App Developer Guide

Siemens Industrial Edge / V 1.2.1

https://support.industry.siemens.com/cs/ww/en/view/109795865

Siemens Industry Online Support

Legal information

Use of app examples

App examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The app examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The app examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective app example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the app examples used by technically trained personnel. Any change to the app examples is your responsibility. Sharing the app examples with third parties or copying the app examples or excerpts thereof is permitted only in combination with your own products. The app examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the app examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the app examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the app examples at any time without notice. In case of discrepancies between the suggestions in the app examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: https://www.siemens.com/industrialsecurity.

Table of contents

1 Introduction and Prerequisites

1.1 Introduction

This manual is a hands-on description for developers of how to create a working app for the Siemens Industrial Edge Platform. It starts with an introduction into the Siemens Industrial Edge Platform and Docker. This includes the architecture and network settings, as well as a brief overview on Docker. The next subsequent chapter guides developers about the required packages and access point form where they can be downloaded. After this, developers learn how to setup the development environment on a (virtual) machine and connect it with an own **GitHub** repository for the example app. After the setup of the development environment, the developer is doing a step-by-step development of his first created Industrial Edge app. This includes the steps of creating the app, the test of it and finally the deployment to the Industrial Edge.

The version of this document with V1.2.1. The first two numbers represent the version of the Industrial Edge released version. The last number is the revision number of the developer guide.

1.2 Prerequisites

This section will describe what are the prerequisites of this Industrial Edge app developer style guide.

- In your network environment you need an installed and running Industrial Edge Management (IEM) instance on a machine. If you do not have it right now, please have a look into the IE-Hub for the image of the IEM and the user documentation.
- For deploying your app to an Industrial Edge Device in section 5.6, it is necessary to have it onboarded in your IEM. If you do not have onboarded an IED right now, you must do this first. For more information of how to onboard your IED, please have a look into the official documentation.
- For using the used system apps in section 5.7, you need to install the following listed system apps on the IEM level. This is done via the IE Hub to install them to the IEM Catalog. It might be necessary to buy licenses for them as well as on Siemens Industry Mall.
    - IE Databus
    - IE Flow Creator
    - SIMATIC S7 Connector (if using with OPC UA Server in section 5.7)
- For the installation of additional plugins for the IE Flow Creator system app. The IED needs to have internet connection to download them.
- Additionally, for the use and configuration of the SIMATIC S7 Connector with an OPC UA server in section 5.7 an OPC UA server is required. For this case you need an own OPC UA Server as a real PLC or Simulation (SIMATIC S7-PLCSIM Advanced). A demo TIA Portal project is available here, which can be used to simulate a OPC UA server. The OPC UA server should be in the same network available and accessible.

2 Introduction and architectural overview of Industrial Edge Platform

This section gives you an introduction into the Siemens Industrial Edge Platform and required basic information about integrated technology stack.

At first a general overview about the architecture of the Industrial Edge is given. After the quick introduction into the Docker technology, which is part of the software stack, the network possibilities are explained. At the end of this section best practices and hints for your Industrial Edge development are illustrated.

2.1 Overview of the Industrial Edge Architecture

The Siemens Industrial Edge platform is the central infrastructure to integrate and expand connectivity for data exchanges between automation and IT systems. It combines local and high-performance data processing directly within the automation system with the benefits offered by the cloud. These benefits are app-based data analysis, data processing and infrastructure-as-a-service concepts with centralized update function.

The general architecture about the Siemens Industrial Edge platform is shown in Figure 2-1.

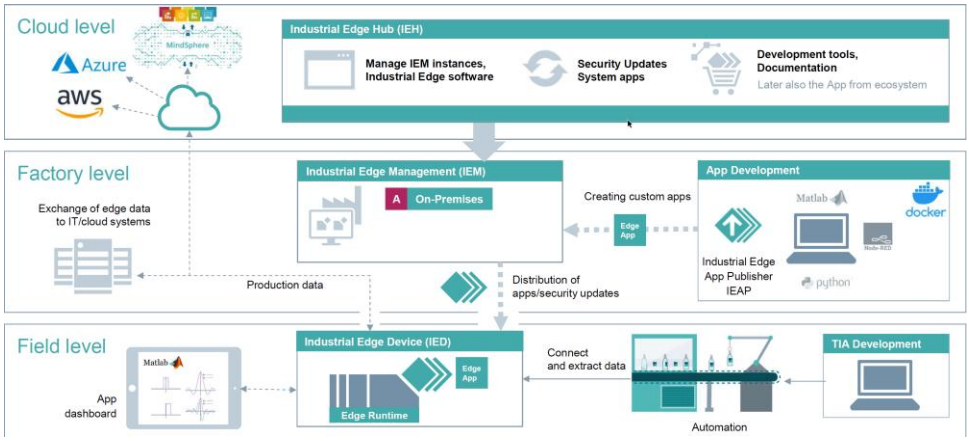

Figure 2-1: Architectural overview of Siemens Industrial Edge platform

The architecture can be divided into four main parts.

Industrial Edge Hub (IEH) – The IEH is the central entry point in the cloud level. It contains all the packages, tools and documentation about the Industrial Edge. The Industrial Edge Management (IEM) software can be downloaded and configured.

Industrial Edge Management (IEM) - The IEM is placed in the factory level. The IEM is the Industrial Edge Management system to manage your (connected) Edge Devices and Apps, which you can install individually on each Industrial Edge device. The tools on the IEM also contains next to the management of Industrial Edge Devices (IED) also tracking analytics for status reports. The IEM is running as a on-premises solution on a computer/ virtual machine in your factory.

Industrial Edge Device (IED) - The IED is placed in the field level, where the data generation from automation systems and acquisition of them take place. These devices can store automation data locally and retrieve it as needed. In addition, IEDs can upload these data to cloud infrastructures like MindSphere, AWS or Azure and retrieve it anytime. The IED is activated by a configuration file, which is created on the IEM.

Industrial Edge App – The Industrial Edge Apps are used for intelligent processing of automation data. These apps are available directly from Siemens,

other business partners (App Partners) or third-party vendors. It is also possible to develop own apps. The IEM is used to configure, deploy and maintain these (Docker) containerized Industrial Edge apps to the targeted Industrial Edge Device.

2.2 Quick overview about Docker

Docker is an open source technology to run, deploy and create apps using containers. Containers are started using images. An image is like a template for containers. A container contains everything a certain app needs for its execution like code, libraries, environment variables and other settings. The image is the static packaged app, which will be a container as soon as the image is run on the Docker Engine.

The IED contains a preinstalled Linux-based operating system and Docker, which enables all the capabilities of containerization. Container technology offers many advantages over using classical VM for isolating execution networks. Containers and Docker are enabling rapid development, portability, scalability and lightweight. A VM always contains a hardware abstraction layer as well as an own operating system. A Docker container is sharing the host´s kernel. The differences are shown in Figure 2-2.

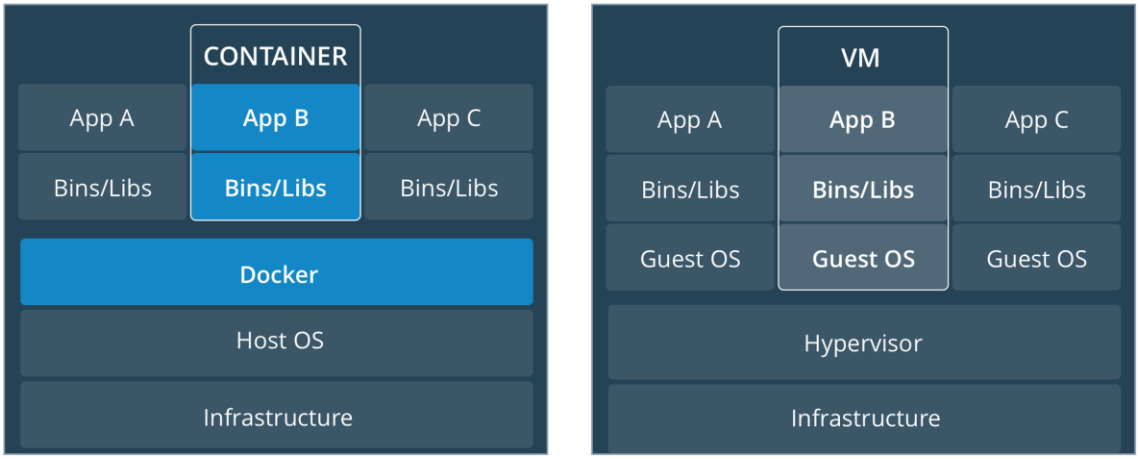

Figure 2-2: Stack overview of container and virtual machine

Ready-made images can be obtained from Docker Hub. From these images own containers can be derived to work with. It is also possible to combine multiple containers to a group with `docker-compose`.

The predefined images will mostly fit the requirements. The essential work for configuration, additional package installation or other settings can be done manually each time or be automated. To automate it, an own image is created. The required changes are done in a file called *Dockerfile*. With this Dockerfile it is possible to create an own local image after calling `docker build` command.

The procedure of creating a new image is done as follows. Create an own directory with the Dockerfile. Inside this file you can specify with instructions attributes for the image. `docker build` is creating the local image. Afterwards it is possible to publish the image to a public Docker-repository (Docker Hub) or a private Docker repository.

For more information about the Dockerfile syntax and best practices please have a look at https://docs.docker.com/engine/reference/builder/ and https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

To provide the images please refer to https://docs.docker.com/docker-hub/builds/ and https://docs.docker.com/registry/.

The container setup can be done by `docker-compose`. The command will evaluate the file *`docker-compose.yml`* in the current directory and configurates the container depending on the content. It is a practical way to combine multiple containers in a single setup.

The YAML-Syntax of the yml-file contains some rules:

- **#** starts a comment, till end of the line
- Strings are expressed with " or '. It is only necessary in special cases if the string contains special character or YAML expressions
- Multiple expressions starting with – are together a list.
- Key-Value pairs are written in `key: value`.
- | starts a text block, which can contain line breaks.
- > starts a text block, line breaks are ignored only empty lines are maintained

There are more expression available but for the docker-compose.yml the listed ones are sufficient.

To validate YAML files you can use online validators like http://www.yamllint.com/ or https://codebeautify.org/yaml-validator.

The docker-compose.yml file starts with the docker-compose file format version number. This version number describes which features can be used. After the version number the service keys configure the services. Then every service starts with an indented self-defined name. These services contain again indented key words to describe the service. The structure of a docker-compose.yml can be seen below:

```
# Basical structure of a docker-compose.yml

version: "2.4"

services:

      serviceName1:

            keyWord1: setting1

            keyWord2: setting2

            …

      serviceName2:

            …

      …
```

A complete reference about docker-compose.yml and its key words can be found under https://docs.docker.com/compose/compose-file/. With the Industrial Edge the support is limited to version 2.4.

This section is only explaining Docker essentials for the Industrial Edge environment. If you want to know more about Docker, we recommend an online training course like e.g. Pluralsight or LinkedIn Learning.

2.3 Network in Industrial Edge environment

A proxy (server) is a communication interface in a network, which is handling the communication between two computer systems. The main task of a proxy is the reception of a client request for a server. It will route the request with the proxy´s ip

address to the server. Therefore, no direct connection between client and server is established.

Proxy server could be realized in two directions, which are also shown in Figure 2-3. The initiator is always the client for example the client machines in the private network before the Forward-Proxy or the client from a public network in the Reverse-Proxy case.

- **Forward-Proxy**: A proxy server is installed between a private network and public network/internet. This configuration will help to prevent private clients from influences from public networks. Requests are taken by the proxy and redirected with proxy´s IP address to be transferred to the destination. The answer to this request is taken by the proxy and redirected to the private client. In the Industrial Edge, the forward proxy is routing all the outgoing communication from apps like loading Plugins or request data from the cloud.
- **Reverse-Proxy**: To protect webserver from direct access from the public network/internet, a proxy server can be preconnected. Public clients will not get direct access to the destination server. Requests are taken from clients and can be checked by security rules. The request can be sent to a server instance after checks. In the Industrial Edge, the reverse proxy is handling the incoming requests like opening the IE Flow Creator page or a Grafana dashboard for data visualization.

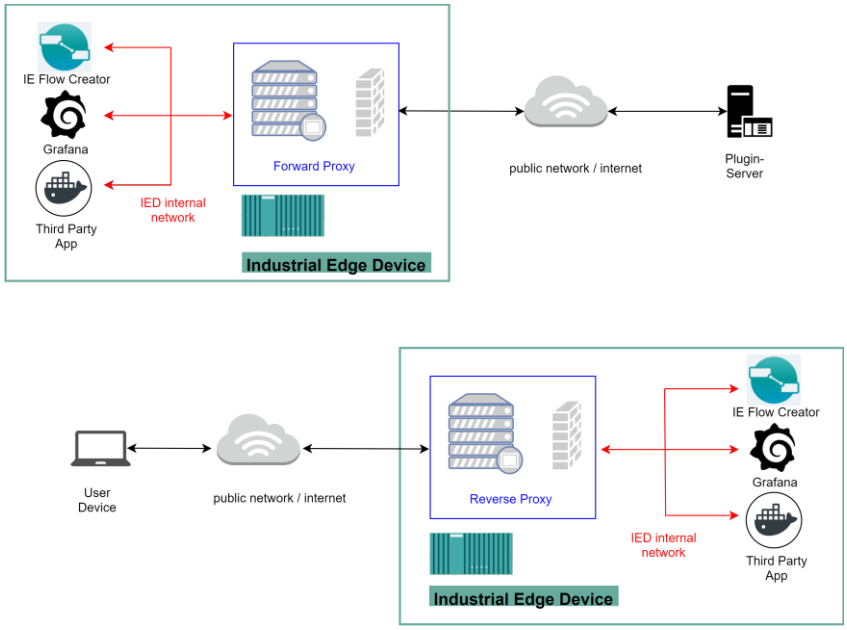

Figure 2-3: Overview about proxy server

Another important network scenario is in a real factory or process plant is that the Industrial Edge components are placed in separate networks. The Figure 2-4 shows a typical real-world scenario. In the plant network the Industrial Edge Devices are configured. The Industrial Edge Management system is placed in the control plane network. A state-of-the-art communication is done via a NAT-Gateway. This means no direct access from the control plane network is possible. Normal operations from the plant network are possible, as the Industrial Edge Device always initiates the communication from its side.

To allow direct access for example “remote access” to debug an Industrial Edge Device a relay server is required. With a relay server it is possible to allow initiating the communication from factory level as well.

In this case the Industrial Edge Devices in the plant network establish the connection to the relay server. The Industrial Edge Management system will act as a relay server for your Industrial Edge Devices. To allow this you must add the IP address of the Industrial Edge Management machine/server. Afterwards you can use the “remote access” through the Industrial Edge Management after activation. It is only a single relay server per Industrial Edge Management instance allowed.

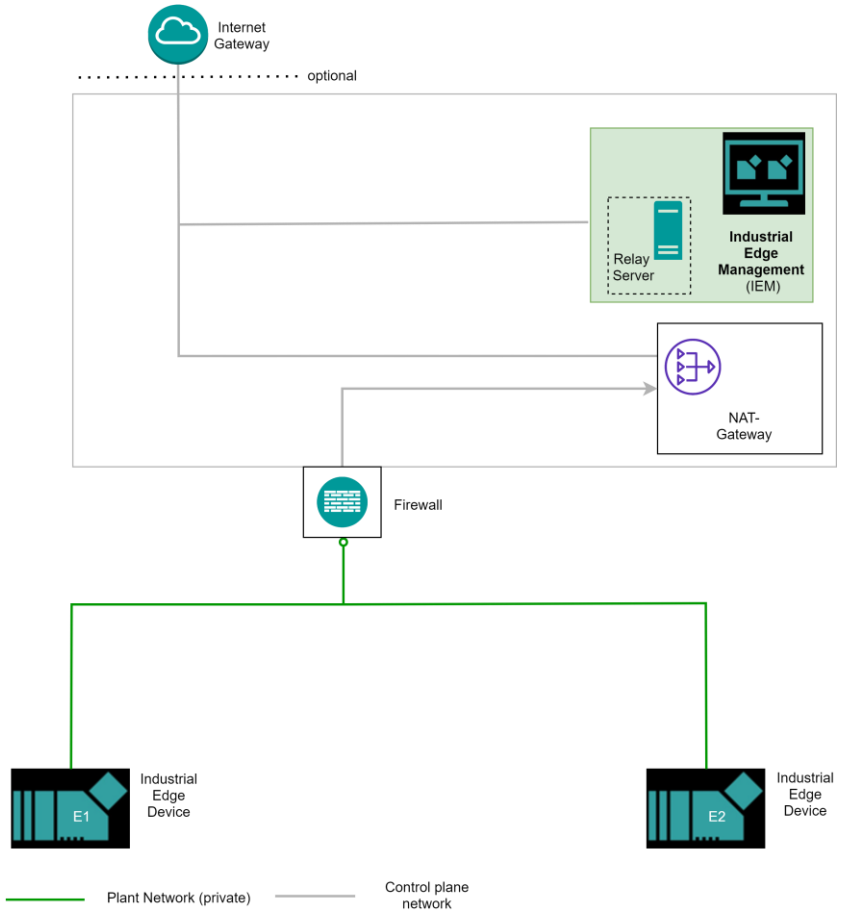

Figure 2-4: Industrial Edge components in separate networks with Relay-Server configuration

2.4 Network on Industrial Edge Device

On an Industrial Edge Device is one preconfigured network configuration available. This is the “proxy-redirect” network in the Docker network. Additionally, a service - the Industrial Edge Databus – is available to provide a communication interface between apps. Both are shown in Figure 2-5.

1. “proxy-redirect”

   The “proxy-redirect” is an internal, predefined virtual network in Docker on the IED. With this predefined network it is possible to communicate with system apps by joining this network. Please be aware that this network will not be created by your app as it as a preexisting virtual network on the IED. So, you must mark this network external in your docker-compose file of your app if you need a communication with system apps. This network should just used for the communication with the system apps over the Industrial Edge Databus. For Third-Party apps and any other

communication between containers except system apps, please use a separate network!

2. Industrial Edge Databus

   In combination with the network “proxy-redirect” the Industrial Edge Databus provides the interface between the system apps and your own apps. The Industrial Edge Databus provides a publish-subscribe-pattern based on the MQTT protocol. You can configure channels where you want to publish data to other apps. Other apps subscribe to these defined channels to consume the data.

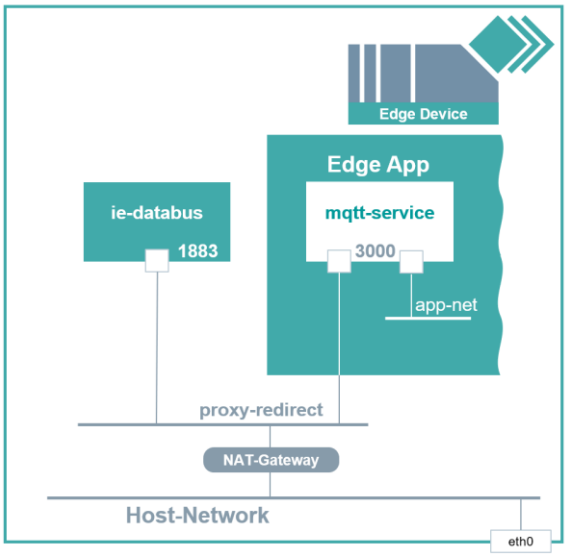

Figure 2-5: Network layout on an Industrial Edge Device

2.5 Network in app level

By default, Docker container are running in a private network. Docker is taking care about the management of it. These containers can communicate to each other or connected to other non-Docker workloads.

The Docker´s networking subsystem is using drivers. There are some existing drivers available by default which can be used in containers:

- `bridge` : It is the default network driver. This type will be automatically created if no driver is specified. Bridge networks are usually used when your apps run in standalone containers that need to communicate. The default IP address is in the range `172.17.0.0/16.` This might be different depending on the onboarding of your Industrial Edge Device.
- `host` : This driver removes the network isolation between the container and the Docker host. The host´s network is used directly. There is no port mapping and host ports are used directly. Normally no need to use host network at all in Industrial Edge application due to some exceptions.
- `none` : For this container all networking is disabled. Such a container is a closed container, because it works insulated on the host.

There are also other Third-party network plugins available from Docker Hub or other vendors.

For the Industrial Edge platform, the Docker bridge and is the most important one.

The communication between containers, between container and the host system (IED) and the communication to the internet (outside the box) is realized by a bridge network.

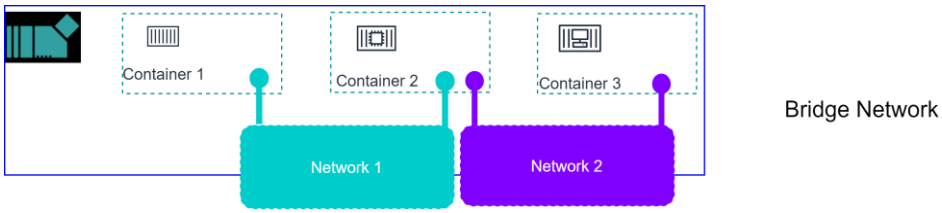

Figure 2-6: Important network between apps in Industrial Edge Environment

To administrate the Docker network functionality the command '`docker network`' is used. These commands will affect the Docker system at all and no single containers or images network.

In general, the configuration of network settings for container can be either done manually or automatically by using `docker-compose.`

For further information about the network topic on Docker, please refer to https://docs.docker.com/network/ .

Another important aspect of Docker when using bridge networks is publishing the container ports by exposing ports. This will allow to make Docker ports accessible by services from outside the host system. Exposing ports to other containers withing the same bridge network is done by **`EXPOSE`** or **`expose`**. The **`expose`** is an instruction for the docker-compose file and described here. The **`EXPOSE`** instruction is used in Dockerfile.

An example for the **`EXPOSE`** rule in a Dockerfile could look like this:

```
Expose 8080
Expose 80
```

In this example multiple ports (80 and 8080) are exposed. Unmapped exposed ports should be avoided in Industrial Edge apps. Without adding the port listening protocol as it is done on port 80, it takes the default configuration TCP. If you want to specify the protocol of the port add it as it is done for port 8080 in the example.

The **`EXPOSE`** is a documentation mechanism that provides information about which initial incoming ports will provide services. By default, the **`EXPOSE`** instruction does not expose the container´s ports to be accessible from the host. It only makes the stated ports available for inter-container interaction.

To publish the exposed ports there are two ways by using the **`-P`** and **`-p`** flag at runtime. The **-P** flag publish all exposed ports to random ports on the host interfaces and is the short form for **-publish-all**. The automatic mapping mechanism prevents potential port mapping conflicts. The **-p** flag publish a container´s specific port(s) to the Docker host. It allows to map a container´s port or a range of ports to the host explicitly. The format for the **-p** flag is '**`ip:hostPort:containerPort`**'. **ip** and **hostPort** references can be left out.

Here are some examples for using the **-p** flag:

Flag Value.	Description
`-p 8080:80`	Bind container´s TCP port 80 to host´s port 8080
`-p 192.0.2.1:8080:80`	Bind container´s TCP port 80 to host´s port 8080 for connections to host IP 192.0.2.1. By default, Docker binds published containers to 0.0.0.0 IP address, which matches any IP address on the system
`-p 2346-2348:2346-2348`	Specify hostPort and containerPort as a range of ports. Note: Number of container ports specified in the range should be equivalent to the number of host ports specified

2.6 Industrial Edge app development hints

This chapter will give you best practices and advice for your development of Industrial Edge apps.

- Always use Docker official or certified images for use in productive environments
- An image should always have a version and clearly defined tags. For CI/CD pipelines use the image digest instead of tag.
- Avoid using latest tagged images. It will check for newest release
- Keep your images small by build patterns (e.g. alpine) and install only required/necessary packages
- Divide your app in microservices. One container should only execute one single task.
- Use multistage builds to reduce container sizes. Further information can be found here.
- Comments on Dockerfile will help to improve understanding. Add an additional maintainer comment for every change in the file.
- Only expose ports when it is necessary
- Reduce security risks by not using root user inside of a container.
- Avoid using privileged mode in containers. Please specify capabilities of your app or container. For more information, please see here.

Small Docker images include less packages and other dependencies. This will reduce effort on open source software and license clearing as well as possible bugs and vulnerabilities. It will increase the upload of an app to the Industrial Edge Management as well as the update or deployment on Industrial Edge Devices. Be aware that on an IED is only a limited set of containers available. Please summarize your app by eye as services.

The Industrial Edge uses the native supported restart policies provided by Docker. The restart policies control whether a container start automatically when it exits or when the Industrial Edge Device restarts.

The restart policy options are described in the table below:

Restart option	Description
`no`	Default behavior. Not automatically restart of container.
`on-failure`	Restart the container if it exits due to an error, which manifests as a non-zero exit code.
`always`	Always restart the container if it stops. If it is stopped manually, it is restarted only when Docker daemon restarts or the container is manually restarted
`unless-stopped`	p to always, except that when the container is stopped, it is not restarted even after Docker daemon restart.

3 Industrial Edge Platform Overview

This section will show, how to get access to the Industrial Edge Hub and what kind of packages are required for the development environment.

3.1 Get access to Industrial Edge Hub

At first it is necessary to get the required access by buying a license at Siemens Industry Mall. For this step an account for the Industry Mall is required. Please register if you do not have an existing account.

If you have purchased the access to the Industrial Edge Hub (IE-Hub), you can open the IE-Hub by opening this link in your browser.

Please login with the provided credentials. After successfully logged in, you should see the overview page of the IE-Hub.

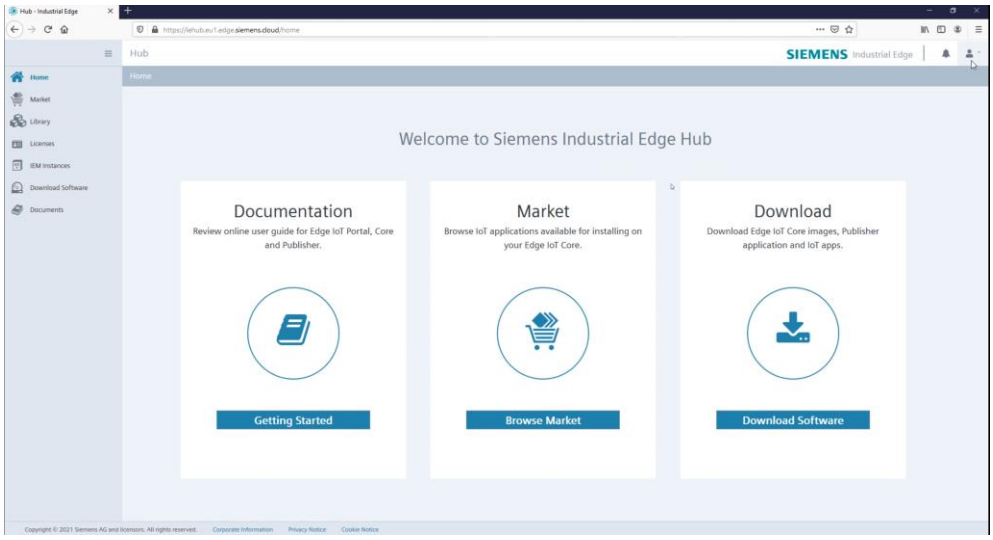

Figure 3-1: Start page of Industrial Edge Hub after login

3.2 Download required packages

On the IE Hub you can obtain the ISO file for setting up the IEM. The same applies to the Industrial Edge App Publisher software. As shown in Figure 3-2 the IEAP is available for Windows and Linux (Ubuntu) as well as Command Line Interface (CLI) for both operating systems. As a prerequisite for chapter 4 please download the Ubuntu package.

All these software products can be found under the ‘Download Software’ in the menu entries.

In Figure 3-3 you can see all the documentation regarding the apps of the Siemens Industrial Edge ecosystem. Please have a look in up-to-date manuals provided by Siemens to work on the current state of Industrial Edge knowledge base.

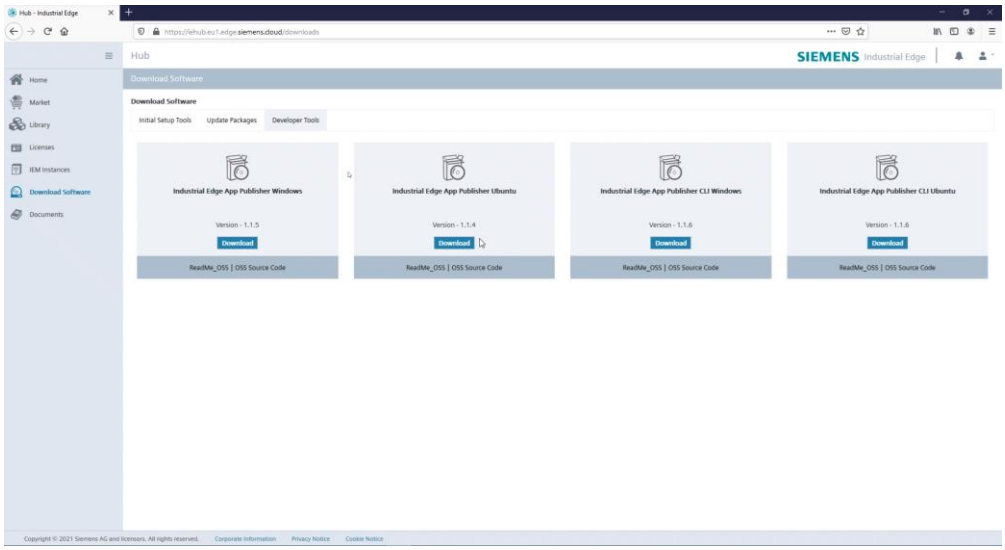

Figure 3-2: Download page of Industrial Edge App Publisher on IE Hub

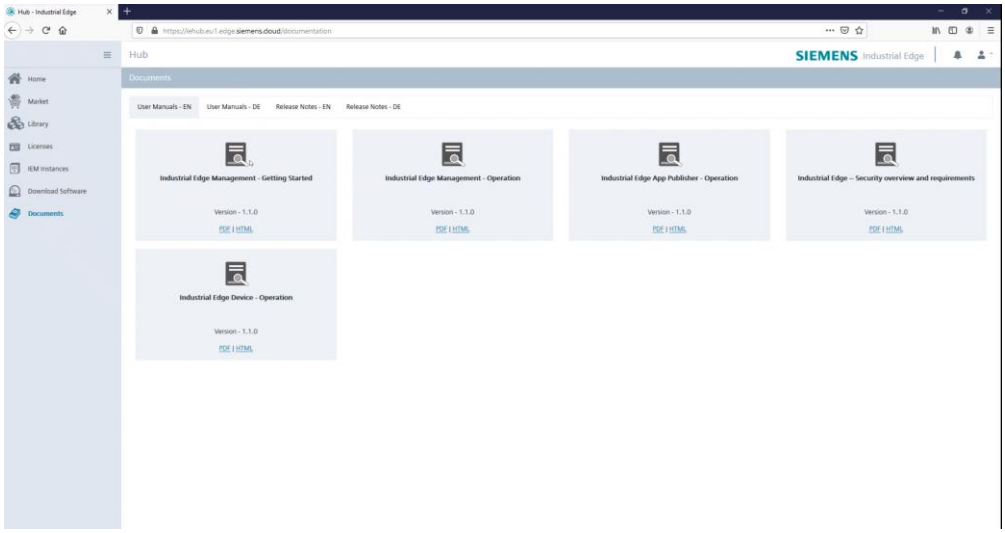

Figure 3-3: Available documentation of Industrial Edge in IE Hub

4 How to setup the development environment

This section starts with the required hardware and software prerequisites for setting up the development environment. We are creating a development (virtual) machine based on an Ubuntu Linux image. You can take any of your preferred Linux distribution as well. After the installation, the Linux system is configured with necessary packages. The next step is to setup a GitHub repository and connect it with your development environment. In the last step the installation and configuration of the Industrial Edge App Publisher (IEAP) is done to finish the setup of the development environment.

The creation of the development environment is done with a virtual Ubuntu machine on a VMware Workstation player.

4.1 Prerequisites

The minimum hardware specs for a development machine are suggested as the following:

- 2 CPU cores
- 4 GB of RAM
- At least 250GB of SSD storage

You can either run on a real hardware or on virtualized one by **VMware Workstation**. As an operating system you can either use a Windows or Linux, which is capable to run Docker apps.

The download of the required packages from chapter 3.2 is done in advance. Additional you need to download the OS image of your development machine. You can obtain the Ubuntu LTS version, which is the used one in this step-by-step setup from here. Please feel free to choose the operating system, which will fit you best.

For the development of your Docker app we recommend using a private GitHub repository. If you do not have an account on GitHub, please create one before chapter 4.4.

4.2 Create development environment

You need to install the downloaded ISO image of Ubuntu on your machine or virtual machine.

At first start the VMware Workstation app on the computer. Then click on ‘File → New virtual machine’ in the menu bar of the Workstation. A configuration dialog is opened.

The most important steps are shown in the figures Figure 4-1 and Figure 4-2. All the steps which are not shown use the default recommendations.

1. Select custom to get advanced configuration possibilities
2. Click Next
3. Select the downloaded ISO file of Ubuntu on file system
4. Click Next
5. Provide credentials and remember them (Login)
6. Click Next
7. Configure processors to requirements
8. Click Next

9. Set RAM of virtual machine to 4GB
10. Click Next
11. Set disk space to 250GB
12. Click next

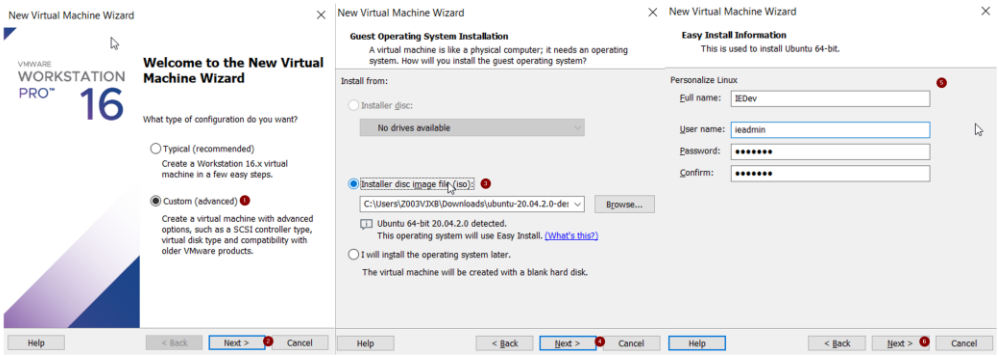

Figure 4-1: Create Ubuntu Development VM - Part 1

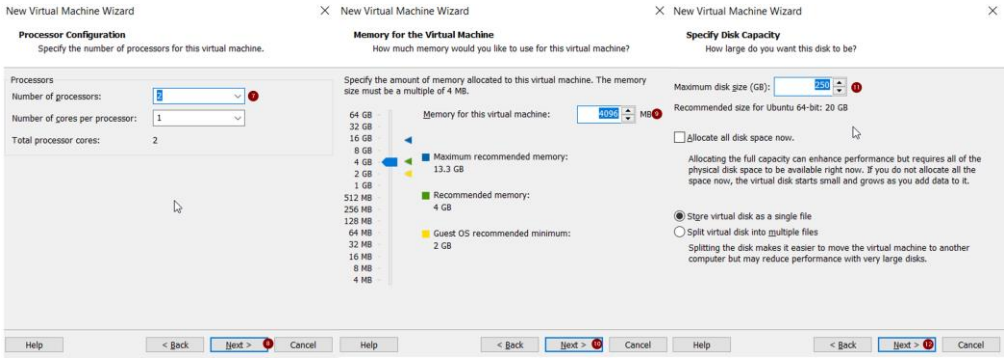

Figure 4-2: Create Ubuntu Development VM - Part 2

After completing the creation dialog for the new virtual machine, the virtual machine will power up and the installation of Ubuntu is done.

The installation will finish automatically, and you can login to do last configuration to the operating system. Please customize the operating system to your needs like languages and keyboard.

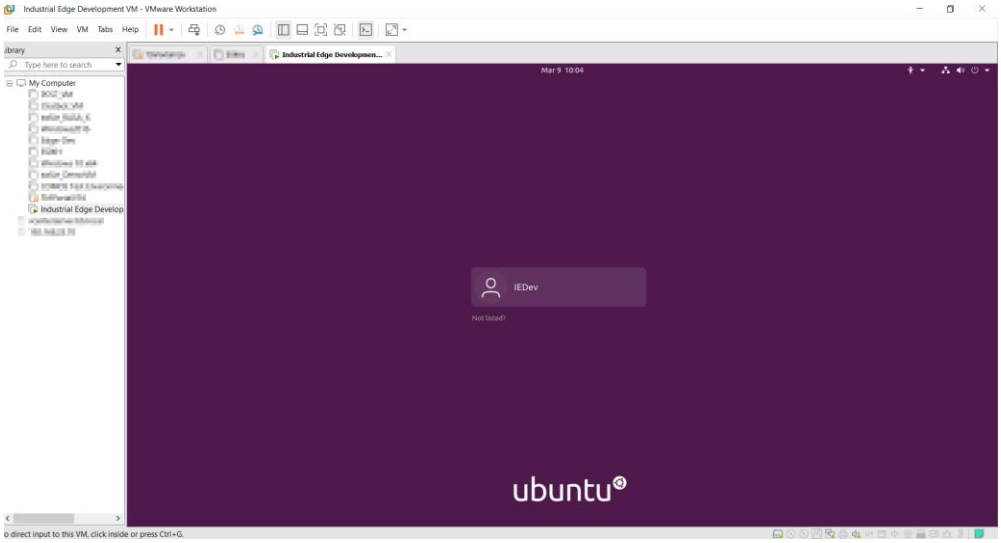

Figure 4-3: Login Screen of Ubuntu after successful installation

4.3 Install required packages on development environment

After chapter 4.2 the development machine is a blank Ubuntu installation. For development purpose for the Siemens Industrial Edge, some packages and tools need to be installed.

As a developer you need any development environment like Visual Studio Code or any other preferred tool, which suits you best.

In the Ubuntu Software Store, you can access such tools. We recommend and install the Visual Studio Code from the store.

We recommend installing some extension on the Visual Studio Code. Please start it and open the extension manager by clicking on  in the sidebar. The Docker and Python are essentials extensions for syntax highlighting. Please install them both as shown in figures Figure 4-4 and Figure 4-5.

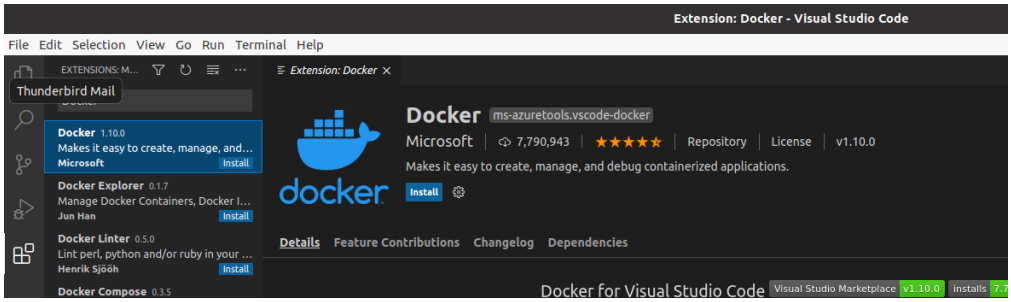

Figure 4-4: Docker extension in Visual Studio Code

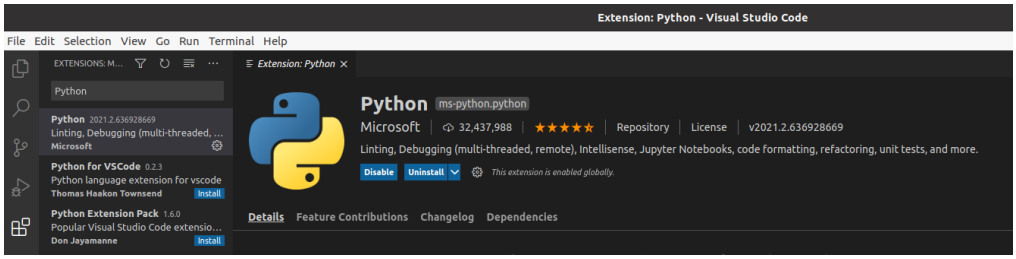

Figure 4-5: Python extension in Visual Studio Code

After the installation and configuration of the development environment, Docker needs to be installed with a minimum version of V18.09. In this guide we will use the V19.03 of Docker. The following steps show the installation on Ubuntu. For other operating systems or further information please have a look into the official docker documentation. **Important**: If you have a previous Docker version and want to upgrade, please uninstall it before installing a new version.

- Execute the steps of ‘Set up the repository’ in section ‘Install using the repository’
- Update ‘apt’ with `sudo apt-get update`
- List all versions of Docker repo with `apt-cache madison docker-ce`

```
root@ubuntu:/home/leadmin/Desktop# apt-cache madison docker-ce
docker-ce | 5:20.10.5~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:20.10.4~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:20.10.3~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:20.10.2~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:20.10.1~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:20.10.0~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.15~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.14~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.13~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.12~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.11~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.10~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
docker-ce | 5:19.03.9~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
```

Figure 4-6: Output of all available Docker versions

- Install Docker with version 19.03.x by replacing the version number in the command.

```
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING> containerd.io
```

- Configure Docker to be usable as a non-root user and automatically start at operating system start. Please see official documentation.

After the installation of Docker, the installation can be verified by using the command `docker run hello-world`. The result should look like the output in Figure 4-7.

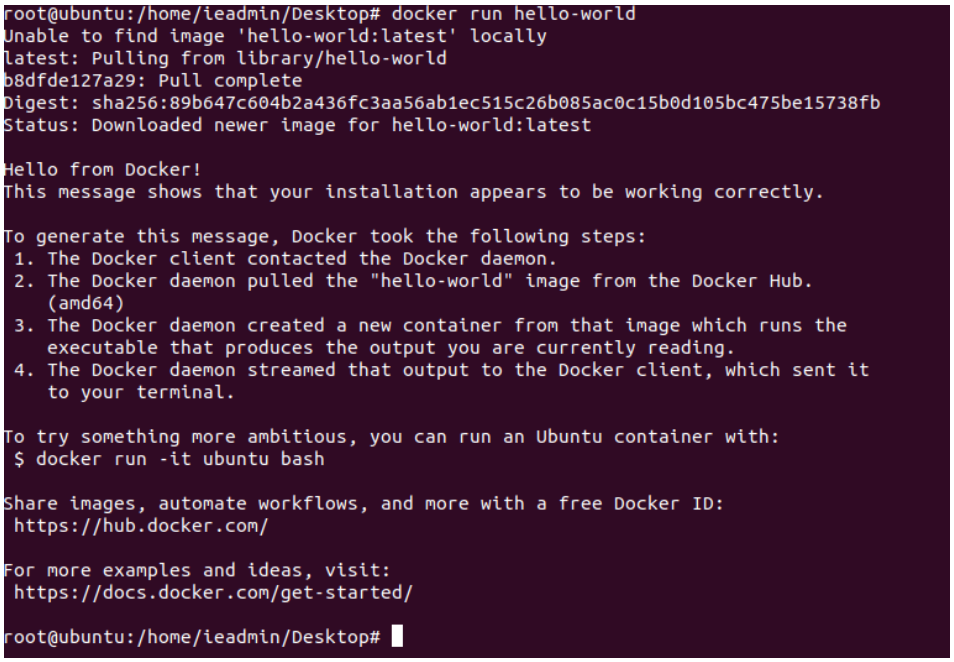

Figure 4-7: Test of Hello World after Docker installation

On Linux you must install the docker-compose explicitly. On Windows and macOS it is an integral part of Docker. For the Industrial Edge it is necessary to have a version of docker-compose file version between V2.0 and V2.4. To install the docker-compose on Linux, please follow the next steps:

- Please check the released version from https://github.com/docker/compose/releases. The latest and stable version will be installed.
- Open terminal and follow the installation steps for 'Install Compose' for Linux systems. You should change the '/usr/local/bin/' directory to '/usr/bin' as it is automatically part of the path variable. With '/usr/local/bin/' the path variable may have to be changed depending on Linux distribution.
- Test installation of docker-compose by `docker-compose --version`

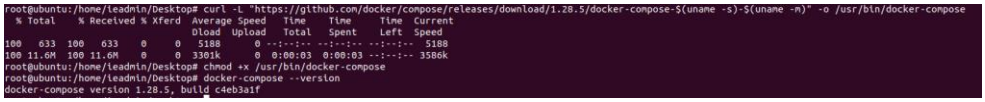

Figure 4-8: Docker-compose installation steps

Python is also required for the example app. First check if already a Python version V3.7.x and higher is installed. You can check this with the command after the quotas '`python3 -version`'. If the command is unknown or the wrong version is installed, please update or install a newer version.

Finally, the IEAP must be installed on the development machine. Copy the downloaded .deb package from the IE Hub to a folder on Ubuntu.

Install the package by executing the following commands

```
sudo -s
apt-get update
apt-get install -f industrial_edge_app_publisher_xxx.deb
```

After executing the commands, the IEAP is successfully installed as shown in console output. In Ubuntu you can also check it graphically as seen in Figure 4-9.

Figure 4-9: Graphical software overview in Ubuntu with IEAP

For more information about, how to install the IEAP, please have a look into the official documentation.

4.4 Setup GitHub Repository and connect to your development environment

In this guide we describe we use GitHub as repository for our app. You can use any other code hosting repository provider or your company's internal one. The steps might look similar. In these cases, we recommend reading the official documentation of the corresponding provider.

The official code repository of the Industrial Edge Computing platform from Siemens which contains open source projects and samples can be found here.

Please register or login with your GitHub credentials. For chapter 5 we need to create an empty public repository for GitHub Free. It must be made public to get support of GitHub Packages and a docker container registry of built images. More information about it and how to use it, can be found in the official documentation.

In the Figure 4-10 the steps for creating a new public repository on GitHub are shown.

1. Click on create new repository
2. Provide a name for your repository
3. Set Public property

4. Click on create repository to set it up

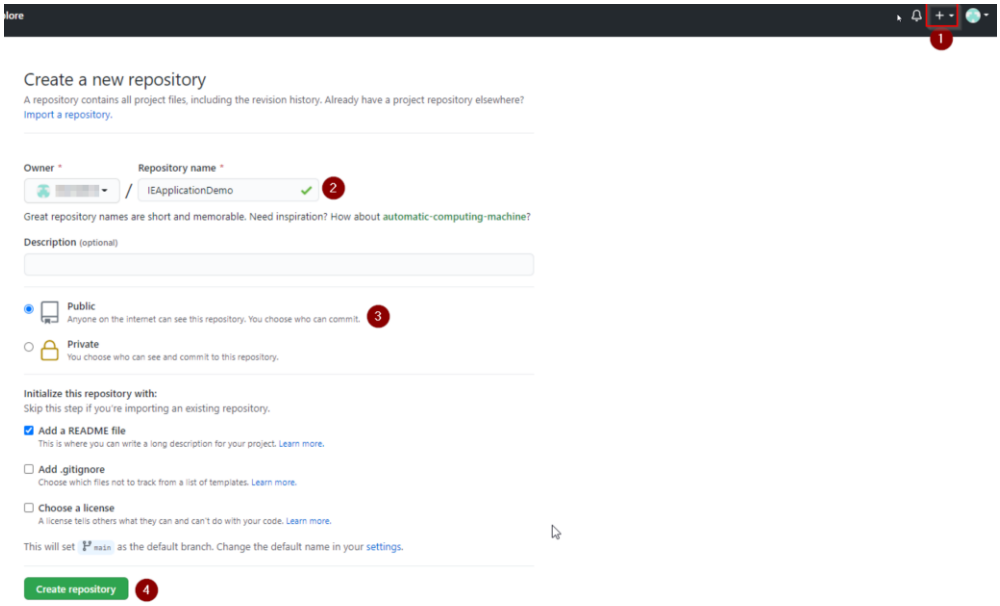

Figure 4-10: Create an own public repository on GitHub

To clone the repository to your development machine, you have to set up an access like SSH. To setup the SSH keys and connect them with your GitHub account please refer to the official documentation. Additionally, git needs to be configured on Ubuntu as well by terminal.

```
git config user.email "you@example.com"
git config user.name "Your Name"
```

After the configuration is done, you can clone this repository to your machine by opening a terminal on a folder of your choice. In the terminal execute the command '`git clone git@github.com:XXX`' with XXX depending on your account and repository name. The address can be found in the repository on GitHub.

A new folder with your repository name should be available and a README.md file is available.

You have successfully cloned your repository from GitHub.

4.5 Configuration of Industrial Edge App Publisher

Important notice for this step is that you have an installed and running IEM instance in your network available. If you do not have it right now, please have a look into the IE-Hub for the image of the IEM and the user documentation.

To work with the Industrial Edge App Publisher, you need to create one folder as a workspace root folder.

Start IEAP app. It will request to provide a workspace. Select the newly created folder for this.

Configure your Docker CLI to be reachable via the TCP socket in your docker daemon. This is described here in section 'Configuring remote access with systemd unit file'. We recommend using the description with the `systemctl` unit file.

Important: Please be aware that opening a socket might be a security threat because your system can be taken over by having access to this socket.

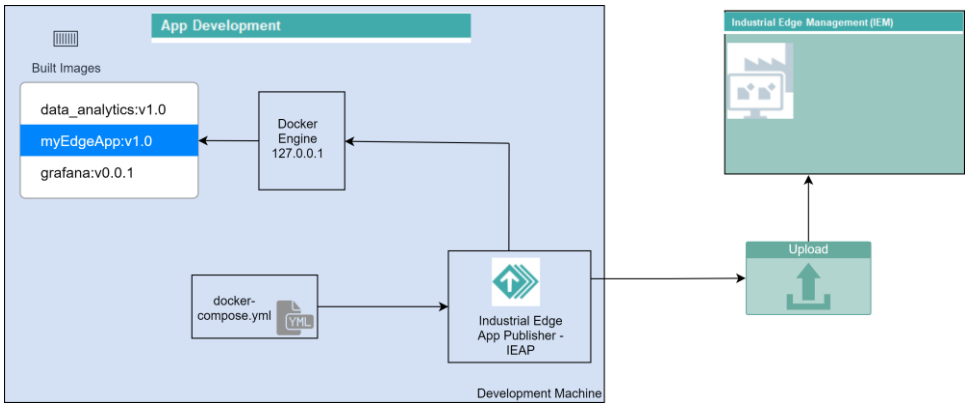

Figure 4-11: Overview of the working mechanism of IEAP and development environment

This step is essential to be able to upload your app to the IEM. The IEAP gets the prebuilt image of the app, which shall be uploaded to the IEM, from the adapted docker-compose.yml file for the Industrial Edge. Therefore for this selection it is important to make the Docker Engine available by this previous configuration step.

Add the Docker Engine to the IEAP of your development environment. This is required to directly integrate the images from there. This is shown in Figure 4-12 and Figure 4-13.

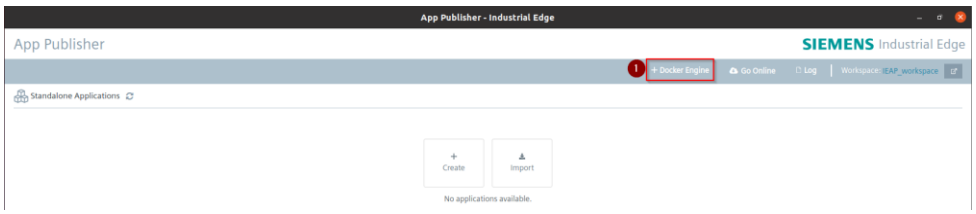

Figure 4-12: Create new local connection to Docker Engine with IEAP

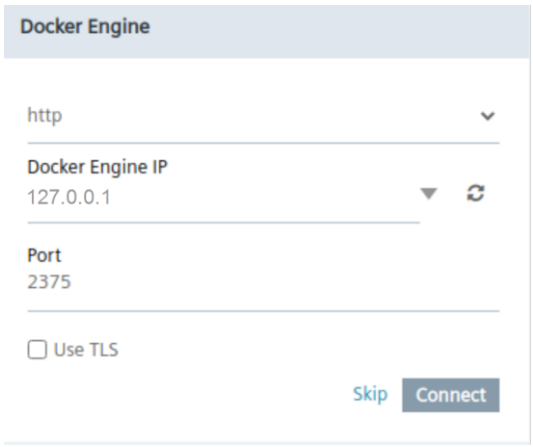

Figure 4-13: Configuration of Docker Engine connection

In the next step you click on “Go Online” to connect to the IEM system. A new window will open to enter the IEM URL – see Figure 4-14. Then you need to provide your personal credentials to login. If you have a proxy in your network, please check on these settings!

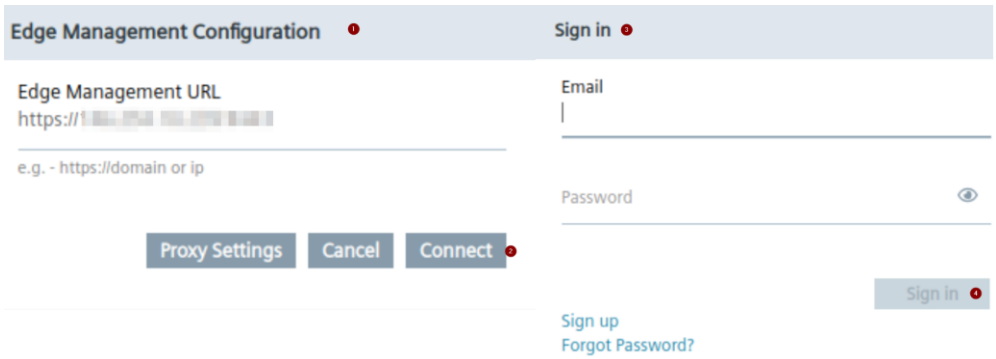

Figure 4-14: Go-Online steps of IEAP to connect with IEM

After you connect to the IEM you should see 2 views, on the top part are your local only apps and in the bottom view the apps loaded in the IEM.

For further Information to setup the IEAP, please see the official documentation.

5 Development of an Industrial Edge App example

The development process could look like the workflow in Figure 5-1.

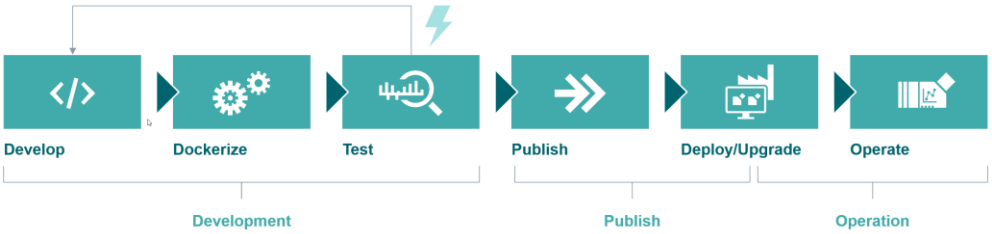

Figure 5-1: Development workflow for an Industrial Edge App

The development phase starts with the development of your own app by coding.

After your implementation the dockerize starts. It contains to write your Dockerfile for your app containing requirements and dependencies of required official images to create a Docker image. Then the docker-compose is done to finish the dockerize step. Next will be testing the app and containers in our development environment. If it is successfully it can be published, deployed, and operated on Industrial Edge Devices. If it is failing, the errors will be fixed in the development stage again.

5.1 Introduction – Use Cases of Industrial Edge

A typical app scenario is to preprocess the data and send the data to the cloud. In this scenario the Industrial Edge is used as a gateway with preprocessing functionality.

However, the Industrial Edge is not only a gateway. The most common use case, which will be implemented in this guide, is to acquire, store, analyze and visualize the data to improve the transparency on the shopfloor.

Especially, from a service point of view a typical scenario is to acquire, preprocess, store and export the data for external usage such as for data analytics services.

The Figure 5-2 shows the common app scenarios for the Industrial Edge.

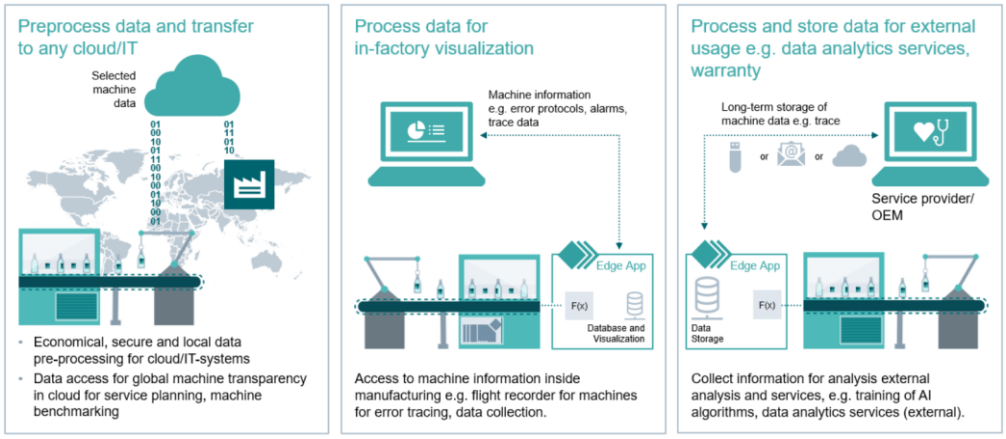

Figure 5-2: Common app scenarios in the Industrial Edge

5.2 Architectural overview of the app

As the example app will cover the most common use case in the Industrial Edge environment, the app on the Industrial Edge Device will look like the architectural

overview in Figure 5-3. The goal of the app will be to collect, process and store data from an OPC UA Server, which provides data from a PLC.

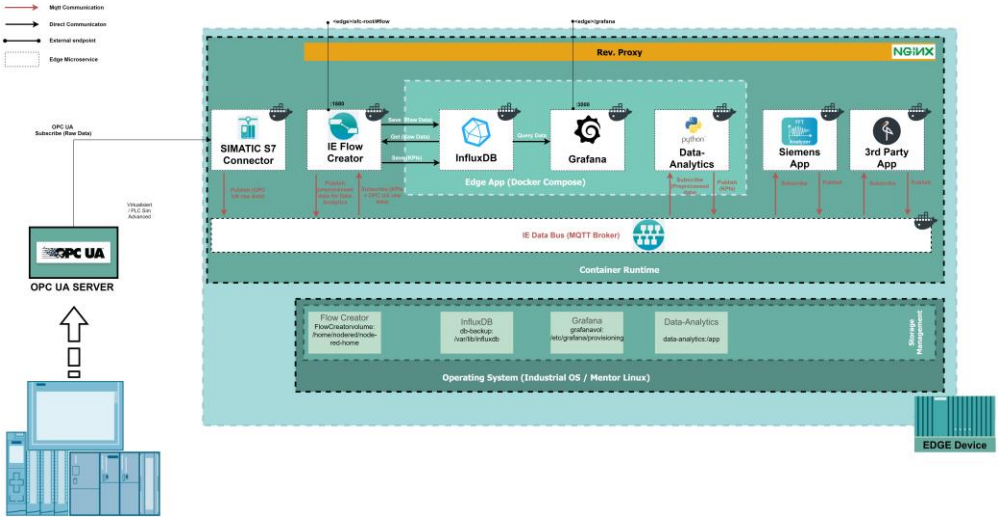

Figure 5-3: Example app architecture overview on Industrial Edge Device

The app contains three parts – the connectivity to collect the data from the OPC UA Server by system apps, the IE Databus for distributions of the data and the process, storing and visualization of data in the Edge App. In the following list of internal and external services of our app example are explained in more details.

1. The IE Databus based on MQTT is responsible to distribute data to certain topics, which are filled by system or custom apps by publishing and subscribing to these topics.
2. To receive the data from the OPC UA server, which is providing data from a PLC, the SIMATIC S7 Connector connectivity is used. SIMATIC S7 Connector is a system app, which publish the data to IE Databus. Another system app, the IE Flow Creator, consumes the data from the SIMATIC S7 Connector topics on the IE Databus. The data is preprocessed in the IE Flow Creator and published again on the IE Databus.
3. A system app called IE Flow Creator provides preprocessed data on certain topics which are consumed by a developed Python data analytics container. The Python data analytics is doing calculations and evaluations and provides the results like KPIs back on the IE Databus. The data analytics container needs a MQTT client to handle the publishes and subscriptions of the IE Databus.
4. The IE Flow Creator consumes the analyzed data again. The IE Flow Creator stores the (raw) data and analyzed data to a InfluxDB database persistently.
5. The InfluxDB is a time series database which is optimized for fast, high-availability storage and retrieval of time series data. It stores the data, which is transmitted by the OPC UA server to the app as well as the analyzed data.
6. Grafana is a visualization and analytics software. It allows to query, visualize, alert and explore metrics. It provides tools to turn time-series database data to graphs and visualization. There is the possibility to build custom dashboards. Grafana is used to visualize the data from the InfluxDB database

On the development machine you cannot use the system apps which are provided by the Industrial Edge. For the testing purpose on the development environment the architecture of the app looks a little bit different as shown in Figure 5-4.

The IE Databus will be a normal MQTT Broker to simulate the behavior of it. The SIMATIC S7 Connector and Flow Creator are replaced by a Node-RED container. Node-RED will create some dummy data, which simulates the receiving of the OPC UA data by the SIMATIC S7 Connector and preprocessing for distributing it for the data analytics container. The writing of the data to the database is also done by Node-RED container.

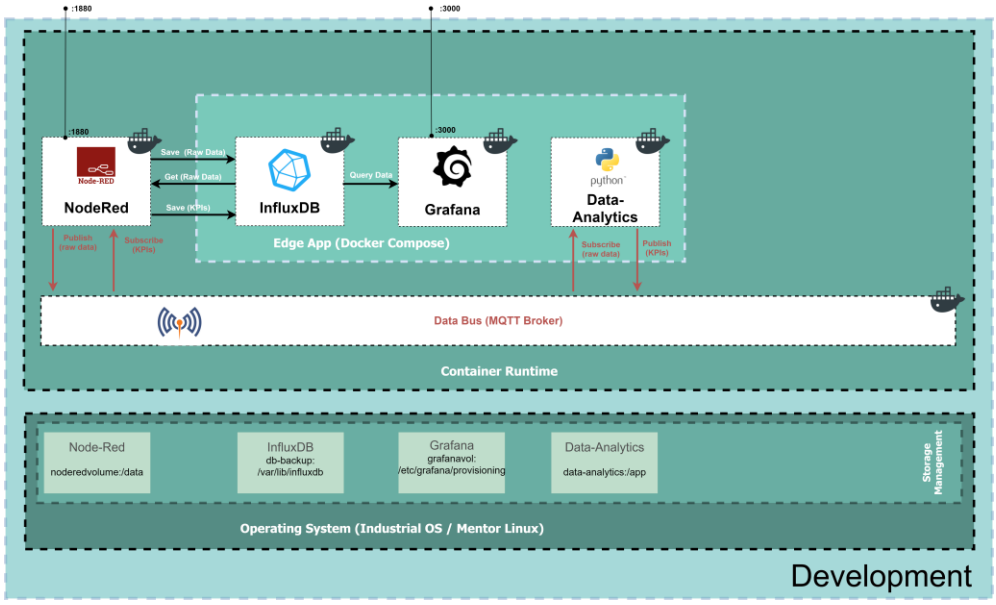

Figure 5-4: Example app architecture overview on development environment

5.3 Setup example app on your development environment

The required sources can be cloned from the official Siemens Industry GitHub team. The repository for cloning can be found here.

Please clone the repository to a personal folder on your development environment.

The sample app in the ‘src’ folder contains the following folder structure, which is shown in Figure 5-5.

Figure 5-5: Folder structure of your first Industrial Edge app

All the required folders of your first app are found in the “solution” folder.

Important: Please be aware that .env files are used and if you are using Docker console in a development environment from a root folder it might not work!

5.3.1 Description of Mosquitto MQTT broker

The Mosquitto MQTT broker container is the IE databus replacement on the development machine. All necessary files for the MQTT broker container are placed in ‘mqtt_broker_mosquitto’ folder.

It contains a docker-compose.yml and a hidden .env file.

```
### Docker Compose File for MQTT Broker - Replacement of IE Databus ###
# This docker-compose file creates a preconfigured MQTT Broker container without authentication


version: '2.4'                              # docker-compose version is set to 2.4

services:

    mqtt-broker:
      image: eclipse-mosquitto:$MQTT_VERSION  # define image to pull from docker hub if not already on your machine available
      container_name: ie_databus              # Name of MQTT broker container
      restart: unless-stopped                 # always restarts (see overview page 12 Industrial Edge Developer Guide)
      logging:                                # allow logging
        options:                              # we use best pactice here as limiting file size and rolling mechanism
          max-size: "10m"                     # File size is 10MB
          max-file: "2"                       # only 2 files created before rolling mechanism applies
      ports:                                  # expose of ports and publish
        - "33083:1883"                        # map containers default MQTT port (1883) to host's port 33083
      networks:                               # define networks connected to container 'mqtt-broker'
        proxy-redirect:                       # Name of the network

###### NETWORK CONFIG ######
networks:                           # Network interface configuration
  proxy-redirect:                   # Reference 'proxy-redirect' as predefined network
    external:
      name: proxy-redirect
    driver: bridge
```

It uses a predefined package which is initially loaded from DockerHub of the ‘eclipse-mosquitto’ in the corresponding version. The container name is the same as the IE databus with ‘ie_databus’ on the IED. The network interface is the ‘proxy-redirect’, which is marked as an external bridged Docker network as it already exists on the IED.

5.3.2 Description of the Node-RED

The Node-RED is the replacement of the SIMATIC S7 Connector and IE Flow Creator on the IED. All the required files for the Node-RED container are in the ‘node_red’ folder.

It contains a docker-compose.yml file, (hidden) .env file and another folder ‘node-red’.

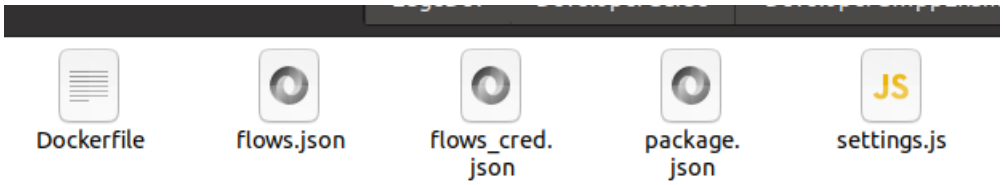

Figure 5-6: Additional files for Node-RED container

The Dockerfile in Figure 5-6 contains how to build the image of the Node-RED container. It describes which version of Node-RED is used and which Plugins will be loaded as well to make configurations like copying the example workflow.

The ‘flows.json’ file contains the app flow of our data acquisition, distribution to ‘data-analytics’ as well as storing to the Influx database. The flow will be described in more detail in 5.4.

```
### Docker Compose File for node-red - Replacement of Southbound and SIMATIC Flow Creator ###
# This docker-compose file creates a preconfigured NodeRed container MQTT connection


version: '2.4'                              # docker-compose version is set to 2.4

services:

  ###### NODE-RED ######
  nodered:
    build:                                  # Configuration applied at build time
      context: ./node-red                   # Relative Path to node-red from this docker-compose file containing Dockerfile
      args:                                 # Args variables available only at build-time
        no_proxy: $no_proxy
        http_proxy: $http_proxy             # Proxy url's from environment
        https_proxy: $https_proxy
    image: nodered:v0.0.1                   # Name of the built image
    container_name: nodered                 # Name of the node-red container
    restart: unless-stopped                 # always restarts (see overview page 12 Industrial Edge Developer Guide)
    environment:                            # Environment variables available at container run-time
      http_proxy: $http_proxy               # Proxy url's from environment
      https_proxy: $https_proxy
    logging:                                # allow logging
      options:                              # we use best pactice here as limiting file size and rolling mechanism
        max-size: "10m"                     # File size is 10MB
        max-file: "2"                       # only 2 files created before rolling mechanism applies
    ports:                                  # expose of ports and publish
      - "33080:1880"                        # map containers port 33080 to host's port 1880
    networks:                               # define networks connected to container 'data-analytics'
      proxy-redirect:                       # Name of the network
    external_links:                          # Dependencie on other container
      - influxdb                             # Wait for start of container 'influxdb'

####### NETWORK CONFIG ######
networks:                           # Network interface configuration
  proxy-redirect:                   # Reference 'proxy-redirect' as predefined network
    external:
      name: proxy-redirect
    driver: bridge
```

The docker-compose.yml contains a build step to build from the Dockerfile the Node-RED image. It is necessary to expose the ports for Node-RED to get access to the web view of Node-RED for accessing the flow. It has the same network interface as the MQTT broker from 5.3.1. The ‘nodered’ container is waiting on the ‘influxdb’ container to be started before it is starting itself.

Important notice: Please be aware about the variables for the HTTP and HTTPS proxy. You must set them in the .env file. The default value is empty. This means no proxy is use. Please configure your proxy if you have one used as e.g., in most company network environments.

5.3.3 Description of your first ‘Industrial Edge’ app

This is the app, which will be published to the IEM and deployed on IEDs. All the files which are required for your first IE app is found in the ‘my_edge_app’ folder.

Figure 5-7: Folder structure of your first IE app

The ‘my_edge_app’ folder contains the following files:

- docker-compose.yml: This is the docker-compose.yml to create the containers for the development environment and the image for the Industrial Edge.
- docker-compose_Edge.yml: This is the adapted docker-compose.yml file, which will be used later for uploading your first app to the IEM with the IEAP in section 5.5.
- data-analytics folder: This folder contains the Python app to do scientific calculations with the collected and preprocessed data. It contains the Dockerfile to build the corresponding image as well.

- grafana folder: This folder contains the Dockerfile to build the Grafana image as well as the preconfigured dashboard for your first app to display the data.

The Dockerfile in the ‘grafana’ contains the build steps for the grafana image. It will load grafana itself as well as plugins. Another important step is copying the grafana.ini file as well as the folders ‘dashboards’ and ‘provisioning’.

The ‘data-analytics’ is built by the Dockerfile in the ‘data_analytics’ folder. All dependencies of Python libraries are loaded from ‘requriements.txt’ and installs them in the base package. It uses a basic python image from the environment variables in the hidden .env file. It copies all the Python files from ‘program’ folder to the containers execution environment and starts ‘app.py’ at container’s start.

The ‘app.py’ calls the main methods of ‘data_analytics.py’. The ‘data_analytics.py’ contains the scientific calculations and a MQTT client for receiving and sending data.

To give a short introduction to the ‘data_analytics.py’ code, the focus points will be described.

The file contains some important constants, which are used in the ‘DataAnalyzer’ class itself.

```
BROKER_ADDRESS='ie_databus'
BROKER_PORT=1883
MICRO_SERVICE_NAME = 'data-analytics'
USERNAME='edge'
PASSWORD='edge'
```

The first to constants describe the address of the MQTT Broker and the port. The username and password are the user and its password for authentication against the broker. This is important for the IE Databus on the IED later.

```
def handle_data(self):
        """
        Starts the connection to MQTT broker and subscribe to respective
        topics.

        """

        self.logger.info('Preparing Mqtt Connection')
        try:
            self.client.username_pw_set(USERNAME, PASSWORD)
            self.client.connect(BROKER_ADDRESS)
            self.client.loop_start()
            self.logger.info('Subscribe to topic StandardKpis')
            self.subscribe(topic='StandardKpis', callback=self.standard_kpis)
            self.logger.info('Subscripe to topic Mean')
            self.subscribe(topic='Mean', callback=self.power_mean)
            self.logger.info('Finished subscription to topics')


        except Exception as e:
            self.logger.error(str(e))
```

The ‘handle_data’ method is responsible to connect the MQTT client to the broker. At first set the credentials in the client and then try to establish the connection. As soon as it is possible to connect, the client subscribes to the two input topics with its corresponding callback methods. These two topics will be published with data from Node-RED.

The callback method ‘standard_kpis’ gets the power measurements of drive 1 and 2. It calculates the mean, median and standard deviation of each power measurement array. The result is packed in a json structure and published by the MQTT client on the result topic for Node-RED.

```
# Callback function for MQTT topic 'StandardKpis'
    def standard_kpis(self, payload):
        values = [key['value'] for key in payload]
        # Calculate standard KPIs
        result = {
            'mean_result' : statistics.mean(values),
            'median_result' : statistics.median(values),
            'stddev_result' : statistics.stdev(values),
            'name' : payload[0]['name'],
        }
        self.logger.info('mean calculated: {}'.format(statistics.mean(values)))
        self.logger.info('median calculated: {}'.format(statistics.median(values
            )))
        self.logger.info('stddev calculated: {} \n =======.format(statistics
            .stdev(values)))
        # publish results back on MQTT topic 'StandardKpiResult'
        self.client.publish(topic='StandardKpiResult', payload=json.dumps(result
            ))
        return
```

The callback method ‘power_mean’ is called when on the ‘Mean’ topic a message is received. On this topic a json object with two arrays is received with each voltage and current of drive 3. The total power is calculated over all array items of voltage and corresponding current. After this is done the mean value of the power of drive 3 is calculated. The result is packed in a json object and published by the MQTT client on the result topic back to the Node-RED.

```
#   Callback function for MQTT topic 'Mean' subscription
    def power_mean(self, payload):
        self.logger.info('calculating power mean...')

        current_values = [item['value'] for item in
            payload['current_drive3_batch']]
        voltage_values = [item['value'] for item in
            payload['voltage_drive3_batch']]
        # Calculate mean of power
        power_batch_sum = sum([current*voltage for current, voltage in zip
            (current_values,voltage_values)])

        power_mean = round((power_batch_sum/payload['sample_number']),2)
        self.logger.info("power mean result: {}\n".format(power_mean))

        result = {
            'power_mean_result' : power_mean,
            'name' : 'powerdrive3_mean',
        }
        # publish result back on MQTT topic 'MeanResult'
        self.client.publish(topic='MeanResult', payload=json.dumps(result))
        return
```

This was a brief overview of the focus points of the Python app which is contained in the ‘my_edge_app’.

To build the containers the docker-compose.yml is used.

```yaml
### Docker Compose File for my Industrial Edge App ###
# This docker-compose file creates a preconfigured
# * Data Analytics container based in Python with Mqtt Connection
# * InfluxDB Container for Storage of Time Series data
# * Grafana Container for visualization of database content

version: '2.4' # docker-compose version is set to 2.4

services:

###### DATA-ANALYTICS ######
  data-analytics:
    build:                              # Configuration applied at build time
      context: ./data-analytics         # Relative Path to data-analytics from this docker-compose file containing Dockerfile
      args:                             # Args variables available only at build-time
        BASE_IMAGE: $BASE_IMAGE         #
        http_proxy: $http_proxy         # Proxy url's from environment
        https_proxy: $https_proxy
    image: data-analytics:v0.0.1        # Name of the built image
    container_name: data-analytics      # Name of the data-analytics container
    mem_limit: 350m
    restart: unless-stopped             # always restarts (see overview page 12 Industrial Edge Developer Guide)
    environment:                        # Environment variables available at container run-time
      http_proxy: $http_proxy           # Proxy url's from environment
      https_proxy: $https_proxy
    logging:                            # allow logging
      options:                          # we use best pactice here as limiting file size and rolling mechanism
        max-size: "10m"                 # File size is 10MB
        max-file: "2"                   # only 2 files created before rolling mechanism applies
    networks:                           # define networks connected to container 'data-analytics'
      proxy-redirect:                   # Name of the network
    depends_on:                         # Dependencie on other container
      - grafana                         # Wait for start of container 'grafana'

##### INFLUXDB ######
  influxdb:
    image: influxdb:$INFLUXDB_VERSION   # Define image to pull from docker hub if not already on your machine available
    container_name: influxdb            # Name of the influx-db container
    restart: unless-stopped             # always restarts (see overview page 12 Industrial Edge Developer Guide)
    mem_limit: 700m
    environment:                        # Environment variables available at container run-time
      INFLUXDB_DB: $INFLUXDB_DB         # Variable of INFLUXDB_DB will be set at runtime as well
    logging:                            # allow logging
      options:                          # we use best pactice here as limiting file size and rolling mechanism
        max-size: "10m"                 # File size is 10MB
        max-file: "2"                   # only 2 files created before rolling mechanism applies
    volumes:                            # mount volume from host
      - db-backup:/var/lib/influxdb     # mount named volume 'db-backup' to host's path to /var/lib/influxdb
    ports:                              # expose of ports and publish
      - "33086:8086"                    # map containers port 8086 to host's port 33086
    networks:                           # define networks connected to container 'influxdb'
      proxy-redirect:                   # Name of the network



##### GRAFANA #####
  grafana:
    build:                                      # Configuration applied at build time
      context: ./grafana/                       # Relative Path to grafana from this docker-compose file containing Dockerfile
      args:                                     # Args variables available only at build-time
        GRAFANA_VERSION: $GRAFANA_VERSION       # Variable which contains version of Grafana
        GF_INSTALL_PLUGINS: $GF_INSTALL_PLUGINS # Variable which contains additional plugins to load
        http_proxy: $http_proxy                 # Proxy url's from environment
        https_proxy: $https_proxy
    image: grafana:v0.0.9                       # Name of the built image to be used
    container_name: grafana                     # Name of grafana container
    restart: unless-stopped                     # always restarts (see overview page 12 Industrial Edge Developer Guide)
    mem_limit: 350m
    environment:                                # Environment variables available at container run-time
      http_proxy: $http_proxy                   # Proxy url's from environment   ###For IEAP "192.xxx.xxx.xxx"
      https_proxy: $https_proxy
    logging:                                    # allow logging
      options:                                  # we use best pactice here as limiting file size and rolling mechanism
        max-size: "10m"                         # File size is 10MB
        max-file: "2"                           # only 2 files created before rolling mechanism applies
    ports:                                      # expose of ports and publish
      - "3000:3000"                             # map containers port 3000 to host's port 3000
    networks:                                   # define networks connected to container 'grafana'
      proxy-redirect:                           # Name of the network


###### NETWORK CONFIG ######
networks:                           # Network interface configuration
  proxy-redirect:                   # Reference 'proxy-redirect' as predefined network
    external:
      name: proxy-redirect
    driver: bridge

###### VOLUMES ######
volumes:                            # Volumes for containers
  db-backup:
```

The docker-compose.yml contains three service descriptions for ‘data-analytics’, ‘influxdb’ and ‘grafana’.

- influxdb:

  The service description will pull a InfluxDB image from DockerHub with corresponding version. A volume is mounted to this container to backup the database. The port for the interaction with the database is exposed to 8086. It uses the bridged network ‘proxy-redirect’.

- grafana:

  The service description contains a build step for the Dockerfile to build the image. The port 3000 for calling Grafana website is exposed. It uses the bridged network ‘proxy-redirect’.

- data-analytics:

  The service description contains a build step for the Dockerfile to build the image. It uses the bridged network ‘proxy-redirect’. It depends on the grafana container. This means data-analytics will start when the grafana container is started.

Important notice: Please be aware about the variables for the HTTP and HTTPS proxy. You must set them in the .env file. The default value is empty. This means no proxy is use. Please configure your proxy if you have one used as e.g., in most company network environments.

5.4 Test of app in development environment

To test your first app in the development environment the three above-described modules must be started in the correct order.

1. Start a terminal in the ‘mqtt_broker_mosquitto’ folder.
2. Create ‘proxy-redirect’ network by command `docker network create --driver bridge proxy-redirect`
3. Execute command ‘`docker-compose up -d`’ in the terminal of 1.
4. Start another terminal in the ‘my_edge_app’ folder
5. Execute command ‘`docker-compose up -d`’ in the terminal of 4.
6. Start another terminal in the ‘node_red’ folder
7. Execute command ‘`docker-compose up -d`’ in the terminal of 6.
8. Execute command ‘`docker ps`’ to see if all containers are running. The console output should look like in Figure 5-8

```
leadmin@ubuntu:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                              NAMES
b0241d5c3533        nodered:v0.0.1                 "npm start -- --user…"   34 seconds ago      Up 33 seconds       0.0.0.0:33080->1880/tcp            nodered
67b7c0c892e8        data-analytics:v0.0.1          "python -u -m app"       52 seconds ago      Up 51 seconds                                          data-analyt
ics
de04df0373d5        grafana:v0.0.9                 "/run.sh"                53 seconds ago      Up 51 seconds       0.0.0.0:3000->3000/tcp             grafana
dbc49250de6e        influxdb:1.7-alpine            "/entrypoint.sh infl…"   53 seconds ago      Up 51 seconds       0.0.0.0:33086->8086/tcp            influxdb
3f485d0b8ebb        eclipse-mosquitto:1.6.14       "/docker-entrypoint.…"   18 hours ago        Up About a minute   0.0.0.0:33083->1883/tcp            le_databus
```

Figure 5-8: Docker PS command after all docker-compose up of your app

As all containers of your first app are running on your development environment, you can open a web browser on your development machine.

Now we open two tabs in the web browser:

1. http://localhost:3000 for Grafana
2. http://localhost:33080 for Node-RED

You should see the login page of Grafana and the Node-RED flow creator page with prebuilt flow to test your Industrial Edge app. Both websites are shown in Figure 5-9.

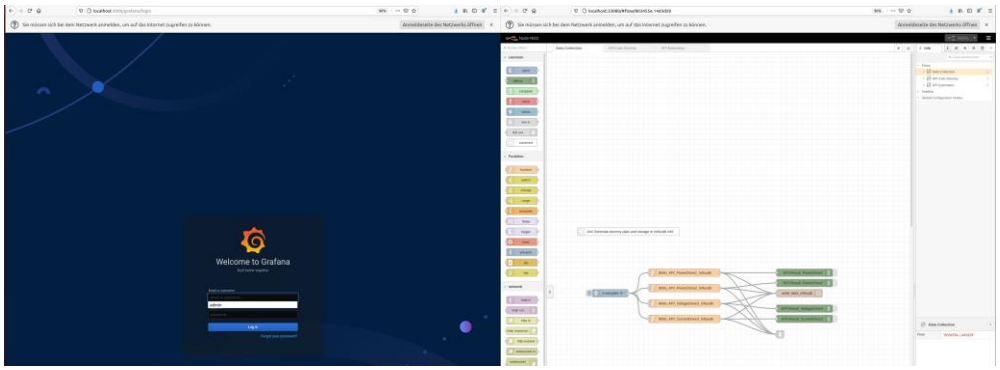

Figure 5-9: Websites of Grafana and Node-RED container

At first you can login with username ‘admin’ and password ‘admin’ into Grafana.

You can open the demo dashboard for your Industrial Edge app by clicking on the ‘Home’ in upper left corner. You should get an overview of the Dashboards and click on the ‘Beta-KPI-Calc-Dashboard’. You should see the dashboard view like Figure 5-10. **Be aware** that you might not see any data, because you did not generate them via Node-RED flow. Grafana is querying the data from the InfluxDB database.

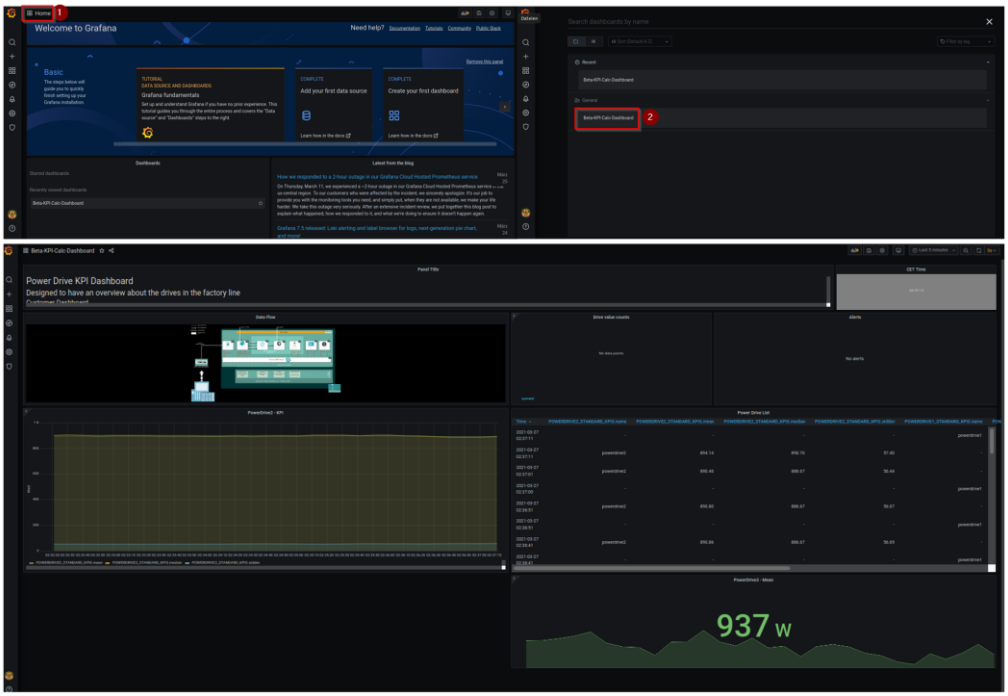

Figure 5-10: Navigation in Grafana to open Dashboard of your app

To see visualization of data from your ‘data-analytics’ app, you have to trigger the data generation in the Node-RED flow.

But at first you will get an introduction into the Node-RED flow. The current use case is to get two power measurements from two drives as well as a current and voltage of another drive. The raw data is stored in the InfluxDB. A collection of measured data is queried from the database to prepare an array of it to be sent to the ‘data-analytics’ container. The calculated data from ‘data-analytics’ is received and stored in the InfluxDB as well.

Therefore, the existing flow is divided into three parts:

- Data Collection: This flow contains a trigger which will fire events every 10 seconds. In the function nodes a random data for the power of the drives

as well as the current and voltage is created. It will be written to the InfluxDB database. All created values are redirected to flow ‘KPI Estimation’.

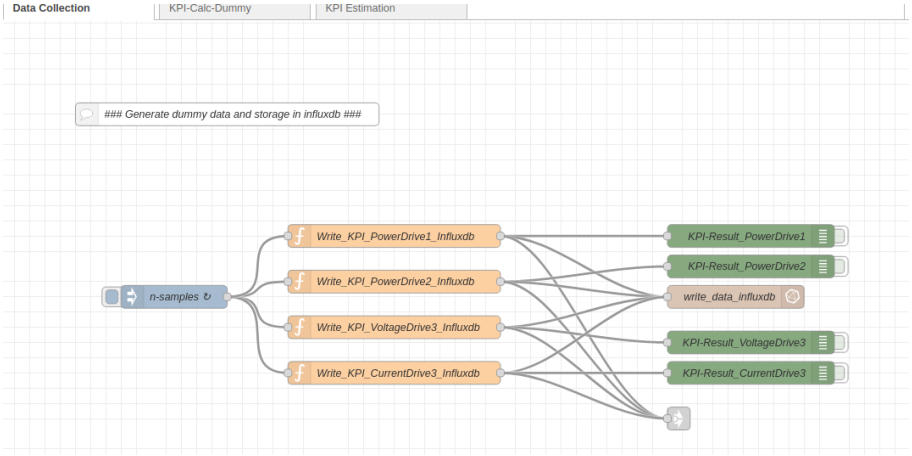

Figure 5-11: Flow to generate data by cyclic trigger

- KPI-Calc-Dummy: This flow queries the stored power values of the two drives and calculates the total power of them. It is stored in InfluxDB as well.

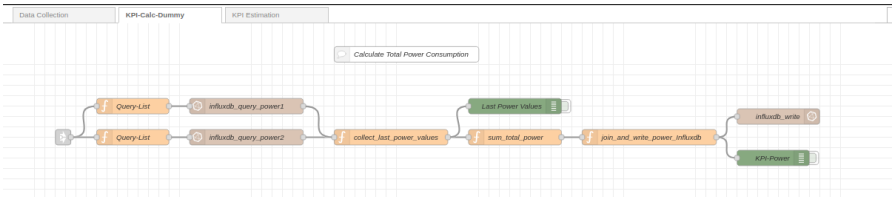

Figure 5-12: Flow to execute some data processing in Node-RED to calculate total power

- KPI-Estimation: The surrounded flow in the upper are of Figure 5-13 filters the incoming JSON messages from flow ‘Data Collection’. The upper flow will query the power of drive 1 and 2 and collect the last 50 entries from the database and publish them to MQTT-topic ‘StandardKpis’. The other flow is querying the last 50 voltage and current values of drive 3 from the database. Both are joined together as a single message to be published to the MQTT topic ‘Mean’. The flow outside of the red rectangle is listening on the MQTT topics ‘StandardKpiResult’ and ‘MeanResult’. The result messages are preprocessed to store the data-analytics results in the InfluxDB as well.

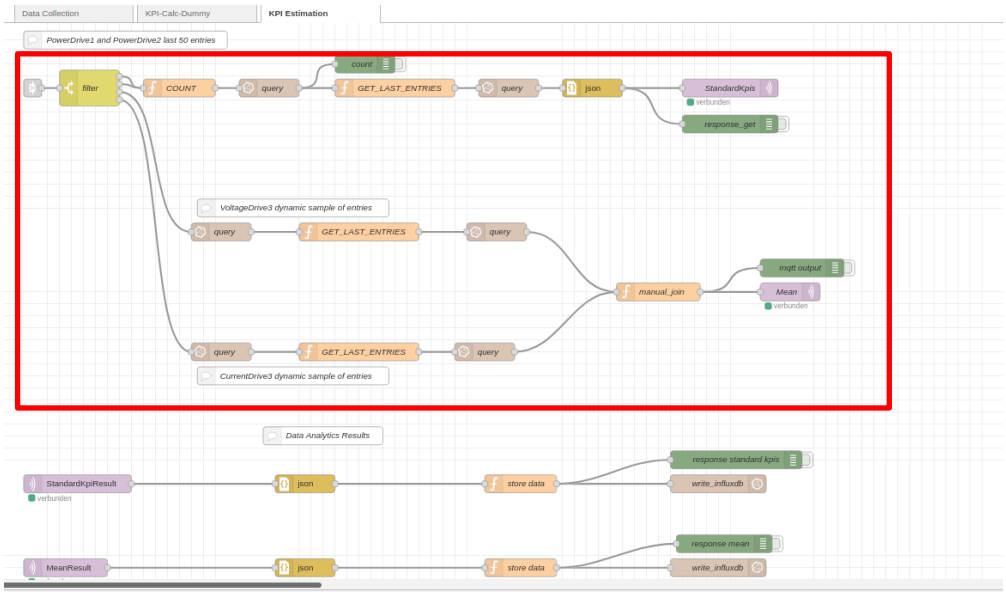

Figure 5-13: Flow to prepare data for data analytics and consume results

If the flow is not deployed yet or any changes are made, please click on the red deploy button in the upper right of the Node-RED website. Please click the trigger in flow ‘Data Collection’ to start data generation.

You can switch to Grafana again and you should see a visualization of data as soon as enough data is available and processed from data-analytics. Your Grafana dashboard should look like Figure 5-14.

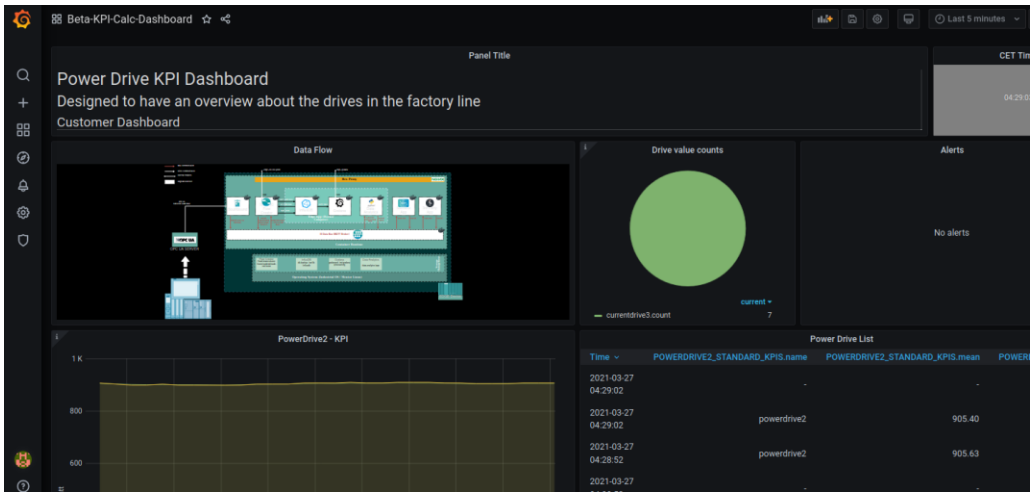

Figure 5-14: Dashboard with stored data from data-analytics loaded from InfluxDB

Congratulation your first Industrial Edge App is working and you can continue the next preparation steps to bring it to your Industrial Edge Device.

5.5 Upload the app with the Industrial Edge App Publisher (IEAP) to the Industrial Edge Management (IEM)

After the successful testing of your first Industrial Edge app on your development environment, the docker-compose.yml file must be adapted for the Industrial Edge. In the folder ‘my_edge_app’ is another docker-compse_Edge.yml file which is also adapted for the Industrial Edge.

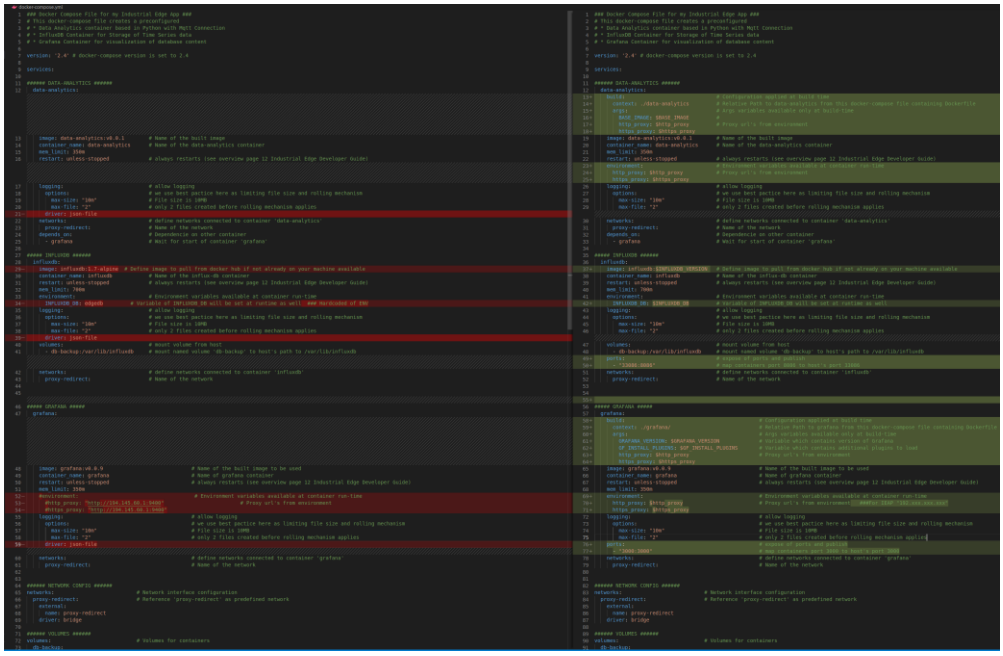

Figure 5-15: Diff of docker-compose of your app

The most important changes are:

- Build steps are completely removed as IEAP will take the created images from the docker-compose file.
- All environment variables are set hardcoded, which are set by the .env file.
- All port exposes are removed as we are using the reverse proxy of the IED.
- The environment variables are also removed as there is no need for a proxy server in your first app.
- For logging the default driver ‘json-file’ is added.

Important notice: Please be aware that the images of your first app are all built.

Open the IEAP app on your development environment. If you are not connected to your IEM, please ‘Go online’ again and login to the IEM in the IEAP app.

Please verify that you have already created a project for your first Industrial Edge app on the IEM. If not just open the IEM and login. Navigate to ‘Apps’ and click on ‘My Projects’. You can create a new project by clicking on ‘Create Project’ in the upper right corner of the page as shown in Figure 5-16. Fill out the form of Project and click on ‘Create’. You have now created a new project. You can now switch back to the IEAP app.

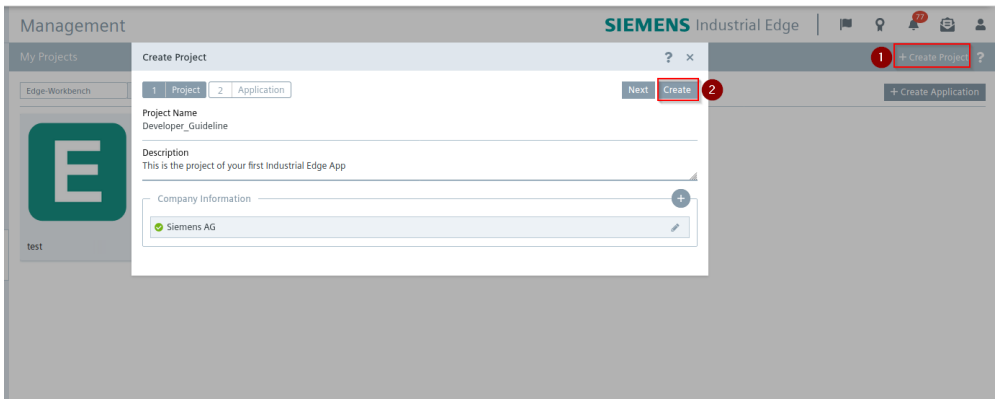

Figure 5-16: Create Dialog of a new project on IEM

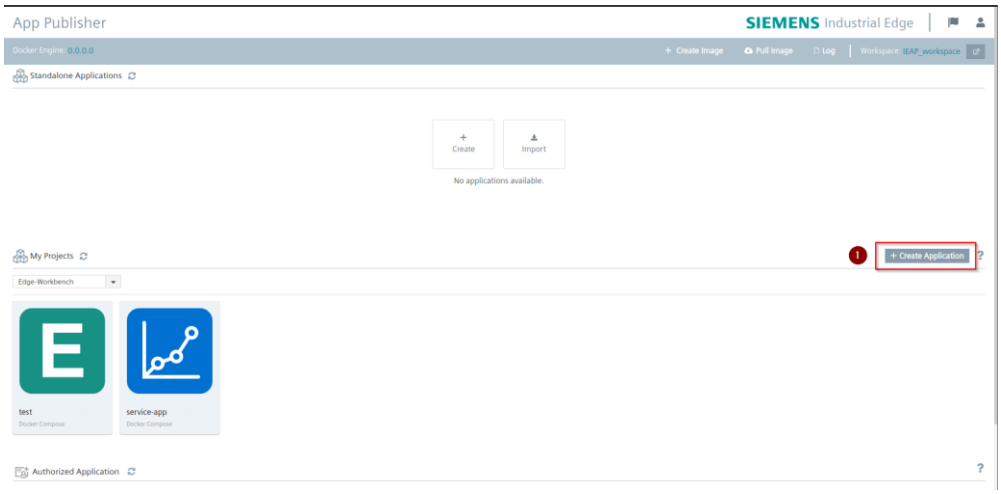

Figure 5-17: IEAP logged in to IEM with selected project

In Figure 5-17 you can click on the refresh button of the ‘My Projects’ section and select your project from the drop-down item.

Click on ‘Create App’ and should be redirected to the IEM where you again click on ‘Create App’.

A new formular is opened. Please fill out all fields which are marked in the Figure 5-18.

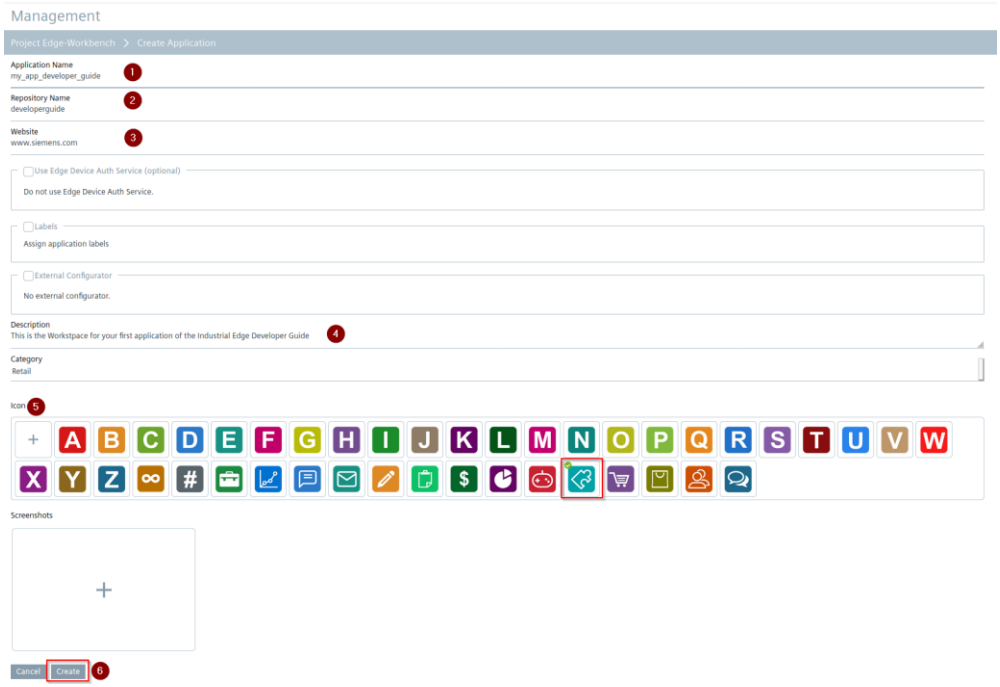

Figure 5-18: Create a new empty Industrial Edge app on IEM

Click on ‘Create’ to create an empty Industrial Edge app on your project workspace. In the IEM you see your currently created app with your icon.

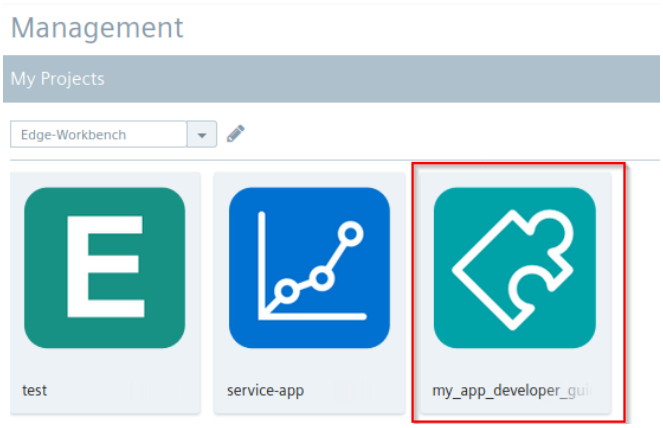

Figure 5-19: Created empty app on IEM

Switch back to the IEAP to refresh the ‘My Projects’ section. Your app will appear as well. Click on the icon of your app.

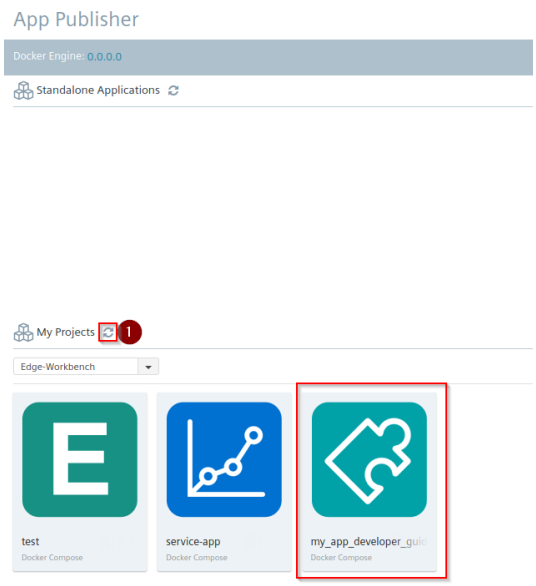

Figure 5-20: Created App available on IEAP

You click on ‘Add New Version’. A new dialog to select your docker compose version. As the docker-compose version of 2.4 is used in the file, you must select ‘2.4’.

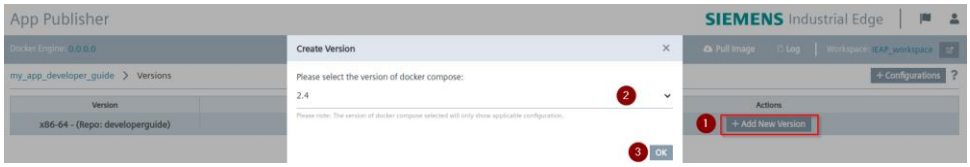

Figure 5-21: Create a new version of app and select docker-compose version

Now you have two options to create a docker-compose file for the Industrial Edge of your app:

- Graphical supported by dialogues to create a corresponding docker-compose.yml file
- Import of an existing one and adapt it afterwards

As in your ‘my_edge_app’ folder is already an docker-compose_Edge.yml file, you click on ‘Import YAML’.

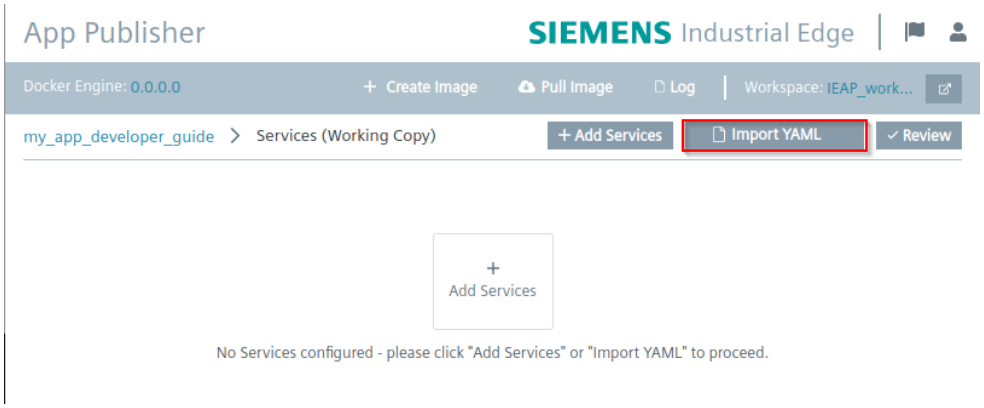

Figure 5-22: Dialog to add a docker compose for your app

Select the docker-compose_Edge.yml file from the file system.

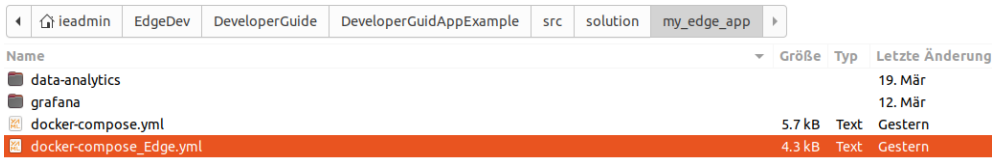

Figure 5-23: Select the Industrial Edge docker-compose.yml file for your app

The docker-compose_Edge.yml is loaded and divided into the service sections. With the pen icon you can edit the service. As we are using the reverse proxy of the Industrial Edge Device. You must configure this reverse proxy for the surface of Grafana.

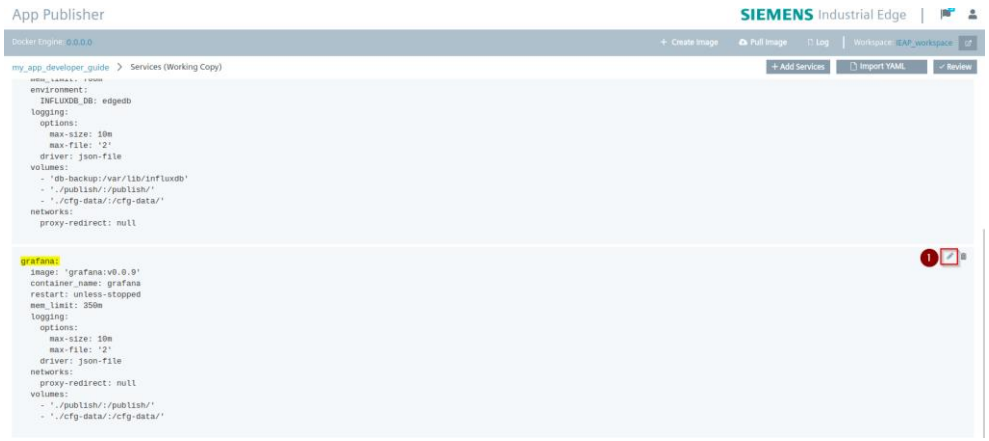

Figure 5-24: Edit Grafana service in the docker-compose definition

Open the network service as shown in Figure 5-25.

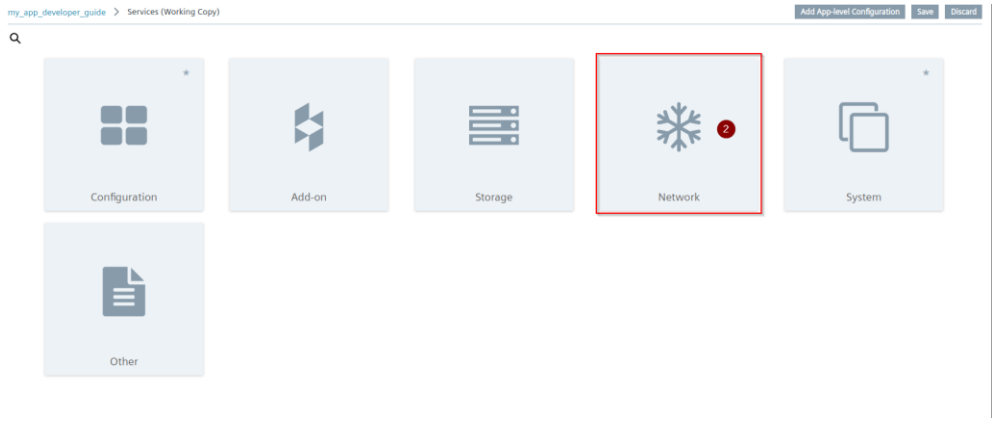

Figure 5-25: Network section of Grafana

Go to the section ‘Reverse Proxy’ and add the same inputs as in Figure 5-26.

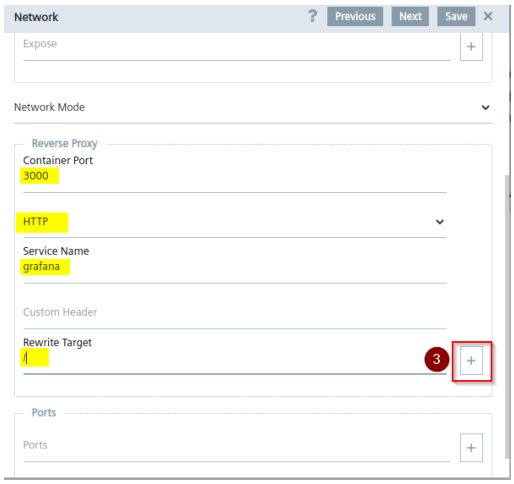

Figure 5-26: Settings of the reverse proxy server for Grafana

Click on the + icon next to the ‘Rewrite Target’ field and Grafana is added to the reverse proxy and click on ‘Save’. Click ‘Save’ again in the service page.

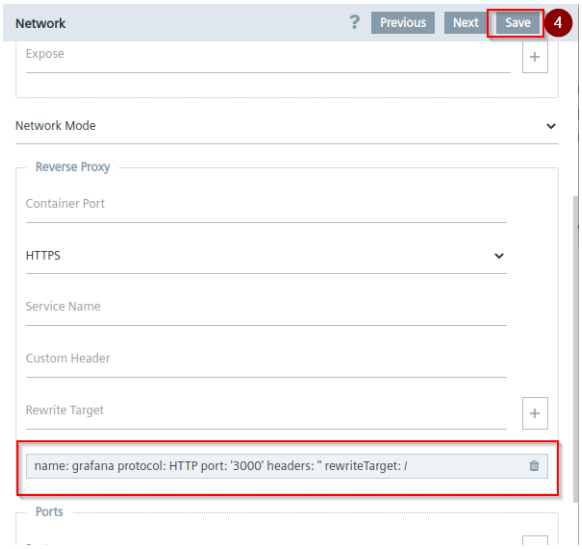

Figure 5-27: Save Grafana reverse proxy server settings

Click on ‘Review’ and then ‘Validate & Create’.

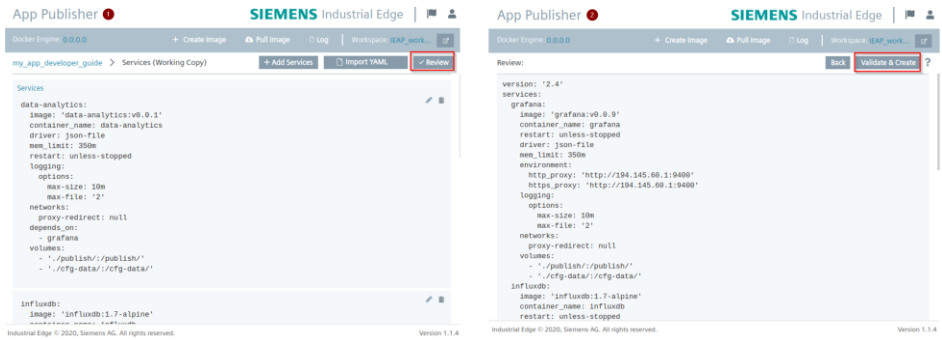

Figure 5-28: Review and Validation of docker-compose file for Industrial Edge app

A window with ‘Add version’ appears. As it is the first version of your app, you do not modify anything and just click on ‘Create’.

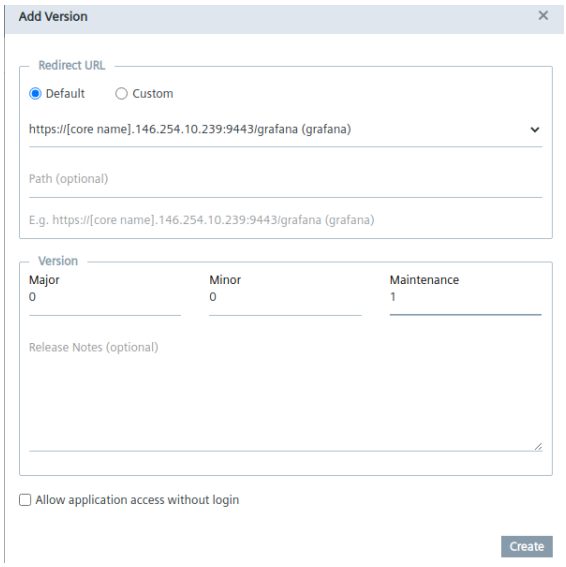

Figure 5-29: Add version of your Industrial Edge app

If everything in the docker-compose is correct and the built images exist are available, your first version of your Industrial Edge app is available. This might take a second.

The result should look like in Figure 5-30.

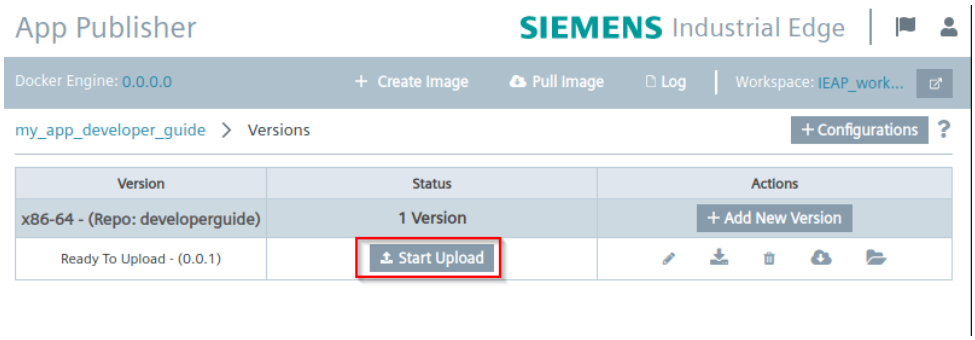

Figure 5-30: Upload of app to IEM

You can click on ‘Start Upload’ to transfer your version of your app to the IEM. This might take a while. The status can be observed when you are clicking on the flag icon next to your account in the upper right corner.

If the file transfer to the IEM is successful, you can open the web view of the IEM and navigate to your app icon in your project. Click on the icon and a first version of your app should be available as shown in Figure 5-31.

5.6 Deploy of your first app on an Industrial Edge Device

Important hint: It is necessary to have an onboarded IED in your IEM. If you do not have onboarded an IED right now, you must do this first. For more information of how to onboard your IED, please have a look into the official documentation.

You click on the download icon of your version and a new window to install the app is opened. Select the IED of the list, where the app should run on. Click on the ‘Install now’ button to execute the deployment to the IED immediately. This will take a while.

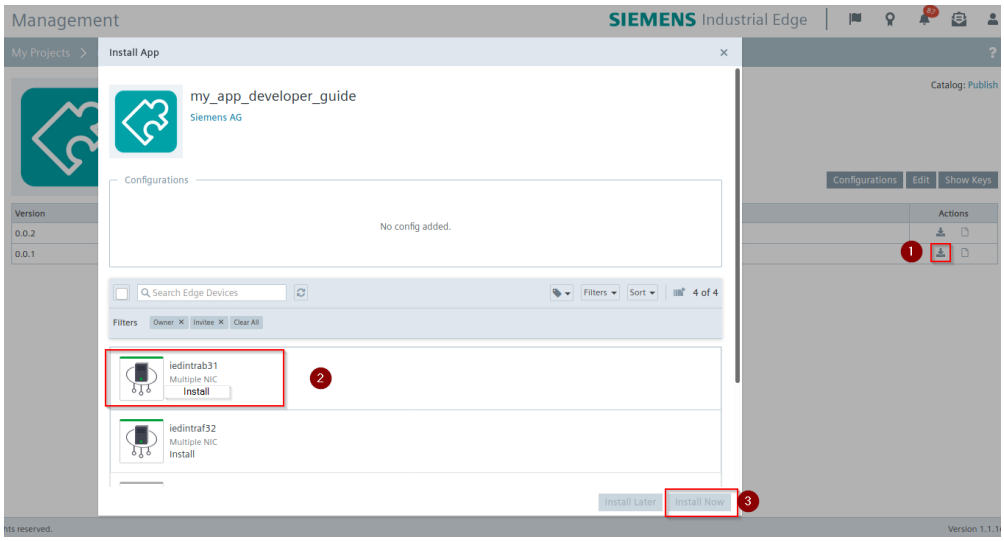

Figure 5-31: Deploy of your app to your IED

In the meanwhile, you can connect to the IED, where the app is being installed. This can be done by two ways:

- Enable the remote access and click after this is applied on the icon of the IED to open the IED website.
- Find the IP address in the statistics and open https://xxx.xxx.xxx.xxx/ with the IP address inserted. This way is preferred.

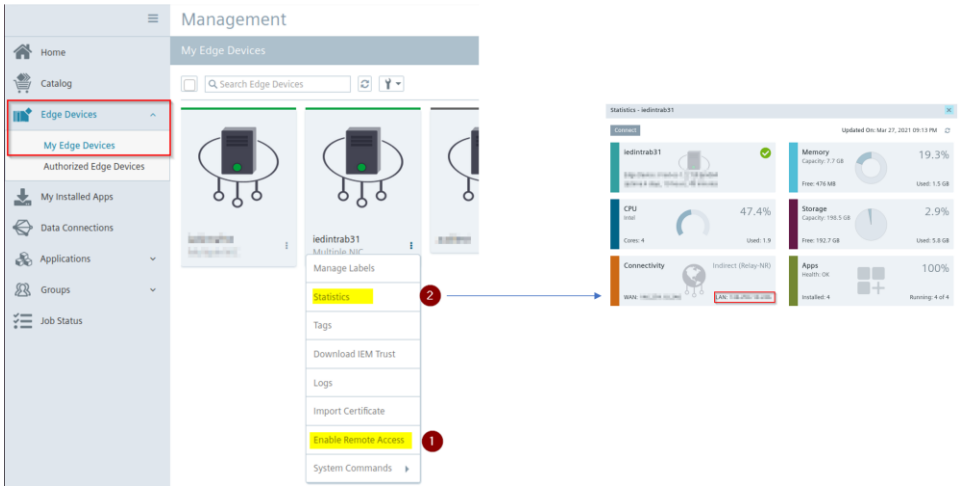

Figure 5-32: Two methods to get access to IED

Navigate to the Apps tab in your IED. If you do not see your app icon in the view, click on the flag icon in the upper right corner and see the status of the app deployment. When the app deployment of your first app is successfully done, you will see the icon as shown in Figure 5-33.

Figure 5-33: App successfully deployed on IED

By clicking on your app icon, a new tab in your web browser is opened. The reverse proxy is opening the Grafana login page, which you previously configured.

You successfully deployed your first app to an IED.

As we do not have the system app IE Databus running, the ‘data-analytics’ container is not working as it is not connected yet.

For further status information about your app or getting logs of your containers, you can enable the Developer Mode in the settings of the IED.

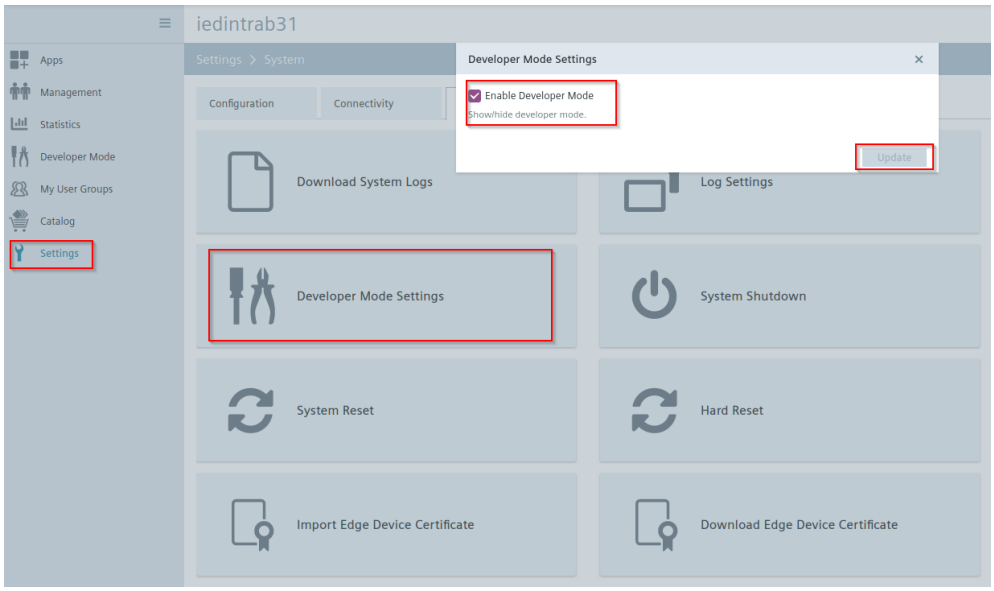

Figure 5-34: Enable Developer Mode on IED

This might be useful in cases the app is not working to get detailed information about the error.

5.7 Interaction with the System Apps

To test your first app as you did in 5.4, you have to deploy two system apps of the Industrial Edge.

Go back to the IEM and open the catalog. Install the IE Databus and IE Flow Creator app to the IED with your app.

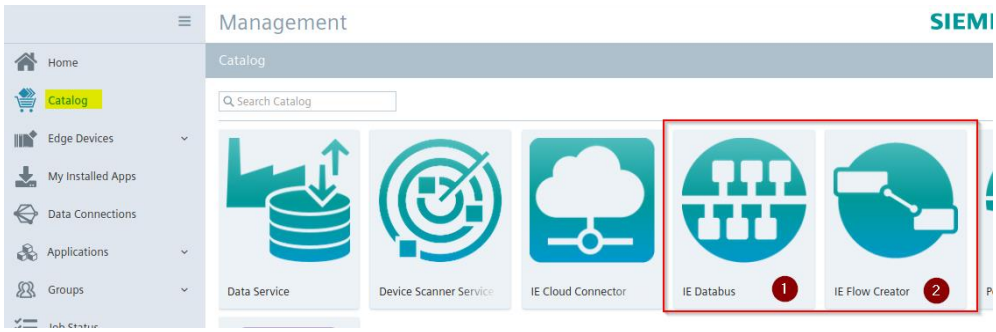

Figure 5-35: Available System Apps in the catalog of IEM

You click on the app icons and then on ‘Install’. Select your IED and click on ‘Install Now’. This will take a minute.

You see the app in the IED ‘Apps’ tab after successfully deployed.

You must do some configuration on the IE Databus to make it executable on the IED with your app. The IE Databus does not allow anonymous publishing or subscripting of a MQTT client on any topic. Therefore, you must configure the IE Databus with your Industrial Edge app topics.

Go in the IEM to the ‘Data Connections’. Click on the IE Databus App.

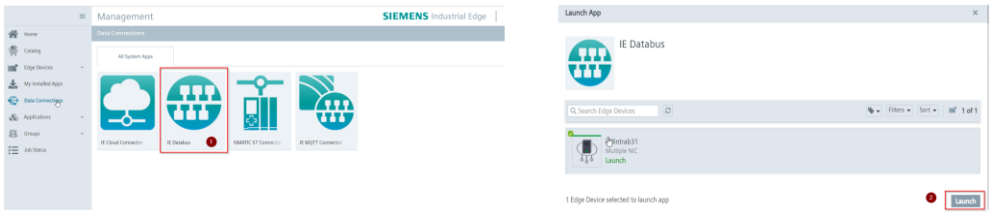

Figure 5-36: Open configuration of IE Databus

Open the configurator of the IE Databus of your IED as shown in Figure 5-36 and Figure 5-37.

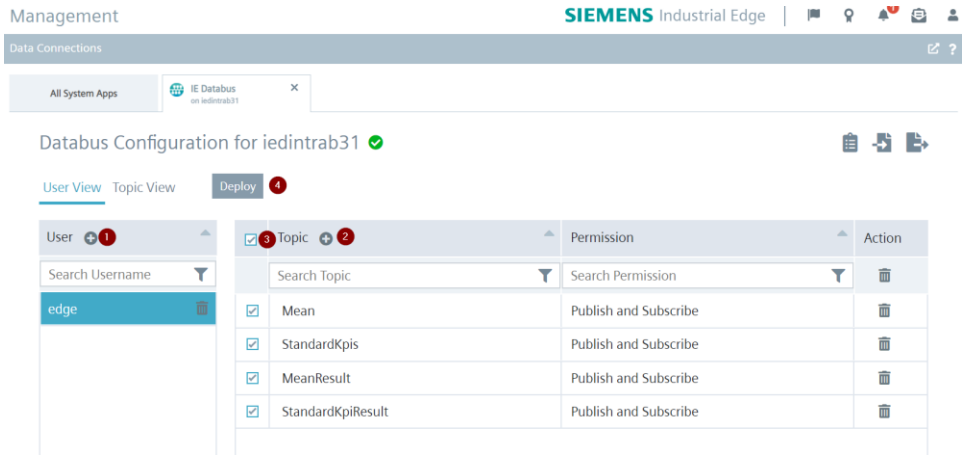

Figure 5-37: IE Databus configuration to work with your app

1. Create a user called ‘edge’ with password ‘edge’ by clicking on the + icon and the first topic
2. Add the other three topics of your app to this user
3. Select all topics
4. Deploy the configuration to the IE Databus on the IED

Important hint: Do not directly close the window until the process of applying the configuration is finished.

You must go back to IED and open ‘Management’ to restart your app to successfully connect your data_analytics container to the IE Databus. You click on ‘Restart’ as shown in Figure 5-38.

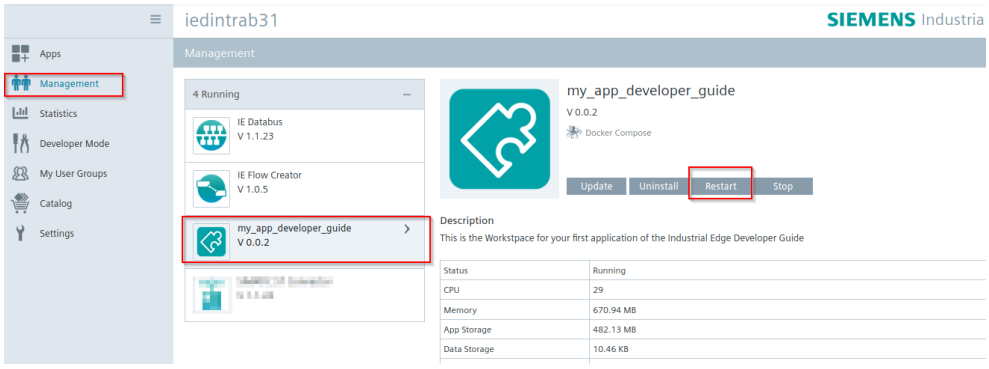

Figure 5-38: Restart of your app to connect properly to IE Databus

After you applied the configuration successfully to the IE Databus and your first app is restarted successfully, you open the IE Flow Creator app by clicking on the icon. A new tab in your web browser opens with a login window. Please login with your IED login data as an admin.

At first you must install the InfluxDB Plugin on the IE Flow Creator. Open the ‘Manage palette’ as shown in Figure 5-39. Click on tab ‘Install’ and type ‘influx’ and select the first result and click on ‘Install’.

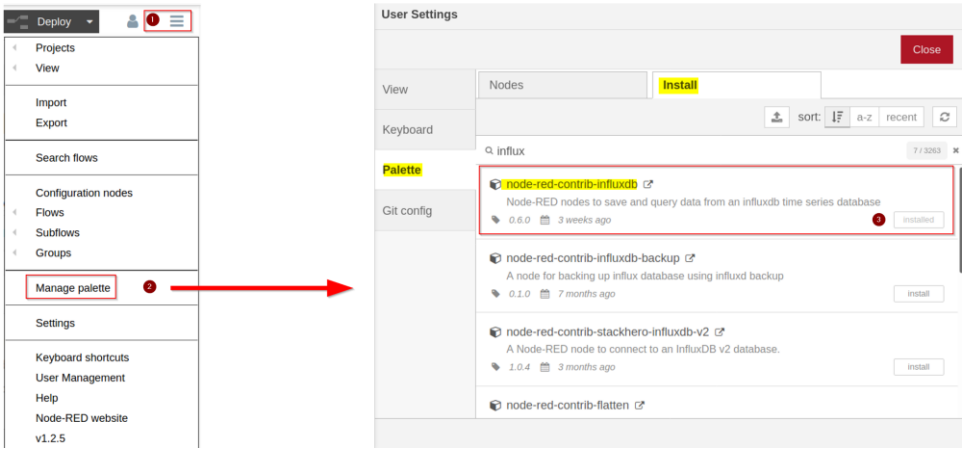

Figure 5-39: Install InfluxDB Plugin in IE Flow Creator

You can import an adapted flow as you have already tested with Node-RED in 5.4.

Therefore click on ‘Import’ in the menu and ‘select a file to import’ as your first app is already shipped with a prebuild data generation flow called ‘flow_data_gen.json’ in the folder ‘IE Flow Creator Flows’.

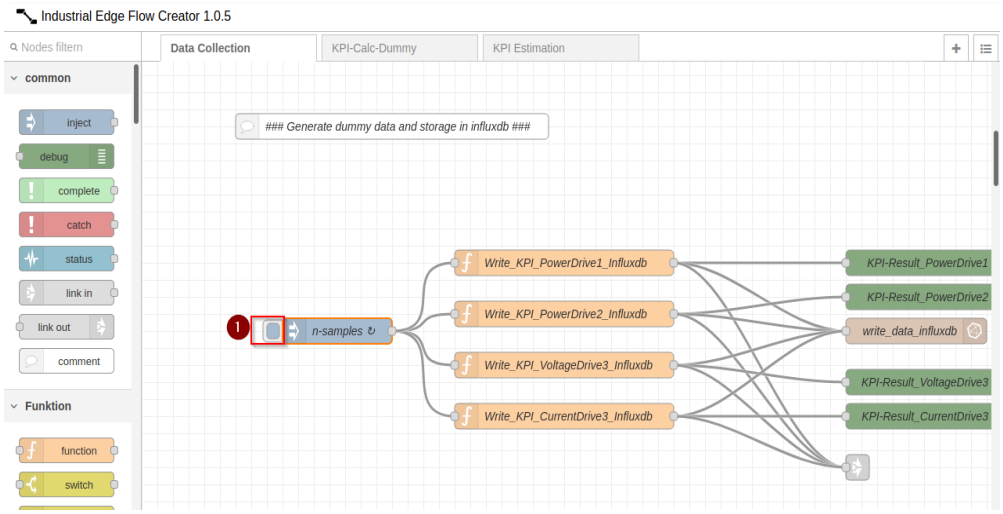

Figure 5-40: Start data generation in imported flow in the IE Flow Creator

You start the data generation by clicking on the node as shown in Figure 5-40.

Open the Grafana web view by clicking on your apps icon on the IED website.

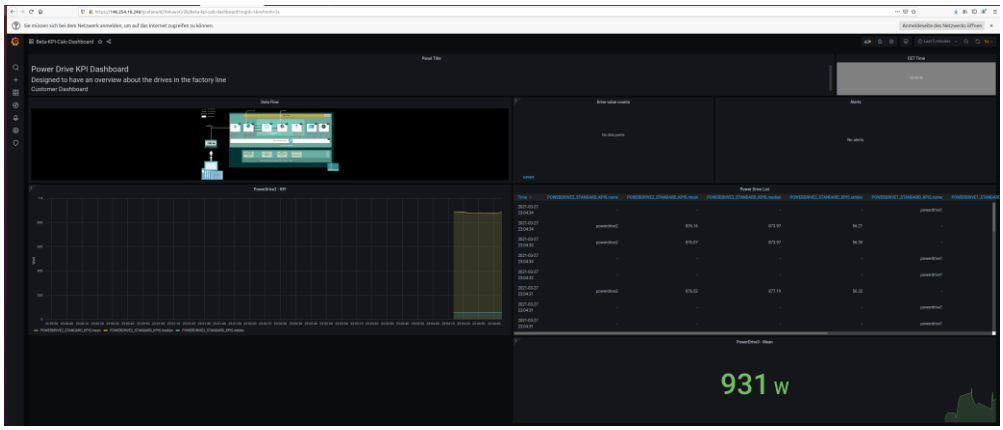

Figure 5-41: Grafana Dashboard with data from your first running app on the IED

The view should contain data as shown in Figure 5-41.

You have successfully verified your first app with generated data on an IED.

The next step is to connect the IE Flow Creator to an OPC UA Server to collect real-world data, if you have one available. If not you will not be able to get displayed data by using SIMATIC S7 Connector app.

Important hint: You will not be able to connect to the OPC UA Server, which is in this tutorial. For this case you need an own OPC UA Server as a real PLC or Simulation (SIMATIC S7-PLCSIM Advanced). A demo TIA Portal project is available here, which can be used to simulate a OPC UA server. The OPC UA server should be in the same network available and accessible.

At first another system app called SIMATIC S7 Connector needs to be installed to the IED, where your app is running.

Figure 5-42: System app SIMATIC S7 Connector in IEM Catalog

Select your IED after clicking on ‘Install’ as shown in Figure 5-42.

Go to menu item ‘Data Connections’ after it is installed. Click on ‘Update Configuration’ and ‘Launch’ as shown in Figure 5-43.

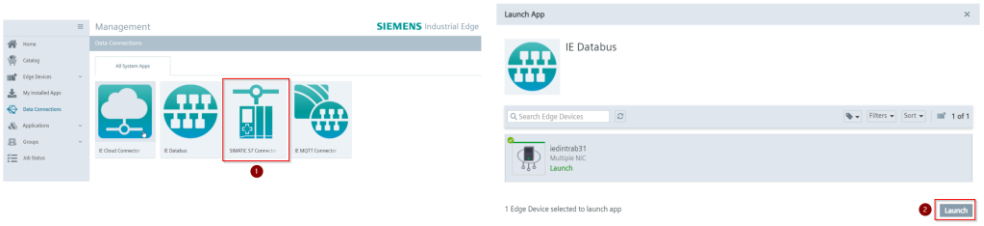

Figure 5-43: Open configuration of SIMATIC S7 Connector

A new window of the SIMATIC S7 Connector opens to add data sources by clicking on ‘Add Data Source’ as shown in Figure 5-44.

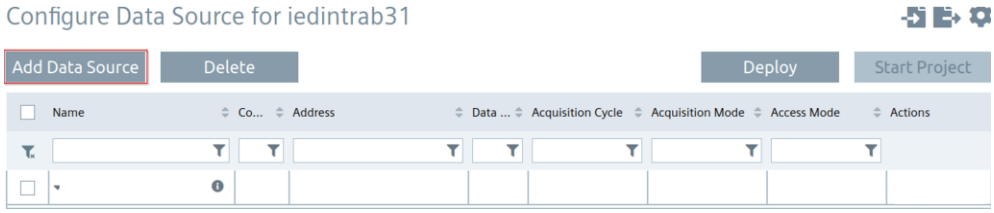

Figure 5-44: Add Data Source to SIMATIC S7 Connector configuration

Select ‘Data Source Type’ as OPC UA and provide the required information like a name, URL, port and security levels. Our test server supports security mode ‘None’ and ‘Anonymous’. This configuration depends on your endpoints of your OPC UA Server.

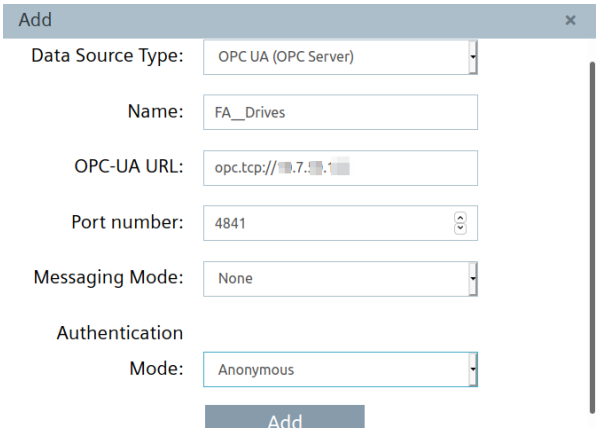

Figure 5-45: OPC UA Data Source configuration

Click on ‘Add’ create the data source. By clicking on the +-icon of the ‘FA_Drives’ as shown in Figure 5-46, you can add nodes of your OPC UA server.

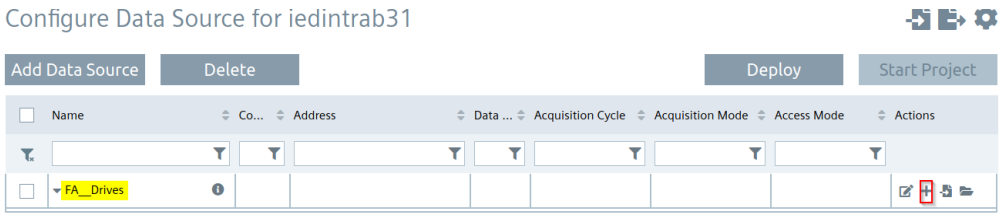

Figure 5-46: Add OPC UA node to Data Source 'FA_Drives'

Enter a name for node, can be any name, in this case we take ‘powerdrive1’ as shown in Figure 5-47. The addresses can be taken from Figure 5-48. It contains from namespace index (ns) and the node id (s). Then the data type is selected as Real as we are reading a float value. The value is checked every 2 seconds and will be read only when changed.

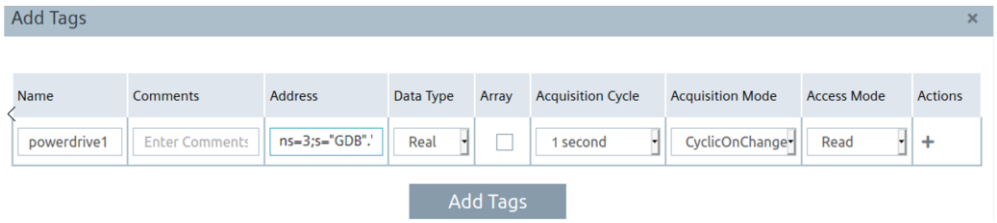

Figure 5-47: Add OPC UA node 'powerdrive1’

This will be done for the next three nodes ‘powerdrive2’, ‘voltagedrive3’ and ‘currentdrive3’. The result should look like the table in Figure 5-48.

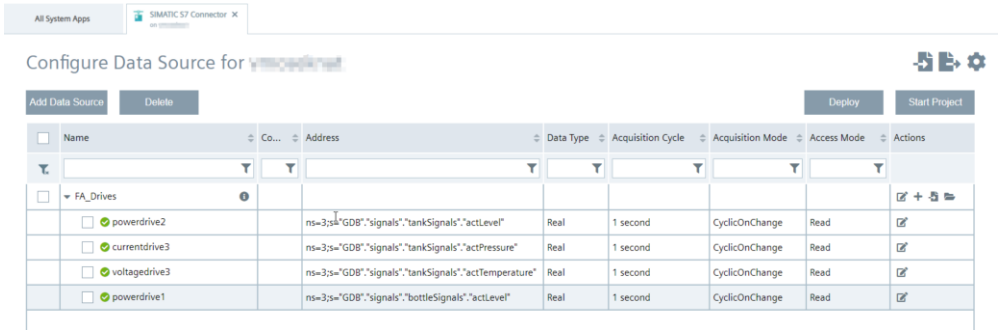

Figure 5-48: Configured nodes of OPC UA in SIMATIC S7 Connector

After this is done, you have to configure the databus, where the data is published. This is shown in Figure 5-49. Enter the name of the IE Databus. The user ‘edge’ and its password ‘edge’. Bulk Publish means, that all the data is collected of a cycle and sent as an array with all the values of the configured nodes.

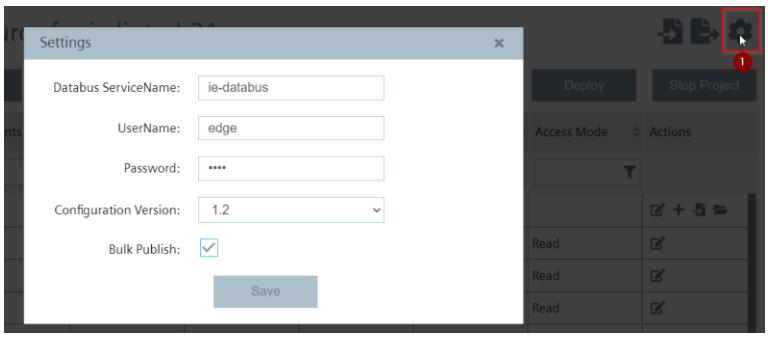

Figure 5-49: General settings of SIMATIC S7 Connector

After the configuration is saved. Select all nodes and click on ‘Deploy’. After the deployment was successful, you can click on ‘Start Project’.

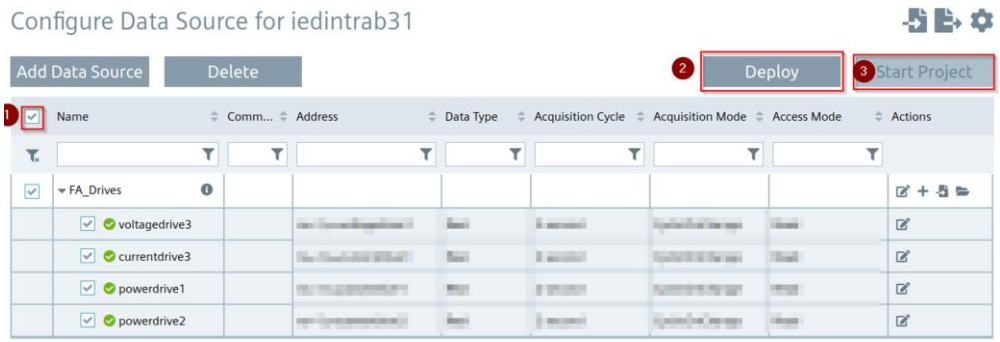

Figure 5-50: Deploy configuration of SIMATIC S7 Connector to IED

The SIMATIC S7 Connector is configured and completed.

To allow the SIMATIC S7 Connector to publish the results and to consume them in the IE Flow Creator, you must add an additional topic to the IE Databus app. Open the configuration of the IE Databus on the IEM for your IED.

You must add the following topic ‘ie/d/j/simatic/v1/s7c1/dp/r/FA_Drives/#’ and ‘ie/m/j/simatic/v1/s7c1/dp’ to your ‘edge’ user as shown in Figure 5-51. Select all topics and deploy them to the IED.

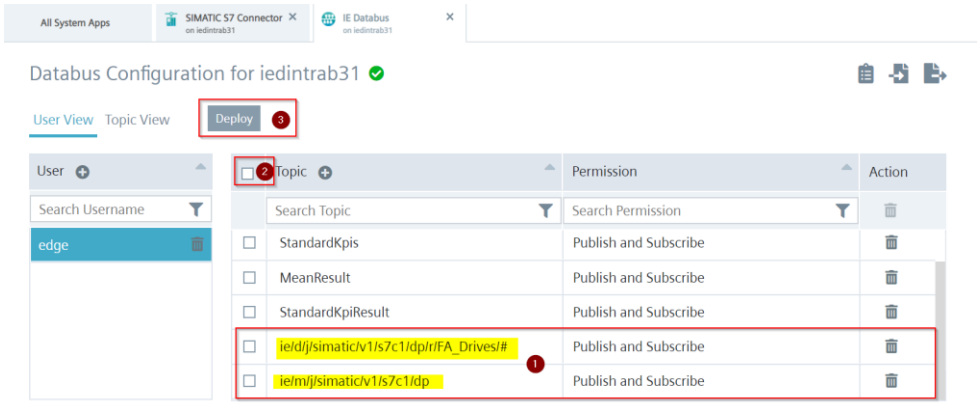

Figure 5-51: Add 'FA_Drives' SIMATIC S7 Connector to IE Databus

The topic is set together the following way:

- ie represents Industrial Edge
- d represents data
- j represents json
- s7c1 represents the SIMATIC S7 Connector Instance 1
- dp represents datapoint
- r represents read
- FA_Drives represents the data source name
- # represent all subtopics under FA_Drives

For further information about the topic structure of the IE Flow Creator, please have a look into the official documentation.

Now the SIMATIC S7 Connector can publish the data cyclic to the IE Databus on the configured subtopics of ‘FA_Drives’.

Important Hint: Please restart your SIMATIC S7 Connector.

Now you must consume the data, which is published by the SIMATIC S7 Connector on the IE Databus. Open the IE Flow Creator app on your IED. You can

import the predefined flow from folder ‘IE Flow Creator Flows’ the JSON file ‘flow_data_S7_Connector.json’. It contains instead of the generating flow a new flow which is shown in Figure 5-52.

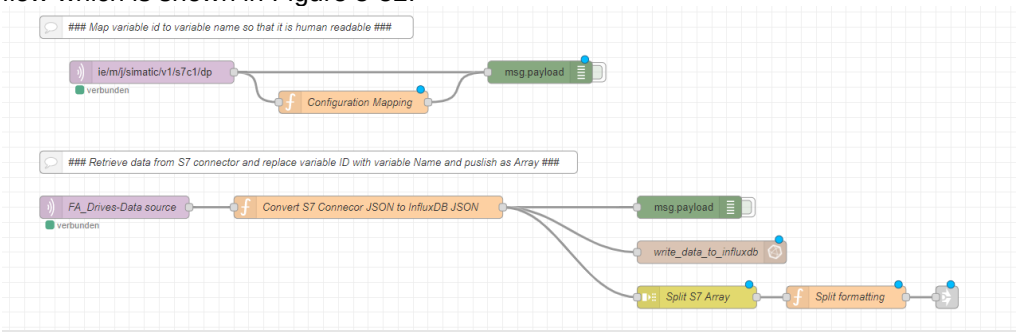

Figure 5-52: IE Flow Creator flow to consume data and preprocessing

The flow consumes on the default topic of the SIMATIC S7 Connector the corresponding json structure. The structure is preprocessed to store it in suitable way for the InfluxDB.

Deploy your flow in the IE Flow Creator and open the Grafana web view of your app on the IED. Wait till 50 values are received from the SIMATIC S7 Connector before data is visible in Grafana.

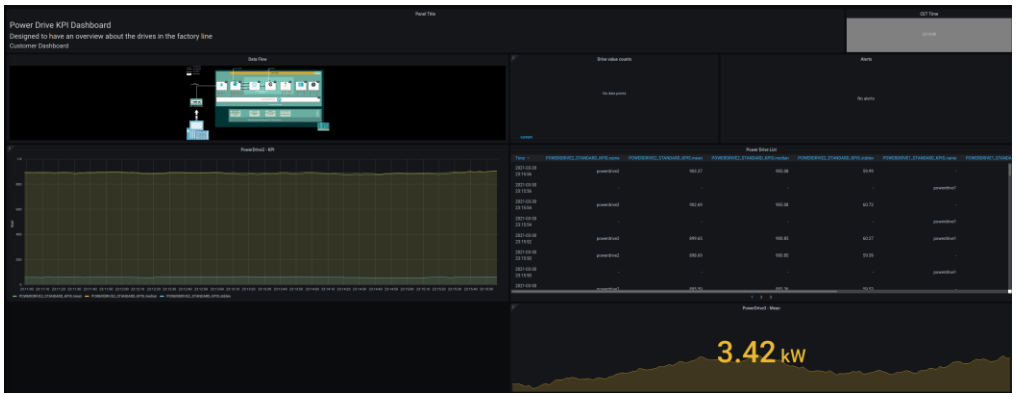

Figure 5-53: Grafana data visualization with data published by SIMATIC S7 Connector

Grafana should show the visualization of the real-world data of the drives. You successfully combined multiple system apps to consume real-world data, preprocess the data, store them, and do analytics on them. Finally, you also created a visualization of the data for users.

6 Appendix

6.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, app examples and videos – all information is accessible with just a few mouse clicks:
support.industry.siemens.com

SITRAIN – Training for Industry

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:
www.siemens.com/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:
support.industry.siemens.com/cs/sc

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for Apple iOS, Android and Windows Phone:
support.industry.siemens.com/cs/ww/en/sc/2067

6.2 Links and literature

Table 6-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to this entry page of this app example https://support.industry.siemens.com/cs/ww/en/view/109795865
\3\	

6.3 Change documentation

Table 6-2

Version	Date	Modifications
V1.0	04/2021	First version
V1.2.0	04/2021	Release version for Industrial Edge V1.2
V1.2.1	04/2021	Changes due to SIMATIC S7 Connector Release V1.2