in the container.

With the None network, the container does not contain a network stack.

For two containers to communicate, they must be on the same network. To do this, this network is first created, and then the containers are bound to this network upon startup.

```
$  docker network create monitoring
```

After executing this command, a network “monitoring” of type Bridge exists.

```
$  docker network ls
```

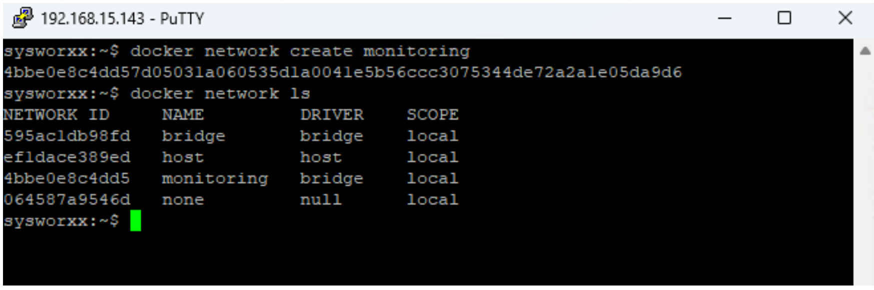


Figure 1: Docker network

2.1.4 Download, Build and Run the InfluxDB Container

Next, create and start an InfluxDB container. Docker Hub provides preconfigured Docker files for various applications, from which you can then build a Docker Image. The Docker file contains the necessary commands to load the sources from the required repositories. Therefore, the device must be connected to the internet.

```
$  docker run -d
    --restart=always
    --name=influxdb
    --network=monitoring
    -p 8086:8086
    -v "influxdb_storage:/var/lib/influxdb2"
    -v "influxdb_storage:/etc/influxdb2"
    -e INFLUXDB_DB=metrics
    -e INFLUXDB_ADMIN_USER=admin
    -e INFLUXDB_ADMIN_PASSWORD=sharedSecret
    influxdb:latest
```

The above command does the following things:

The Docker Image file for the most recently released InfluxDB Docker Image is downloaded from Docker Hub, then an image named influxdb is built. The image is then started for background execution (flag -d) and various environment variables are set in the container (flag -e). The option --restart=always causes the Docker Engine to restart the container after every reboot (e.g. after powering on the device).

The container is bound with the bridge network named “monitoring”. The port 8086 is mapped with the following flag:
-p 8086:8086 maps the port 8086 of the container to the port 8086 of the host. An IP frame with the host's IP address and port 8086 is then forwarded to the InfluxDB in the container.

A Docker Volume is created for storing data and config values outside the container.[1] The volume is attached to the specified paths within the container.
-v influxdb_storage:/var/lib/influxdb2 creates a volume called *influxdb_storage* which stores the InfluxDB data. The path */var/lib/influxdb2* is the default path where InfluxDB stores its data. The database data is stored in the volume. The other mount path is used the save configuration data.

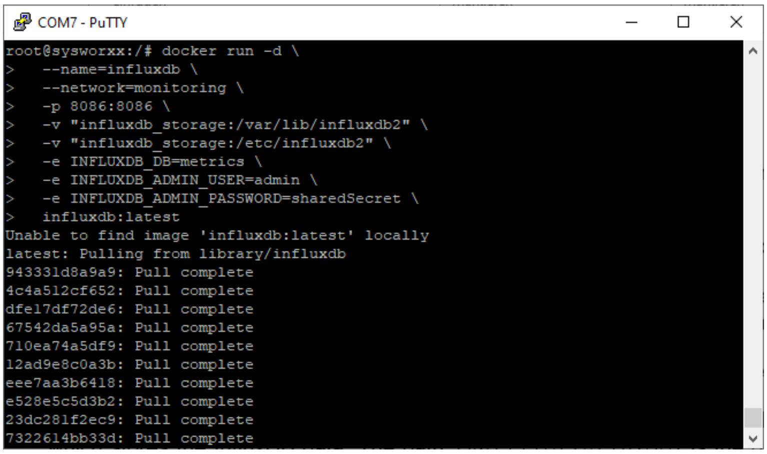


Figure 2: Docker pulling and build InfluxDB image

[1] Creating a Docker Volume has the advantage that the database data is preserved when Docker is terminated or deleted. This makes it easier to move this data to another location. Note: The “2” at the end of the path is important because the Docker Image uses influxDB version 2.xx.

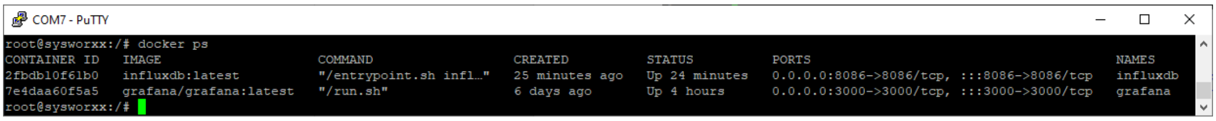


Figure 3: Docker container InfluxDB

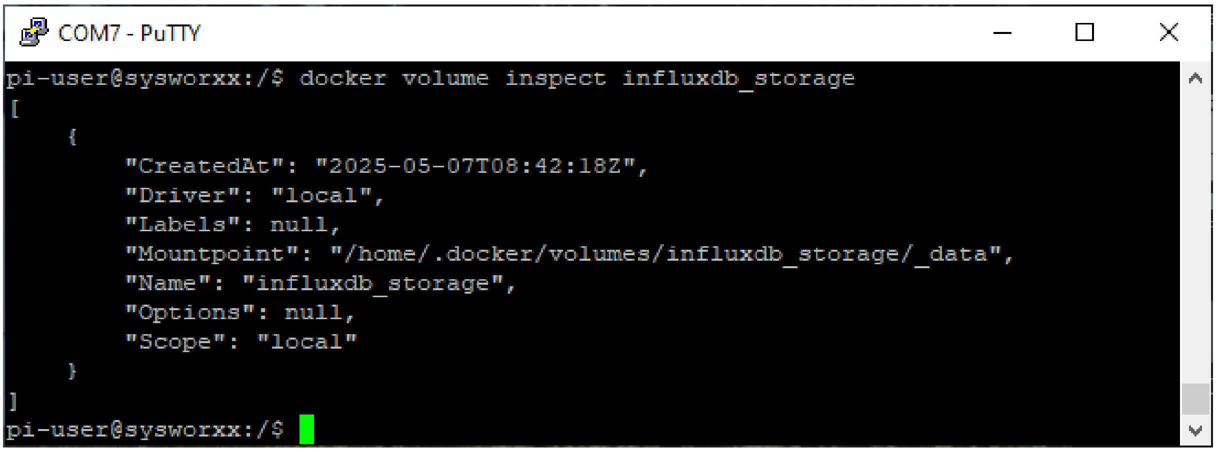
Docker Volumes are a way to persist data generated by and used by Docker containers. They are stored outside of the container's filesystem, which means that the data can be retained even if the container is removed.

Here are some key points about Docker Volumes:

1. Persistence: Volumes allow to keep data even when containers are stopped or deleted.
2. Sharing: Multiple containers can share the same volume, making it easy to share data between them.
3. Performance: Volumes are optimized for performance and can be managed more easily than bind mounts.
4. Backup and Restore: It can easily backup and restore volumes, which is essential for data management.
5. Isolation: Volumes provide a level of isolation from the host filesystem, which can enhance security.

To get the mount point of the Docker Volume use command:

```
$  docker volume inspect influxdb_storage
```



Setup influx from host or network see: https://docs.influxdata.com/influxdb/v2/get-started/setup/

2.1.5 Download, Build and Run the Grafana Container

Next, build and run an Grafana container. Grafana container will be connected to the InfluxDB container.

```
$  docker run -d
    --restart=always
    --name=grafana
    --network=monitoring
    -p 3000:3000
    -v "grafana_storage:/var/lib/grafana"
    -v "grafana_storage:/etc/grafana/provisioning"
    -e GF_SECURITY_ADMIN_PASSWORD=sharedSecret
    grafana/grafana:latest
```

`-v grafana-storage:/var/lib/Grafana` creates a volume called `grafana-storage` which is associated with the directory `/var/lib/Grafana` directory in the container. Grafana stores its files here. The configuration and dashboards are saved in the volume.

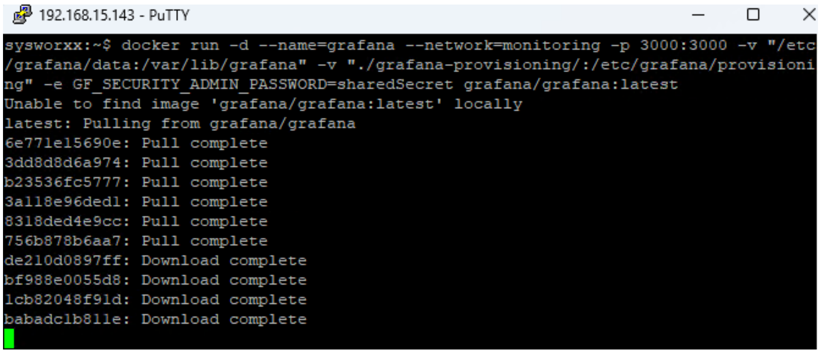


Figure 4: Docker pulling Grafana image

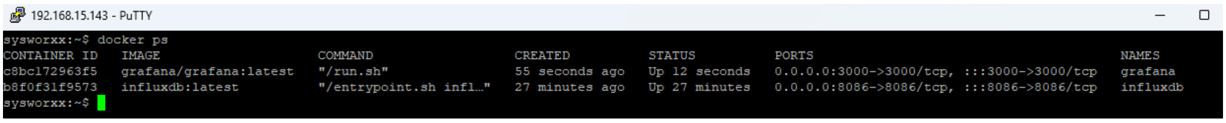


Figure 5: Docker container InfluxDB and Grafana

Docker Volumes are a way to persist data generated by and used by Docker containers. They are stored outside of the container's filesystem, which means that the data can be retained even if the container is removed.

Here are some key points about Docker Volumes:

1. Persistence: Volumes allow to keep data even when containers are stopped or deleted.
2. Sharing: Multiple containers can share the same volume, making it easy to share data between them.
3. Performance: Volumes are optimized for performance and can be managed more easily than bind mounts.
4. Backup and Restore: It can easily backup and restore volumes, which is essential for data management.
5. Isolation: Volumes provide a level of isolation from the host filesystem, which can enhance security.

2.1.6 Useful Docker Commands

Here are some useful Docker commands that can help manage the container applications:

1. The following command starts a defined Docker.
   docker run IMAGE
   Add the *-d* flag to run them in detached mode (in the background):
   docker run -d IMAGE
2. The following command starts a specific container.
   docker start CONTAINER_ID
3. The following command stops a specific container.
   docker stop CONTAINER_ID
4. The following command restarts a specific container.
   docker restart CONTAINER_ID
5. The following command removes a specific container.
   docker rm CONTAINER_ID
6. The following command displays the logs for all services. It can specify a service name to view logs for a specific service:
   docker logs CONTAINER_ID
7. The following command lists the containers that are currently running.
   docker ps
   Add the *-a* flag show all stopped container.
   docker ps -a

2.1.7 Docker Compose

This chapter describes the alternative approach for installing Docker Containers using Docker Compose.

Docker Compose is a tool that simplifies the management of multi-container Docker applications. It allows developers to define and run applications using a single YAML file, called *docker-compose.yml*, where they can specify the services, networks, and volumes required for their application. With Docker Compose, once can easily configure the settings for each container, such as environment variables, ports, and dependencies.

Using Docker Compose, to start all the services defined in the YAML file with a single command (`docker-compose up`), making it easier to manage complex applications that consist of multiple interconnected containers. This tool is particularly useful for development and testing environments, as it streamlines the process of setting up and tearing down applications. Overall, Docker Compose enhances productivity by providing a clear and efficient way to manage multi-container Docker applications.

2.1.7.1 Docker Compose Commands

Here are some useful Docker Compose commands that can help manage the multi-container applications:

1. The following command starts all the services defined in the *docker-compose.yml* file.
   `docker-compose up`
   Add the `-d` flag to run them in detached mode (in the background):
   `docker-compose up -d`
2. The following command stops and remove all the services defined in the *docker-compose.yml* file.
   `docker-compose down`
3. The following command displays the logs for all services. It can specify a service name to view logs for a specific service:
   `docker-compose logs`
4. The following command lists the containers that are currently running as part of the Docker Compose application.
   `docker-compose ps`
5. The following command validates and displays the configuration of the *docker-compose.yml* file.
   `docker-compose config`

2.1.7.2 Docker Compose Example

The first step is to connect to the device via SSH or Serial. The create a YAML file:

```
touch docker-compose.yml
```

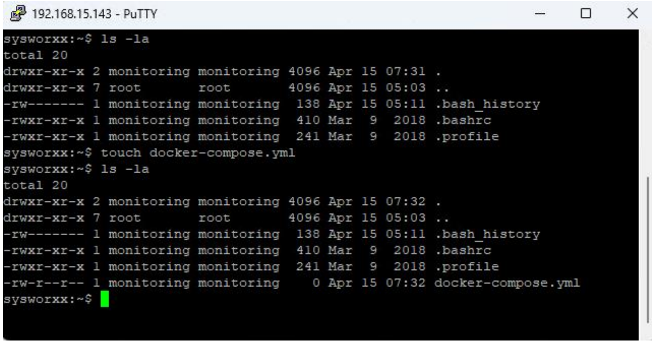


Figure 6: Add docker-compose file

Add the configuration for all services:

```
vim docker-compose.yml

services:
    influxdb:
        image: influxdb:latest
        container_name: influxdb
        restart: unless-stopped
        ports:
            - 8086:8086
        networks:
            - monitoring
        volumes:
            - influxdb_storage:/var/lib/influxdb2
        environment:
            - DOCKER_INFLUXDB_INIT_DB=metrics
            - DOCKER_INFLUXDB_INIT_MODE=setup
            - DOCKER_INFLUXDB_INIT_USERNAME=user
            - DOCKER_INFLUXDB_INIT_PASSWORD=sharedSecret
            - DOCKER_INFLUXDB_INIT_ORG=environment
            - DOCKER_INFLUXDB_INIT_BUCKET=energy
    grafana:
        image: grafana/grafana:latest
        container_name: grafana
        restart: unless-stopped
        ports:
            - 3000:3000
        networks:
            - monitoring
        volumes:
            - grafana_storage:/var/lib/grafana
            - ./grafana-provisioning/:/etc/grafana/provisioning
        depends_on:
            - influxdb
        environment:
            - GF_SECURITY_ADMIN_USER=admin
            - GF_SECURITY_ADMIN_PASSWORD=sharedSecret

networks:
    monitoring:
        driver: bridge

volumes:
    grafana_storage: {}
    influxdb_storage: {}
```

The example compose data above contains information about to deploy 2 containers at a time

- InfluxDB database to store data in time series manner
- Grafana graphical tool for visualizing InfluxDB

The first start of the container takes a few minutes, as all non-existent images have to be downloaded. See 2.1.7.1 for configuration Docker compose.

2.2 Install Node-RED Services

The device comes with a few “systemd” services such as *Node-RED*. There are two ways to run these services:

1. The following command is used to show the status of the *Node-RED* service:
   ```
   systemctl status node-red
   ```
2. The following commands are used to manually start or stop *Node-RED* services:
   ```
   systemctl start node-red
   systemctl stop node-red
   ```
3. There is also the possibility to start the *Node-RED* services automatically on power-on reset or reboot.
   ```
   systemctl enable node-red
   ```
4. To disable the automatic start, the following command is used:
   ```
   systemctl disable node-red
   ```

Execute the following commands to enable and start Node-RED editor.

```
systemctl enable node-red
reboot
```

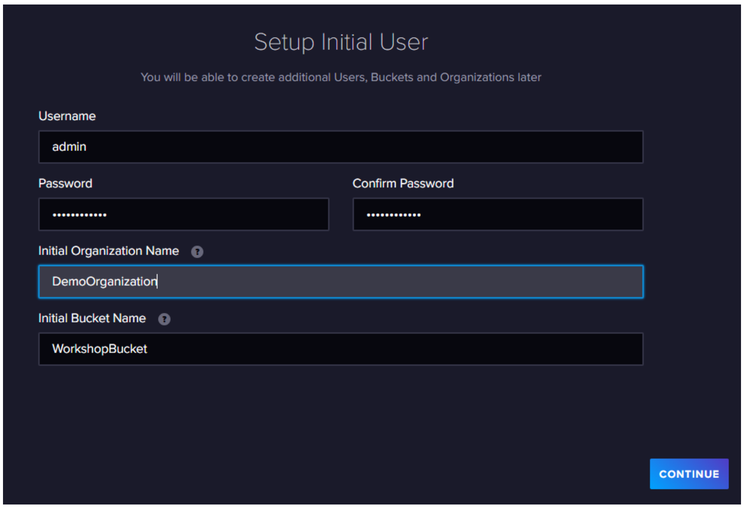
The Node-RED editor listens by default on port 1880. The Node-RED editor can then be opened in the web browser via the address http://<device-ip-address>:1880.

3 Connection between Node-RED, Grafana und InfluxDB

3.1 Configuration of InfluxDB

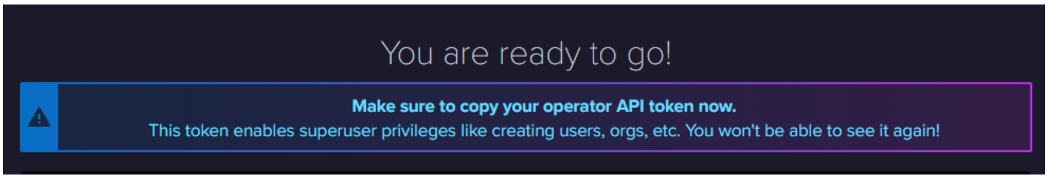
At first start the UI of InfluxDB. Access InfluxDB by navigating to http://<device ip address>:8086 in the web browser. The default login credentials are:

- Username: admin
- Password: sharedSecret

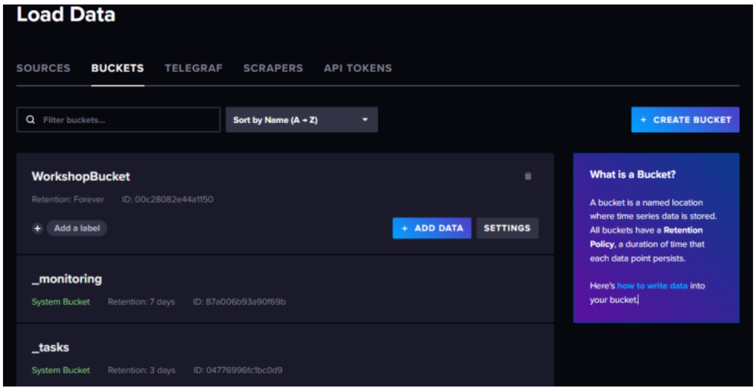


Give the organization and bucket[2] a name.

In the next step, copy your operator API token and save it in a safe place.



Now we are ready to collect data in the database. The default setting for retention policy is set to "forever". You can change it in the Bucket Settings dialog.



[2] A bucket is the central container where time series data is stored in InfluxDB, coupled with a defined retention policy that automatically deletes old data and serves as an organizational unit for managed access. A bucket is assigned to an organization.

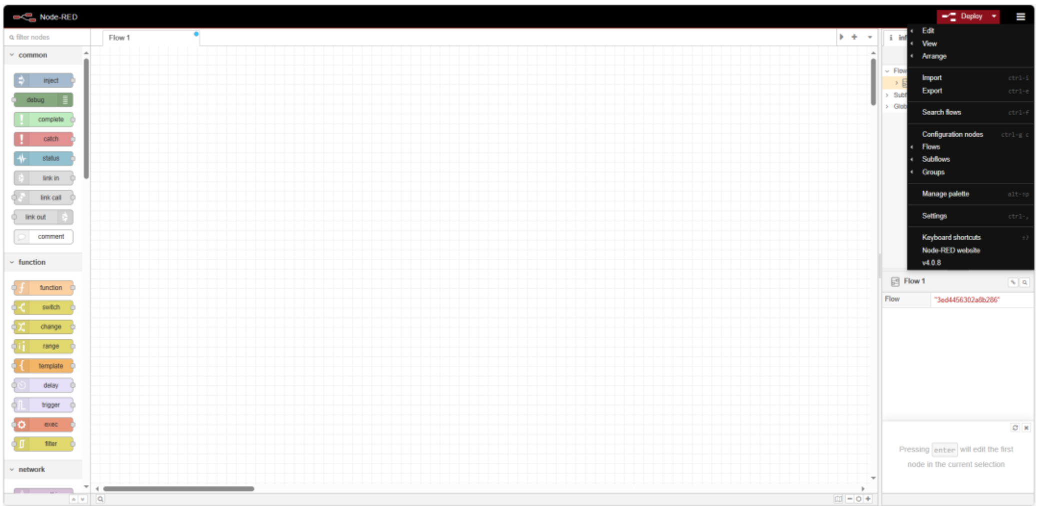
3.2 Connect Node-RED to InfluxDB

In order to keep the next example simple, we will use http as the protocol and we will not use a password. However, for a real system, the following note should be taken into account.

> Important note: If you access your Node-RED Editor via HTTP and have not set up password protection, you are putting your system in a critical security situation. Without HTTPS, data transmission is unencrypted, which means that sensitive information such as login credentials, configurations and flow information can potentially be intercepted by third parties. This makes it easier for man-in-the-middle attackers or unauthorised parties to access your entire Node-RED environment.
>
> Node-RED is often used to control automation and IoT processes. An unprotected editor not only gives attackers the ability to manipulate your operations, but also to execute commands that can take down entire systems. The lack of password protection means that anyone who knows your server address can make changes or access confidential control data - a scenario that can lead to significant operational and security risks.
>
> Use HTTPS for live systems: If possible, enable SSL/TLS in your Node-RED installation to encrypt traffic. Even if you are working behind a firewall, you should always protect access to the Node-RED Editor with a strong, complex password. Consider using additional measures such as VPN access or IP whitelisting to further restrict access.
>
> These security precautions are particularly important with Node-RED, as the system often controls critical processes that, in the wrong hands, could lead to damage to networked systems and major security incidents.

Node-RED is by default pre-installed in the sysWORXX-IO and can be started using the “systemd” service (see section 2.2). After starting navigating to: http://<device ip address>:1880 in the web browser.

To send data to the InfluxDB via Node-RED, a palette *node-red-contrib-influxdb* must first be installed. Open the menu and click to Manage palette.



Open menu

Figure 7: Node-RED editor website

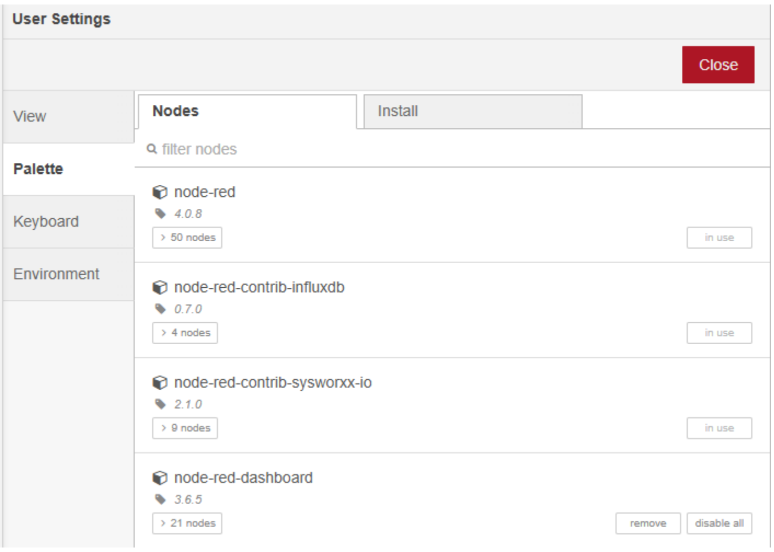


Figure 8: Node-RED installed nodes used by example

A corresponding flow can then be implemented. The subsequent flow cyclically sends values to the InfluxDB.

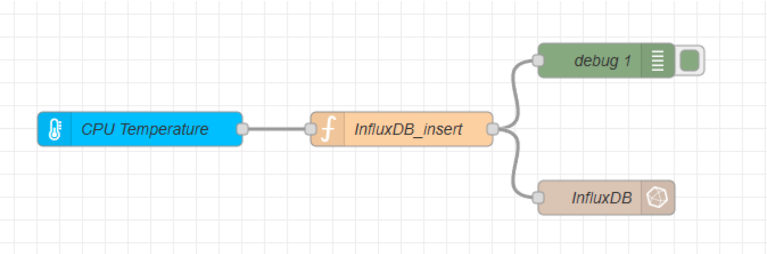
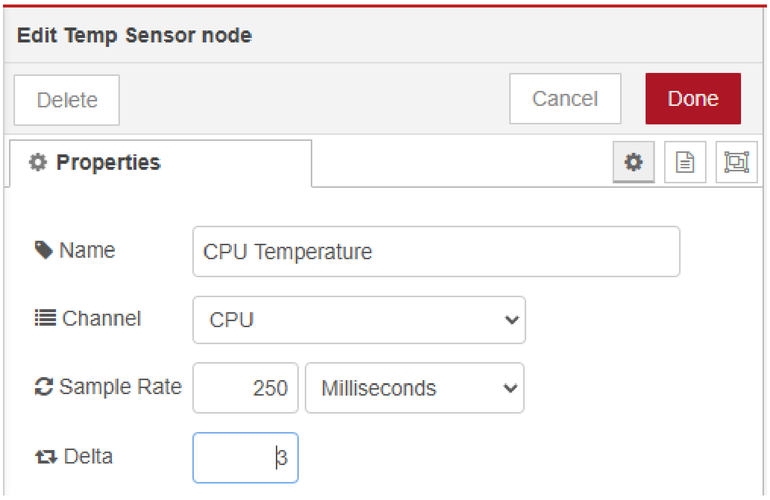


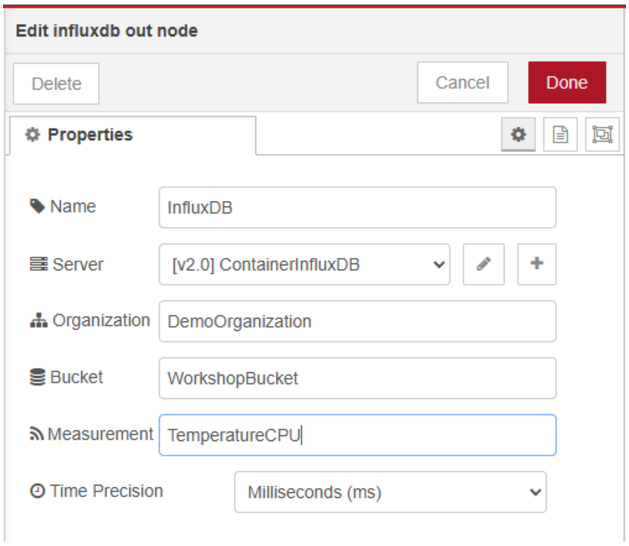
Figure 9: Node-RED example flow

Select *TempSensor* from the sysWORXX Node palette and configure the name and IO-channel:

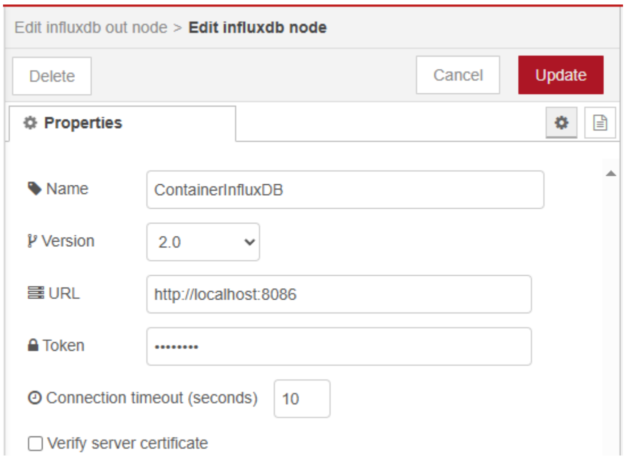


Add the *function* node and connect this node with the *TempSensor* node and the *influxdb out* node.

The *influxdb out* node has the following properties:

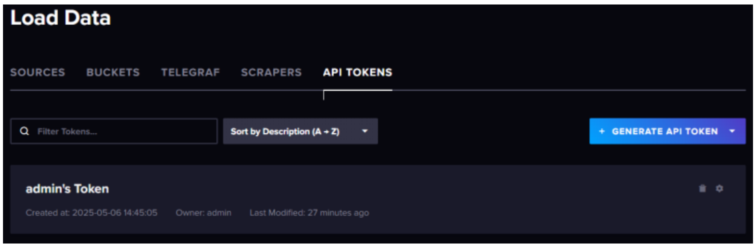


Enter the organization, the bucket name, and the metric name. To write the measured values to the database, the server must be initialized. This involves configuring the network connection and creating an API token.



In InfluxDB generate an API-Token for Node-RED. Follow the steps:

- Click *API Tokens*
- Click *GENERATE API TOKEN*
- Click *Custom API Token*
- Add a description
- Select the Resources and Authorization
- Generate the Token
- Save the Token and paste the Token in Node-RED influxdb out Server properties (see figure above)



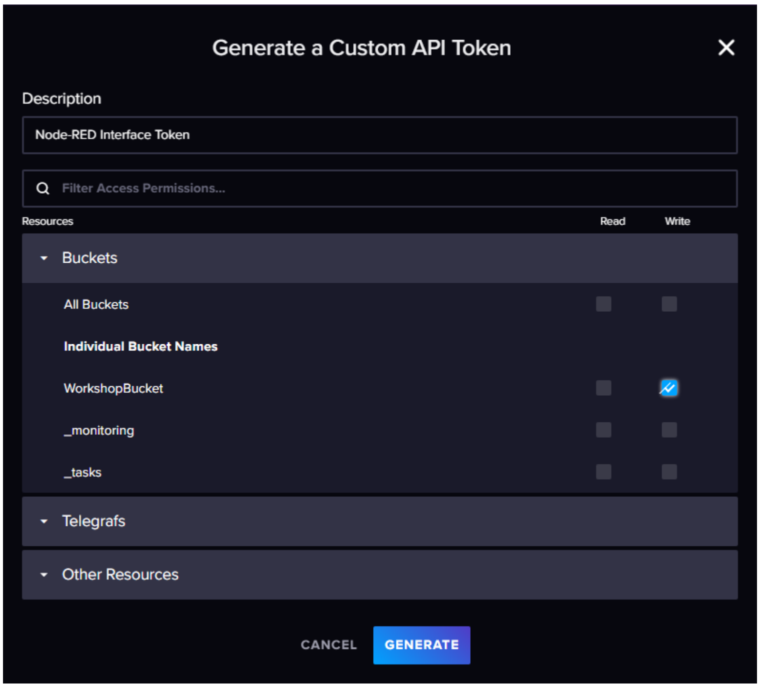


Figure 10: Generate an InfluxDB API Token

Close the Node configuration dialogs and deploy the flow to your device.

3.3 Connect Grafana to InfluxDB

The Docker containers Grafana and InfluxDB must be started see section 2.1. Once both containers are running, then follow the steps.

Step 1: Open Grafana in web browser

Access Grafana by navigating to http://<device ip address>:3000 in the web browser. The default login credentials are:

- Username: admin
- Password: sharedSecret

Figure 11: Grafana Login

Step 2: Add InfluxDB as a Data Source in Grafana

In InfluxDB generate an API-Token for the Data Source in Grafana. Follow the steps:

- Click *API Tokens*
- Click *GENERATE API TOKEN*
- Click *Custom API Token*
- Add a description
- Select the Resources
- Generate the Token
- Save the Token

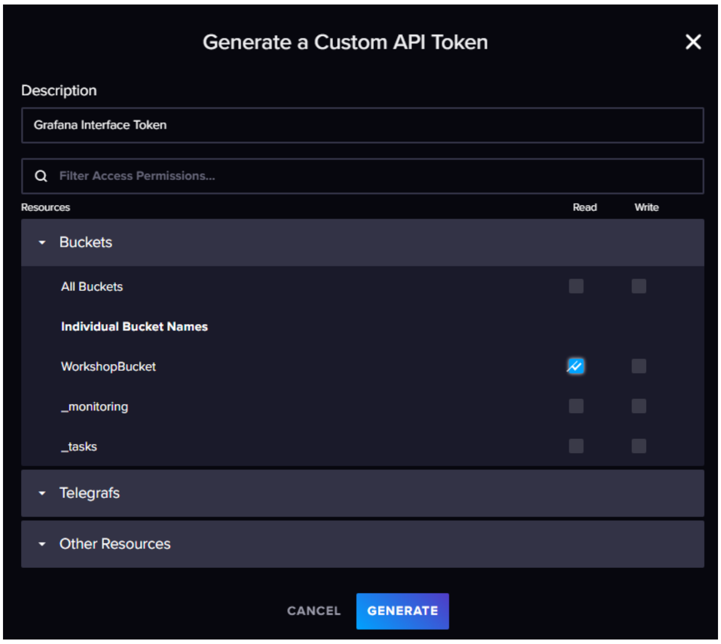


Figure 12: InfluxDB API Token

In Grafana, go to *Configuration* (the gear icon) and select *Data Sources*. Click on *Add data source* and choose *InfluxDB*. Fill in the details:

- Query language: Flux
- URL: http://influxdb:8086
- User: admin
- Password: sharedSecret
- Organization: DemoOrganization
- DefaultBucket: WorkshopBucket
- Token

Click *Save & Test* to verify the connection.

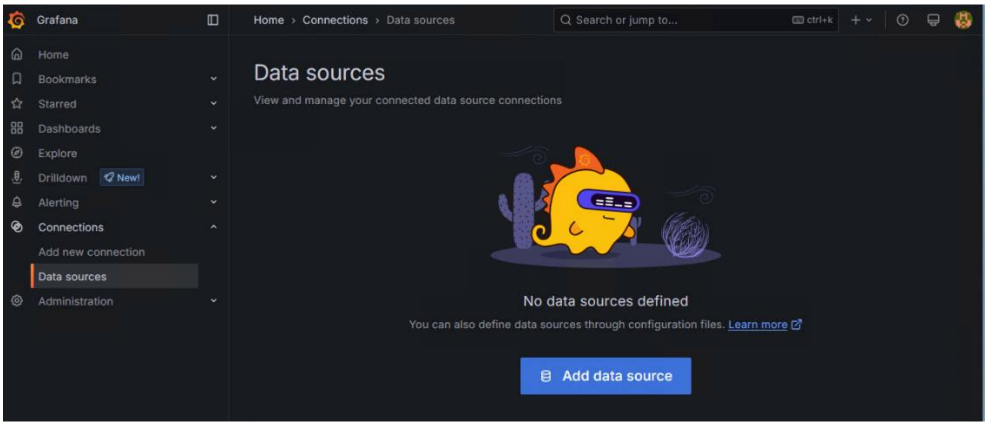


Figure 13: Grafana new Data Sources

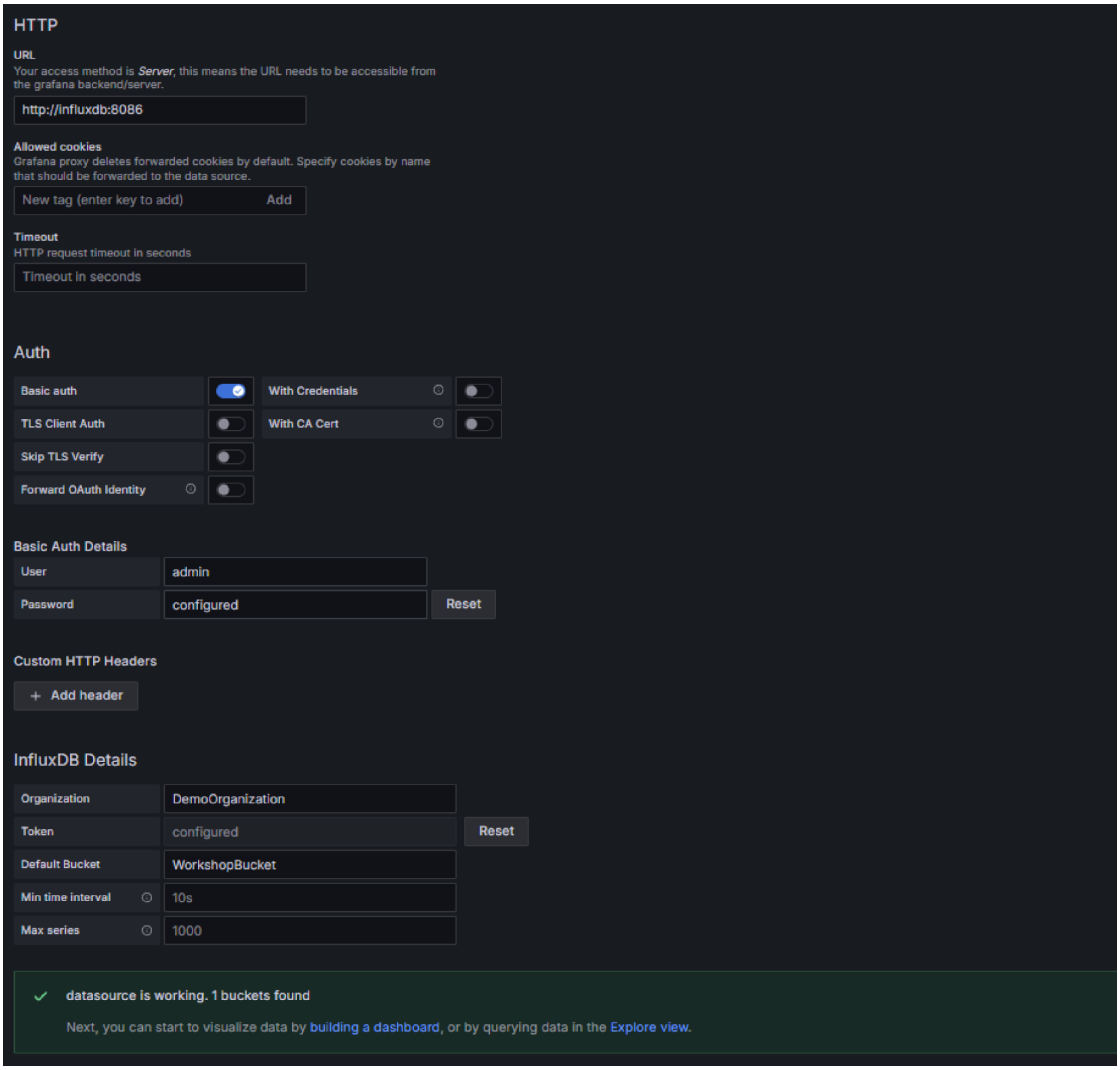


Figure 14: Grafana new Data Sources details

Step 3: Create a Dashboard in Grafana

Now once can create a dashboard in Grafana to visualize the data stored in InfluxDB. Once can add panels to display various metrics, such as CPU usage, memory usage, or any other time-series data that are collecting. Follow the steps:

- Click *Dashboards*
- Click *New*
- Click *Add Visualization*
- Click *Panel title/edit*
- Change Data Source in the screen middle from *default* to *Influxdb*

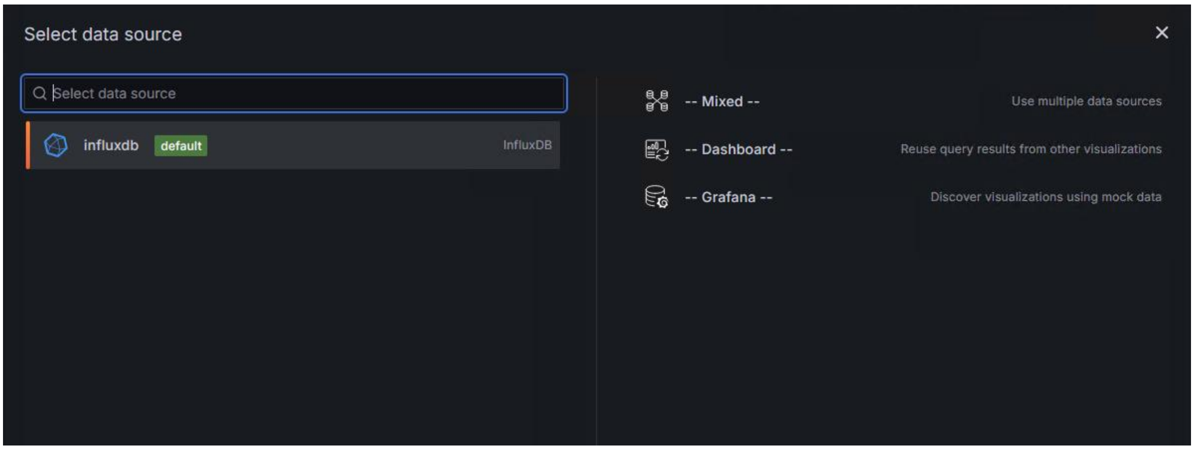
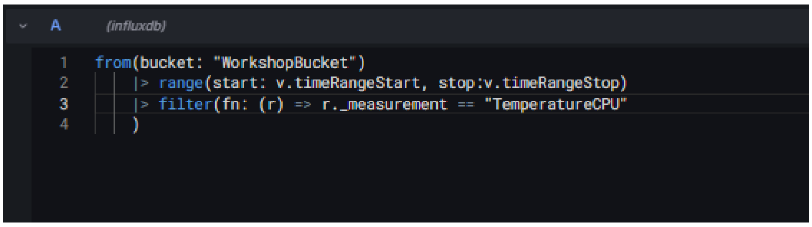


Figure 15: Grafana new Dashboard

Add influxdb query:

```
from(bucket: "WorkshopBucket")
   |> range (start: v.timeRangeStart, stop:v.timeRangeStop)
   |> filter(fn: (r) => r._measurement == "TemperatureCPU" )
```



Note: Sometimes the "" character causes a compile-time error. Delete the character and then add it again in the query code.

Select Refresh Time to 5s and the time range to last 30 minutes.

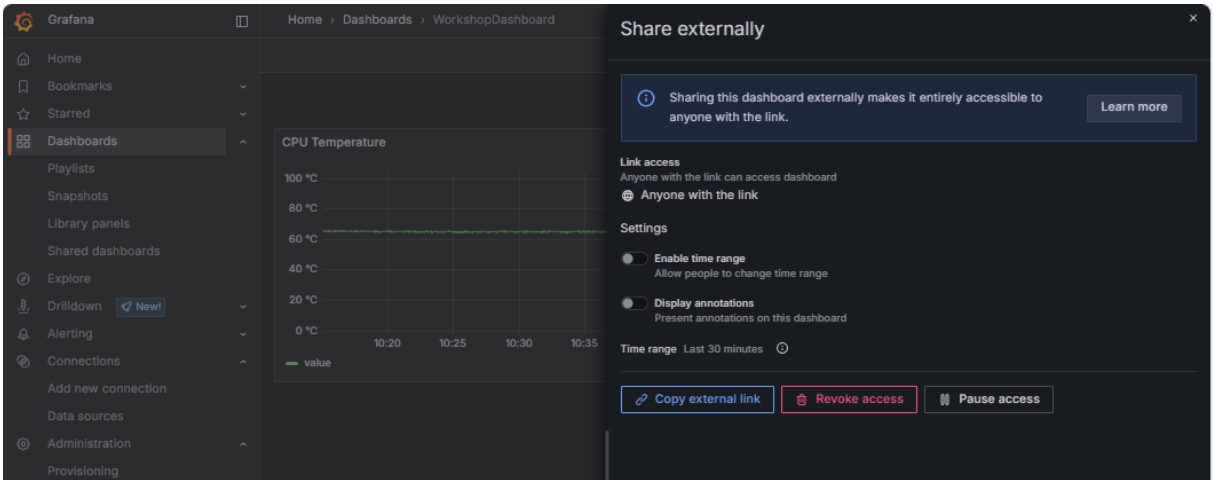
Figure 16: Grafana Dashboard with live data

3.4 View Dashboard in Browser

The Image contains the browser application Chromium. To use the browser, a background service *browser-hmi.service* must be started.

```
$  systemctl enable --now browser-hmi
```

Accept the security and copy the external link:



The service has some configuration options for different usage scenarios. To add the shard external link edit the service with:

```
$  EDITOR=nano systemctl edit browser-hmi.service
```

Note: `EDITOR=nano` changes the default editor to `nano`. Users which prefer using `vim` can ignore this part. In case the terminal emulator over UART produces strange output for TUI applications one could try to prepend `TERM=xterm` to fix this.

Important: The changes must be made in the section above the comment “### Edits below this comment will be discarded”.

To display the dashboard in the browser (Kiosk mode), the following information must be adapted. Modify the HMI URL to match the external link you copied.

```
### Editing /etc/systemd/system/browser-hmi.service.d/override.conf
### Anything between here and the comment below will become the contents of
the drop-in file

[Service]
Environment=HMI_URL="http://localhost:3000/public-
dashboards/2634c1b2697146c58455e7ce10841b8e"
Environment=HMI_ARG3="--kiosk"

### Edits below this comment will be discarded

### /usr/lib/systemd/system/browser-hmi.service
# # SPDX-License-Identifier: MIT
# # SPDX-FileCopyrightText: 2025 SYS TEC electronic AG <https://www.systec-
electronic.com/>
#
# [Unit]
# Description=Web Browser Based HMI
# Requires=NetworkManager-wait-online.service
# After=NetworkManager-wait-online.service
#
# [Service]
# User=root
# Environment=HMI_URL="file:///usr/share/systec-logo/systec_logo.html"
# Environment=HMI_ARG1="--user-data-dir=/home/root/.chromium"
# Environment=HMI_ARG2="--no-sandbox"
# Environment=HMI_ARG3="--kiosk"
# Environment=HMI_ARG4=""
…
```

Save the changes and restart the service.

```
$  systemctl restart browser-hmi.service
```

Now you should see the Dashboard in your browser display.

Type the next command to change the CPU temperature. This command generates load for each core so that the temperature rises and reduces the CPU clock frequency when the threshold of 85°C is reached.

```
$  burn-a53
```

Stop the command with CTRL+C.

The next command sets the CPU clock frequency depending on the required processing load.

```
$   cpufreq-set -c 0 -g conservative
```