AMORPH.pro
SMARTUNIFIER

SMARTUNIFIER Demo Guide

Release 1.4.0

Amorph Systems GmbH

Apr 25, 2022

TABLE OF CONTENTS


Integrate perfectly your
Production-IT using
SMARTUNIFIER
ADVANCED IT-INTEGRATION PLATFORM
AMORPH.pro
SMARTUNIFIER

CHAPTER

ONE

ABOUT SMARTUNIFIER DEMONSTRATOR

- Learn about the *Demonstrator*.
- Learn about the *SMARTUNIFIER* connectivity platform.

1.1 What is SMARTUNIFIER Demonstrator

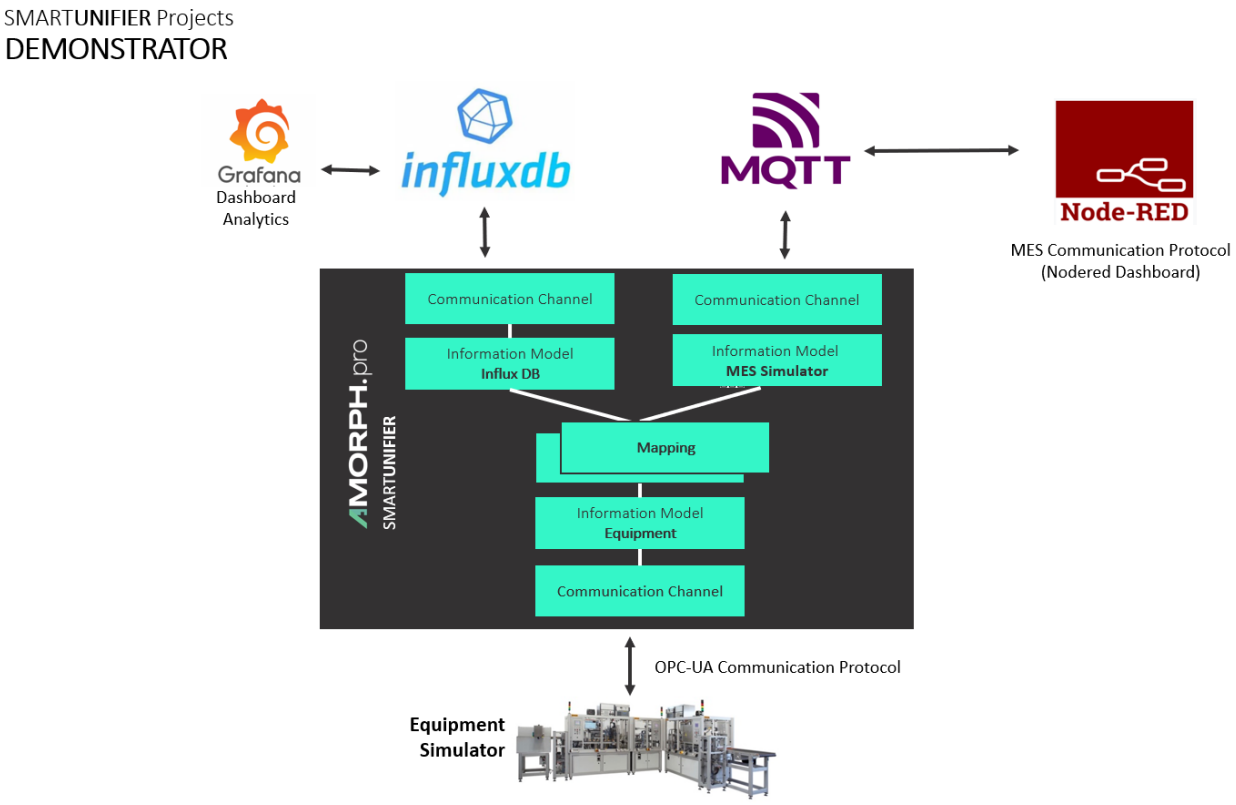
SMART**UNIFIER** Demonstrator is a package that allows users to simulate the connection between a MES system and a production equipment. The package contains all the necessary tools to run a complete communication scenario out of the box.

1.1.1 Components

The Demonstrator package contains the following components:

- **SMARTUNIFIER Manager** – a modern web application to create SMART**UNIFIER** Instances that enable the communication between the MES system and the Equipment simulator.
- **Node-Red** - a Docker image containing the NodeRed application preconfigured to act as a MES Simulator.
- **Equipment Simulator** - a Docker image that contains a custom application that simulates a real PLC from a production equipment.
- **MQTT Broker** - a Docker image containing the Moquette application acting as a message bus server.
- **Influx Database** - a Docker image containing the Influx Database where the data coming in from the Equipment Simulator will be stored and used to build visualizations.
- **Grafana** - a Docker image containing a preconfigured Grafana application that has a built-in dashboard displaying the key parameters sent by the Equipment Simulator.

1.1.2 Data Flow Diagram



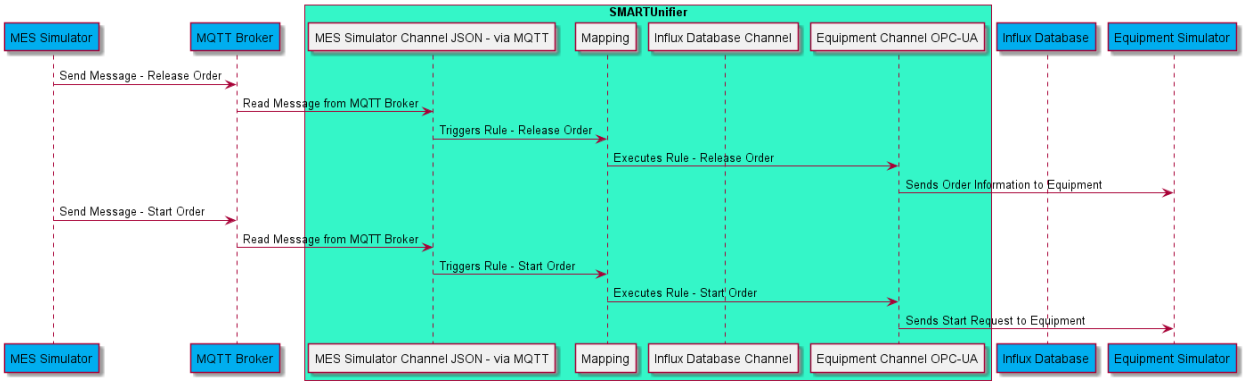
1.1.3 Demonstrator Artefacts Structure

The following table shows the SMART**UNIFIER** artefacts that are used to create this demo:

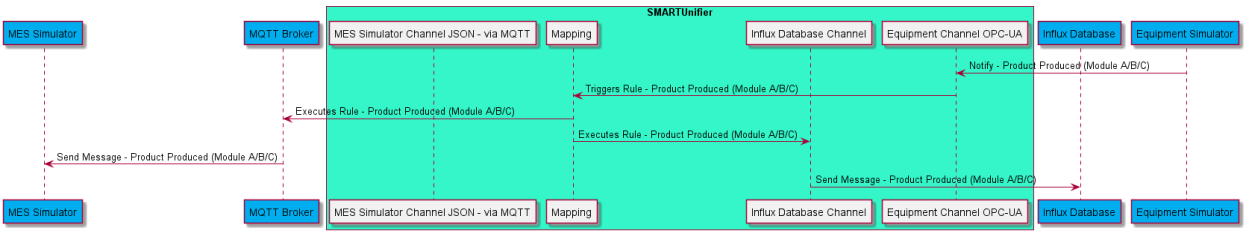
Group	Type	Name	Description
su.demo.dashboard	Information Models	Analytics	Stores data from the PLC on InfluxDb
		MESSimulator	Represents the MES Simulator structure
		SiemensS7PLC	Represents the PLC blocks structure
	Communication Channels	SiemensS7PLC	Represent the PLC communication protocol
		PLCToInfluxDb	Is used to transmit data from PLC to InfluxDb
		MESSimulator	Is used to transmit data from MES Simulator to MQTT broker
	Mappings	PLCToInfluxDb	Defines when and how to extract data from the PLC and store it on the InfluxDb
		PLCToMESSimulator	Defines data exchange between the PLC and the MES Simulator
	Device Types	SUDeviceType	Represents the template for the SMARTUNIFIER Instance
	Instances	SUInstance	Represents the configuration for the runnable application

1.1.4 Simulation Process Flow Diagram

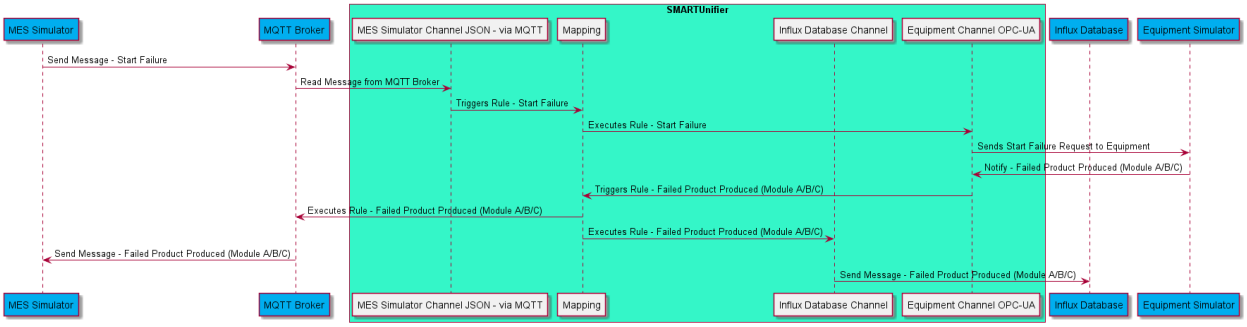
Release/Start - Order Request



Equipment Response - Notify Product Produced



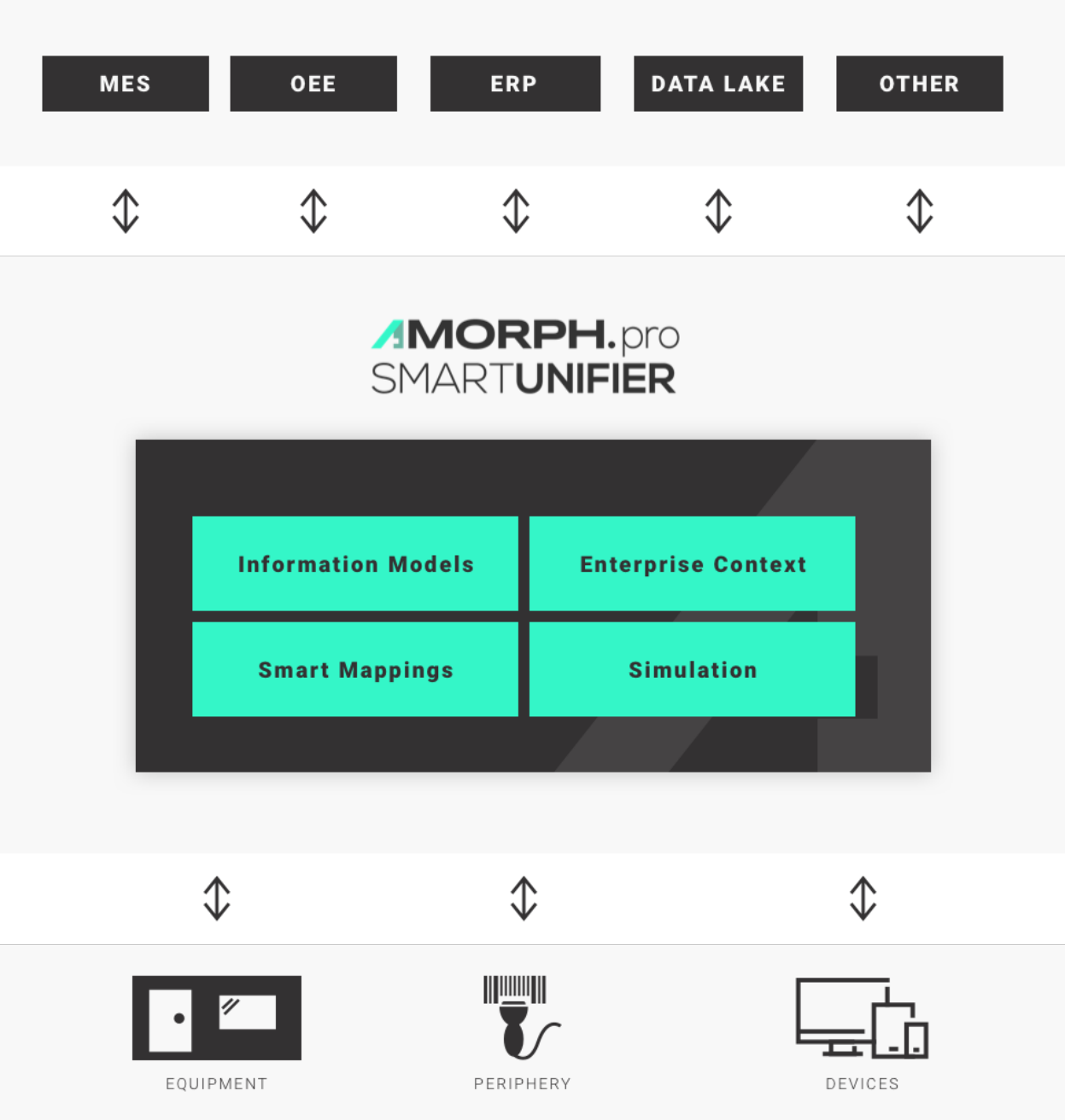
Introduced Failure



1.2 What is SMARTUNIFIER

SMART**UNIFIER** represents a powerful but very easy to use decentralized industrial connectivity platform for interconnecting all industrial devices and IT systems including equipment, peripheral devices, sensors/actors, MES, ERP as well as cloud-based IT systems.

SMART**UNIFIER** is the tool of choice for transforming data into real value and for providing seamless IT interconnectivity within production facilities.



1.3 What does SMARTUNIFIER do

- SMART**UNIFIER** provides an easy way to collect data from any Data Source and is able to transmit this data to any Data Target.
- Data Sources and Data Targets (commonly referred to as Communication Partners) in this respect may be any piece of equipment, device or IT system, communicating typically via cable or Wi-Fi and using a specific protocol like e.g., OPC-UA, file-based, database, message bus.

- With SMART**UNIFIER** several Communication Partners can be connected simultaneously.
- With SMART**UNIFIER** it is possible to communicate unidirectional or bidirectional to each Communication Partner. I.e., messages and events can be sent and received at the same time.
- SMART**UNIFIER** is able to translate and transform data to any format and protocol that is required by a certain Data Target. This includes different pre-configured protocols and formats, e.g., OPC-UA, file-based, database, message bus, Webservices and many direct PLC connections. In case a certain protocol or format is currently not available it can be easily added to SMART**UNIFIER**.
- By applying so called Information Models, SMART**UNIFIER** enables the same view to data regardless of the protocol or format being used to physically connect an equipment, device or IT system.
- A big advantage of SMART**UNIFIER** is, that in many cases there is no need for coding when providing connectivity between different Communication Partners. SMART**UNIFIER** Mappings enable users to assign data sources to data targets via drag and drop.

CHAPTER

TWO

INSTALLATION OVERVIEW

Installing SMART**UNIFIER** on a host is the first step in realizing your connectivity scenario.

1. See the *system requirements* prior to the installation.
2. Amorph Systems supports using SMART**UNIFIER** on the following operating systems:
   - *Windows*
   - *Linux*

2.1 System Requirements

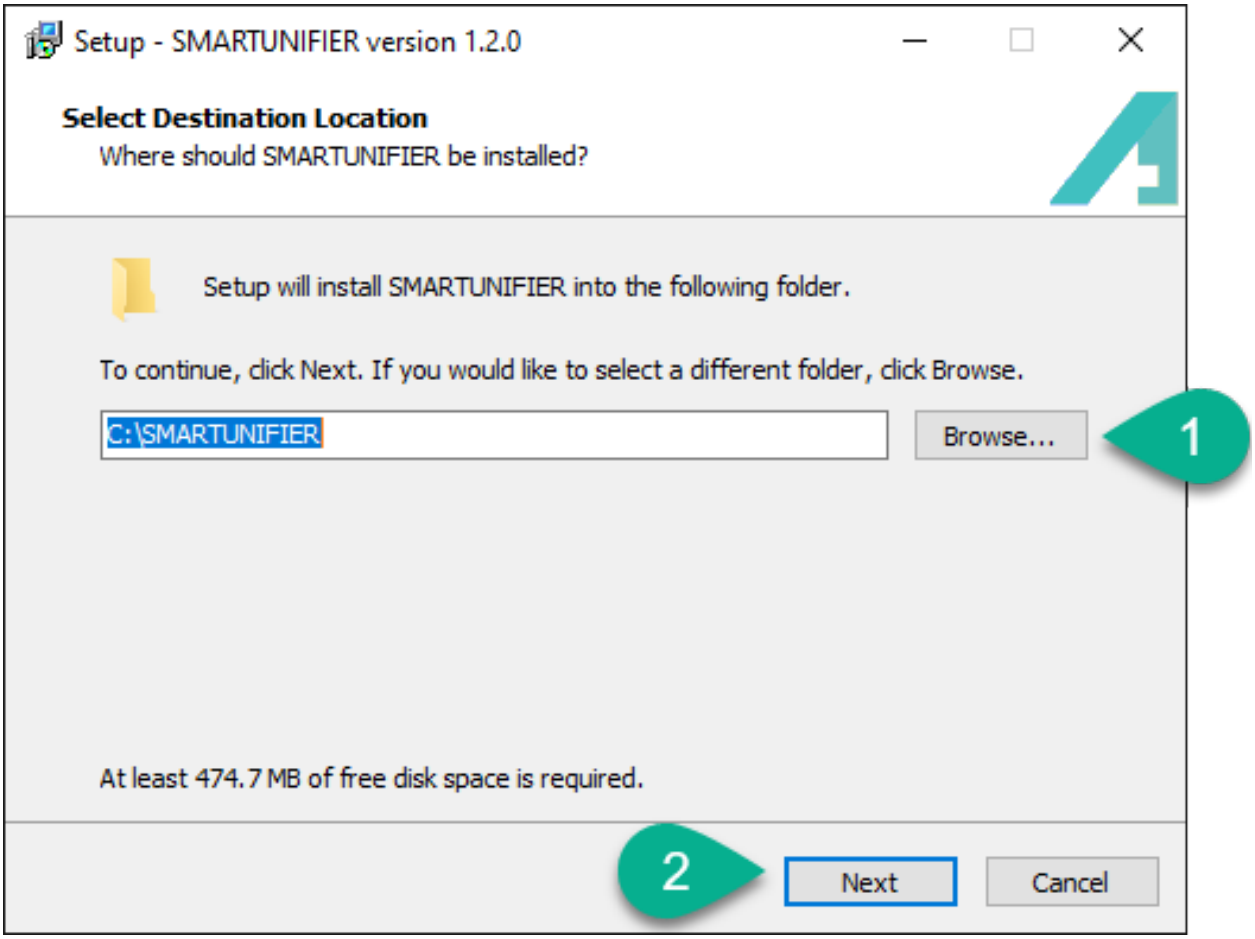
Minimum requirements for running the SMART**UNIFIER** Manager

- **Computer and Processor**: 1 GHz or faster, x86-bit- or x64-bit-processor.
- **Memory**: 512 MB RAM.
- **Hard Disk / SSD**: 1 GB free space.
- **Display PC (Engineering, Dashboard)**: 1280 x 1024 Resolution.
- **Mobile Devices (Dashboard)**: Apple iPhone 6 or higher, Android.
- **Operating System**: Windows 10, Windows 8, Windows 7, Windows Server 2016, Windows Server 2012 R2, Windows Server 2012, Linux, MacOS - For an optimal user experience always use the newest version of the operating system.
- **Browser**: Latest version of Chrome, Microsoft Edge/Internet Explorer, Firefox.
- **Other**: Latest version of Docker Daemon (including Docker-compose). For more details follow the on-screen instructions from https://docs.docker.com/compose/install/.

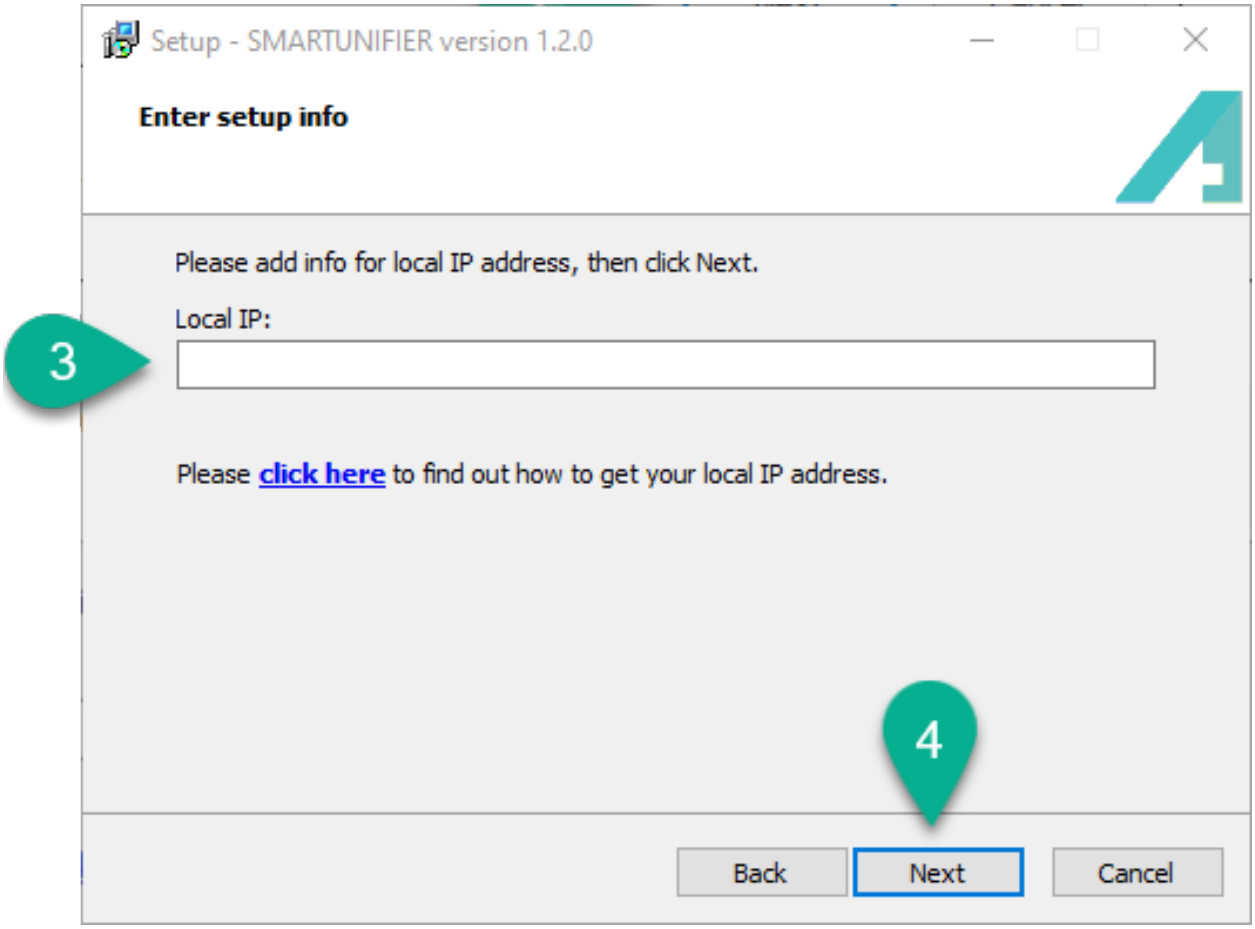
2.2 Windows

2.2.1 Step 1 - Install the SMARTUNIFIER and the Docker Components:

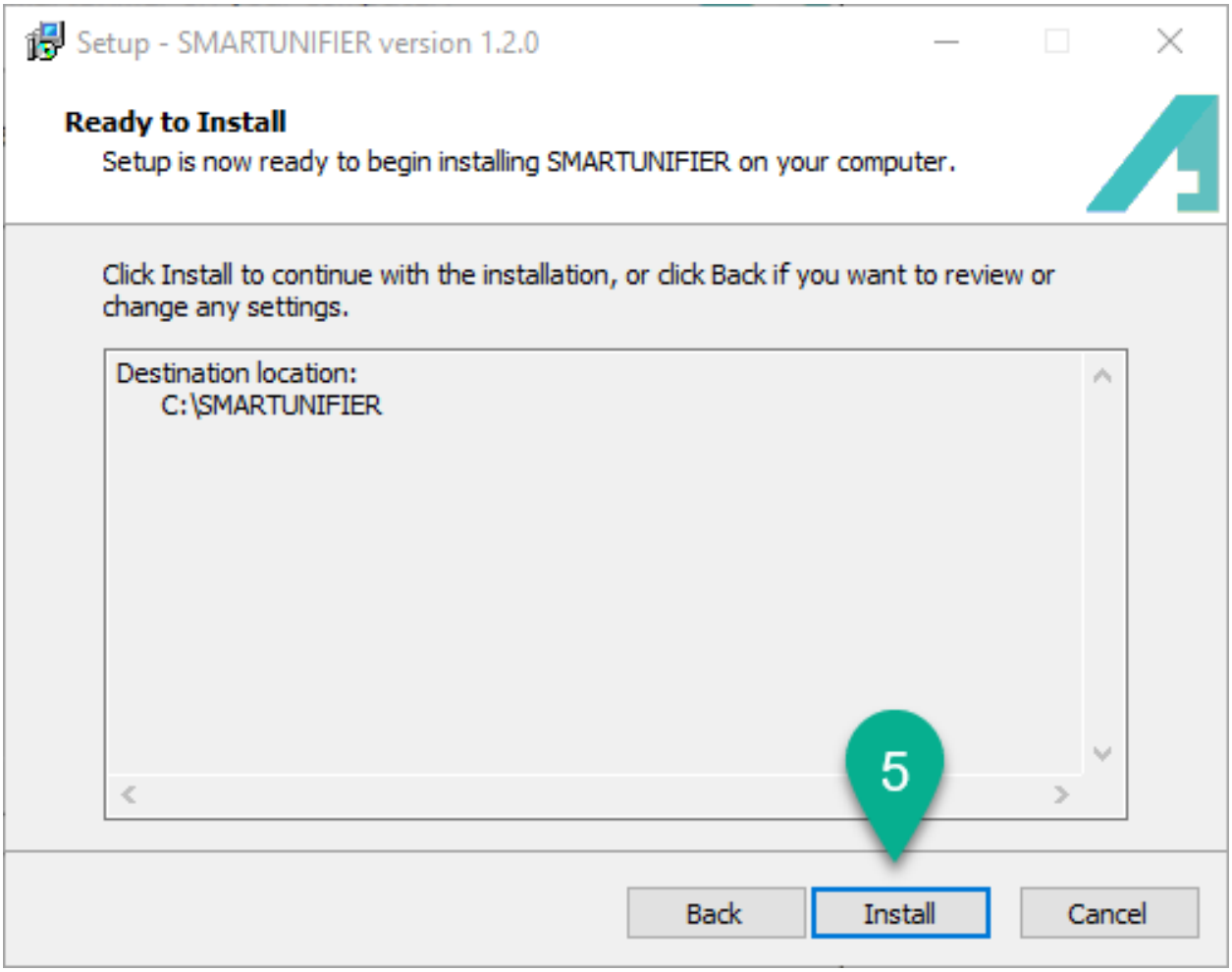
- Move the installation package to a suitable location.
- Open “Docker Desktop” application.
- Run the **.exe** file to start the setup.
- Click on the “Browse” button **(1)** to change the location of the installation.
- Select the “Next” button **(2)** to continue.



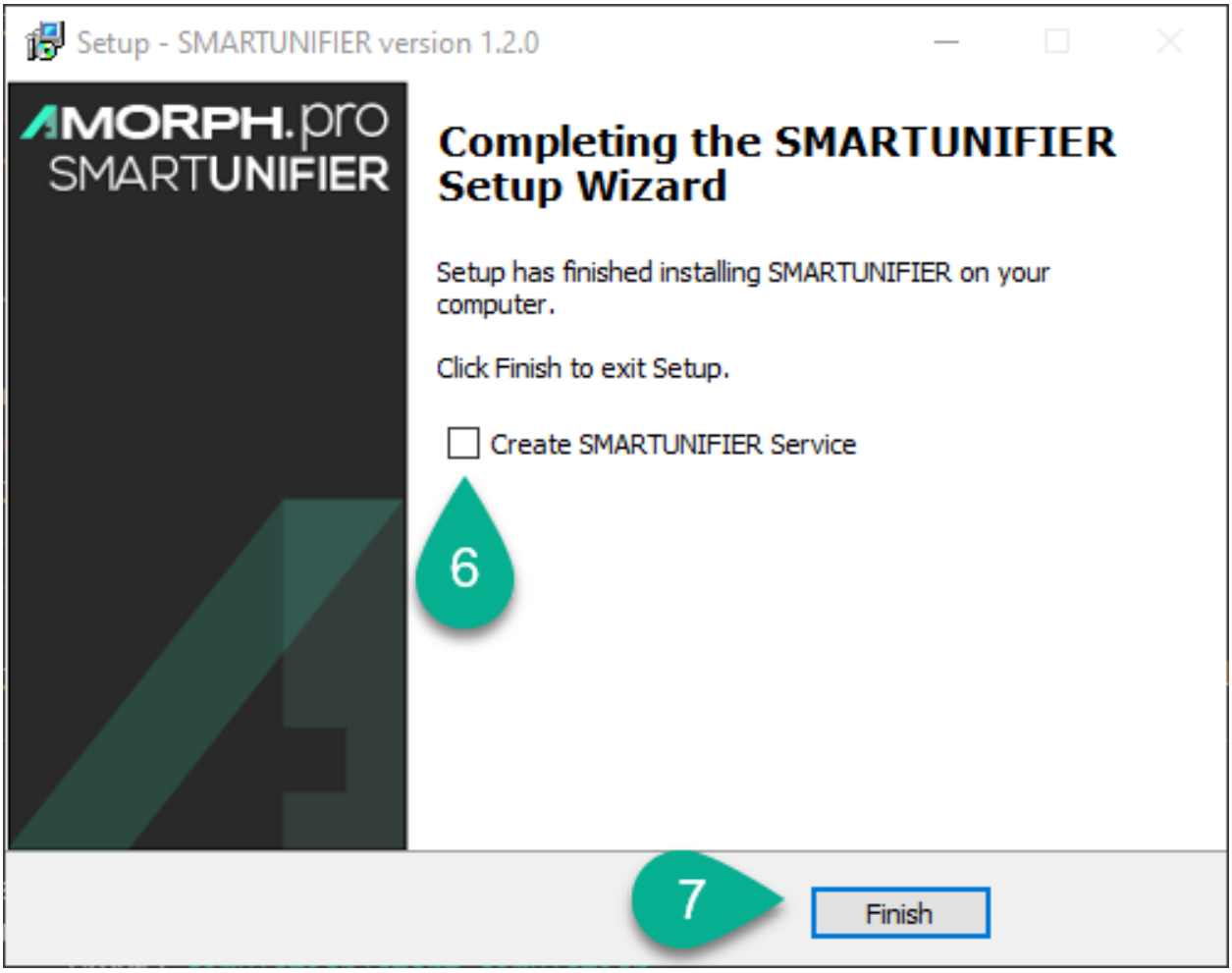
- Enter the local IP address **(3)**. Select the “Next” button **(4)** to continue.



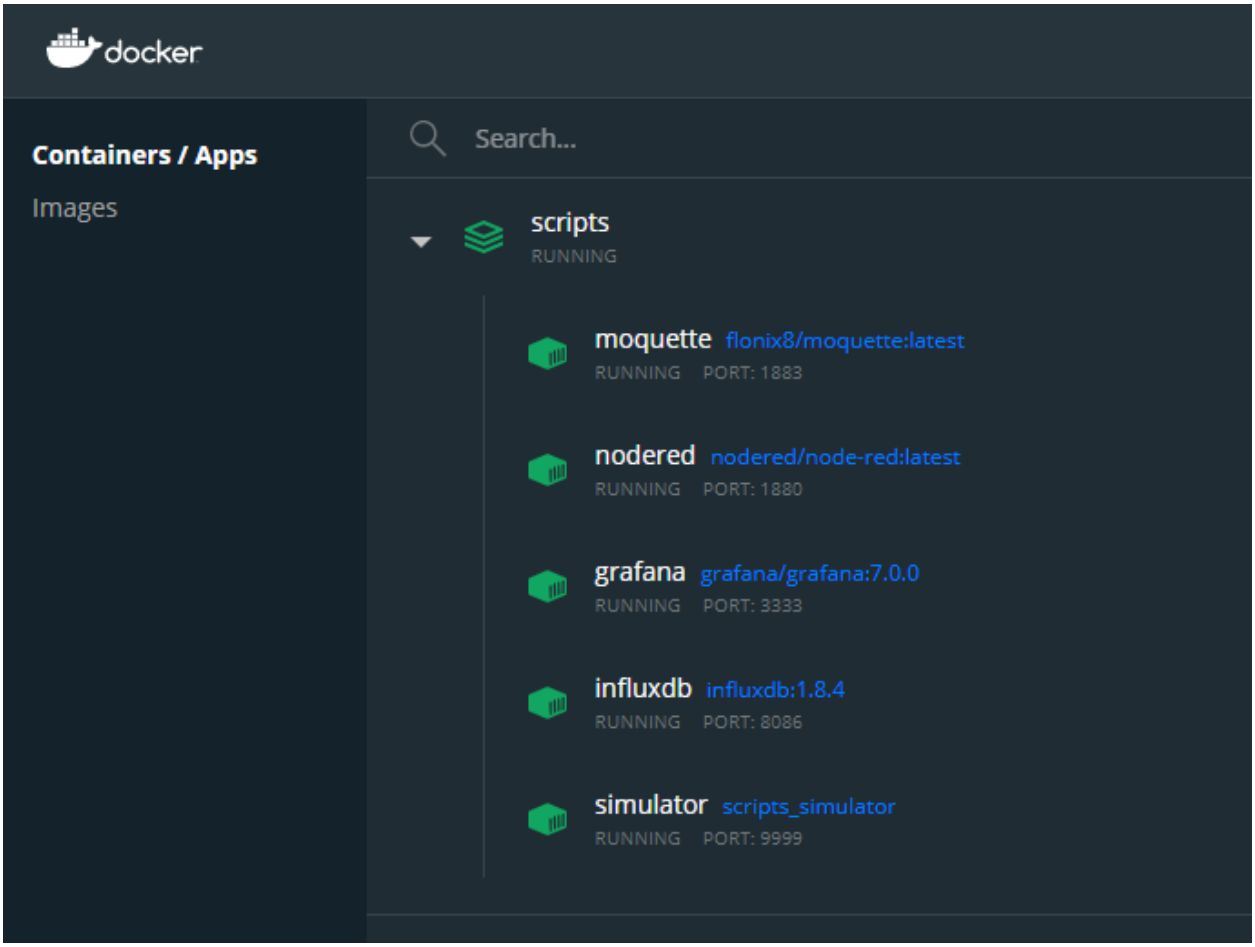
- Click on the “Install” button **(5)** to start the installation.



- Check the box **(6)** to create SMART**UNIFIER** Service (optional).



- Select the “Finish” button **(7)** to finallize installing the SMART**UNIFIER**. The console will open and the extraction of the Docker components starts.
- After the console is closed, the Docker components are installed in a few seconds:
  - moquette
  - nodered
  - grafana
  - influxdb
  - simulator



If a Docker container is deleted, it can be redeployed. From the **scripts** folder (install_location/SMART**UNIFIER**/scripts) open the console (CMD) and execute:

- to deploy InfluxDb container:

```
deploy.bat influxdb
```

- to deploy Grafana container:

```
deploy.bat grafana
```

- to deploy Node-Red container:

```
deploy.bat nodered
```

- to deploy MQTT broker container:

```
deploy.bat moquette
```

- to deploy Equipment Simulator container:

```
deploy.bat simulator
```

- to deploy all containers:

```
deploy.bat
```

2.2.2 Uninstalling

To undeploy a docker container, from the **scripts** folder (install_location/SMART**UNIFIER**/scripts) open the console(CMD) and execute:

- to remove InfluxDb container:

```
cleanup.bat influxdb
```

- to remove Grafana container:

```
cleanup.bat grafana
```

- to remove Node-Red container:

```
cleanup.bat nodered
```

- to remove MQTT broker container:

```
cleanup.bat moquette
```

- to remove Equipment Simulator container:

```
cleanup.bat simulator
```

- to remove all containers:

```
cleanup.bat
```

Follow the steps below to uninstall all the Demonstrator components (folders, files, Docker container):

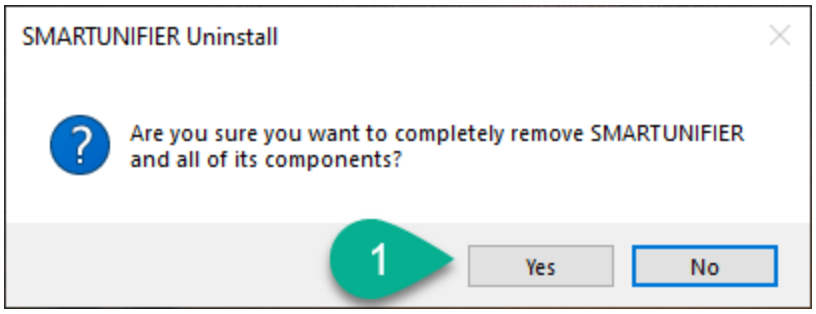
- Make sure the Demonstrator Instance is **NOT** running.
- Make sure the "Docker Desktop" application **IS** running.
- There are two options to uninstall the Demonstrator:

1. Using Windows System settings section:

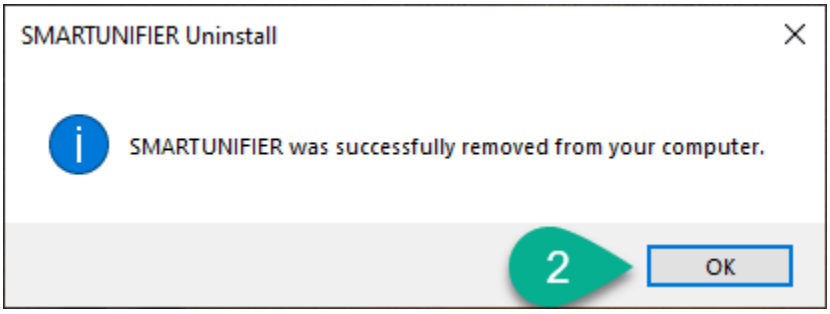
- Go to "Add or remove programs" and uninstall SMART**UNIFIER** application.

2. Run the **unins000.exe** file from the "SMART**UNIFIER**" folder.

- Select the "Yes" button **(1)** to confirm.



- Select the "OK" button **(2)** to finish.



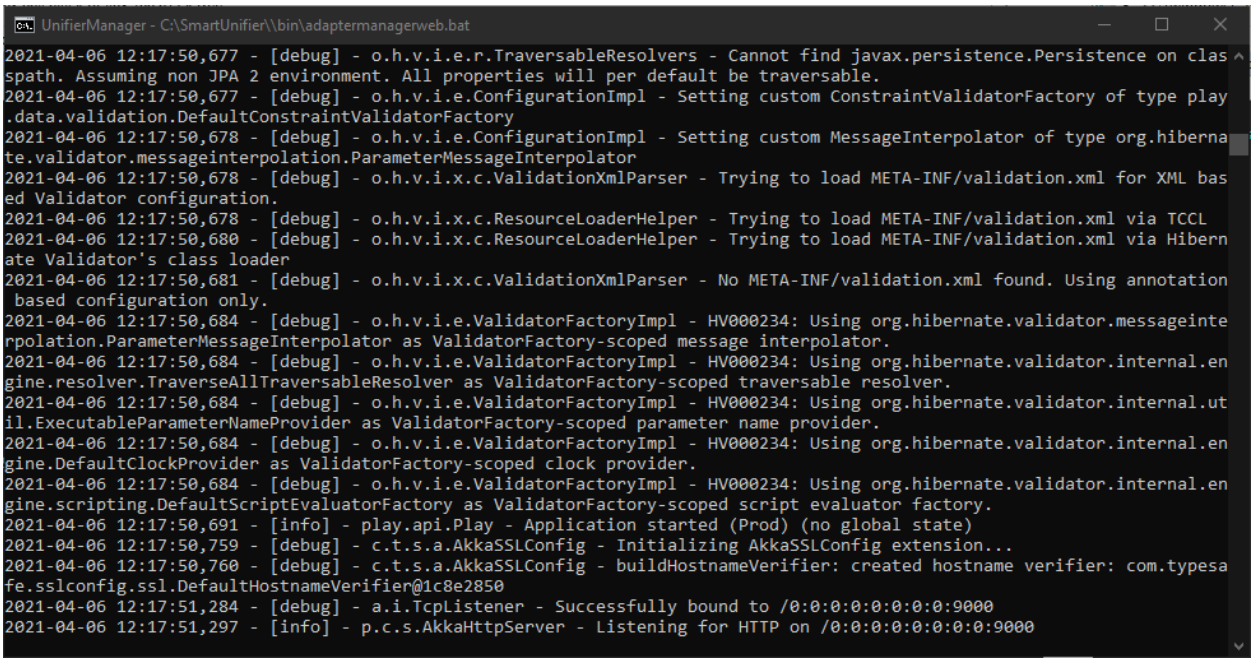
All the Demonstrator components are removed.

2.2.3 Step 2 - Running the SMARTUNIFIER

SMART**UNIFIER** can be started as an application or as a service.

2.2.4 Running SMARTUNIFIER as an Application

- If SMART**UNIFIER** was not installed as a service, execute the **UnifierManager.bat** script located in the installation folder. Afterwards the SMART**UNIFIER** Manager Console appears on the screen.

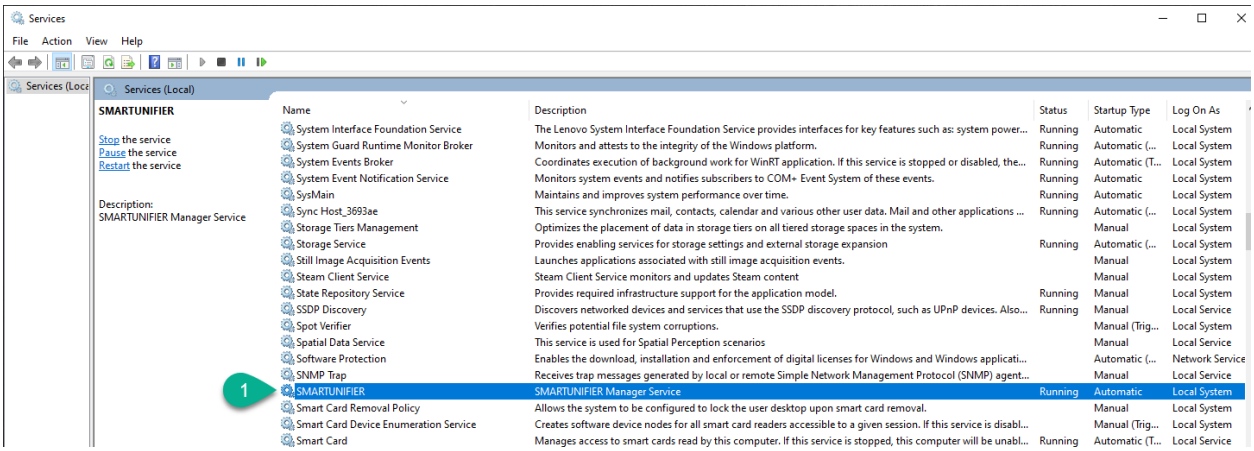


- After successfully starting up the SMART**UNIFIER** Manager, it can be accessed by opening an Internet Browser (e.g., Chrome or Firefox) and navigating to http://localhost:9000. Use the administrator credentials to login:
- Username: **admin**
- Password: **admin**

Note: The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

2.2.5 Running SMARTUNIFIER as a Service

- If SMART**UNIFIER** was installed as a service, the service is already running.
- To check open “Services” in Windows (press the “Windows” button and type “services”) and search “SMART**UNIFIER**" from the list **(1)**.



- Open an Internet Browser (e.g., Chrome or Firefox) and navigating to http://localhost:9000. Use the administrator credentials to login:
- Username: **admin**
- Password: **admin**

2.3 Linux

2.3.1 Step 1 - Install SMARTUNIFIER and Docker Components:

- Move the installation package to a suitable location. Make sure the path to the directory does not include any white spaces!
- Extract the **.tar.gz**-archive.

```
tar -xvzf SMARTUNIFIER-Manager-linux-x64.tar.gz
```

- To change the deploy file properties and to convert it's format to Unix, open the terminal from the **scripts** folder (install_location/SMART**UNIFIER**/scripts) and execute the following commands:

```
chmod 775 deploy.sh
```

```
dos2unix deploy.sh
```

- To deploy the Docker components execute the following commands (terminal opened from the **scripts** folder):

```
sh deploy.sh
```

- During the Docker components deploy, the input of the local IP address is required. To get the local IP run the command `ifconfig`.
- After entering the local IP address, the Docker components are deployed in a few seconds:

- moquette
- nodered
- grafana
- influxdb
- simulator

2.3.2 Step 2 - Running SMARTUNIFIER:

- Start the SMART**UNIFIER**Manager by executing in a terminal the following commands:

```
chmod +x UnifierManager.sh
```

```
./UnifierManager.sh
```

- After successfully starting the SMART**UNIFIER** Manager, it can be accessed by opening an Internet Browser (e.g., Chrome or Firefox) and navigating to http://localhost:9000. Use the administrator credentials to login:
- Username: **admin**
- Password: **admin**

Note: The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

CHAPTER

THREE

INSTANCE DEPLOYMENT

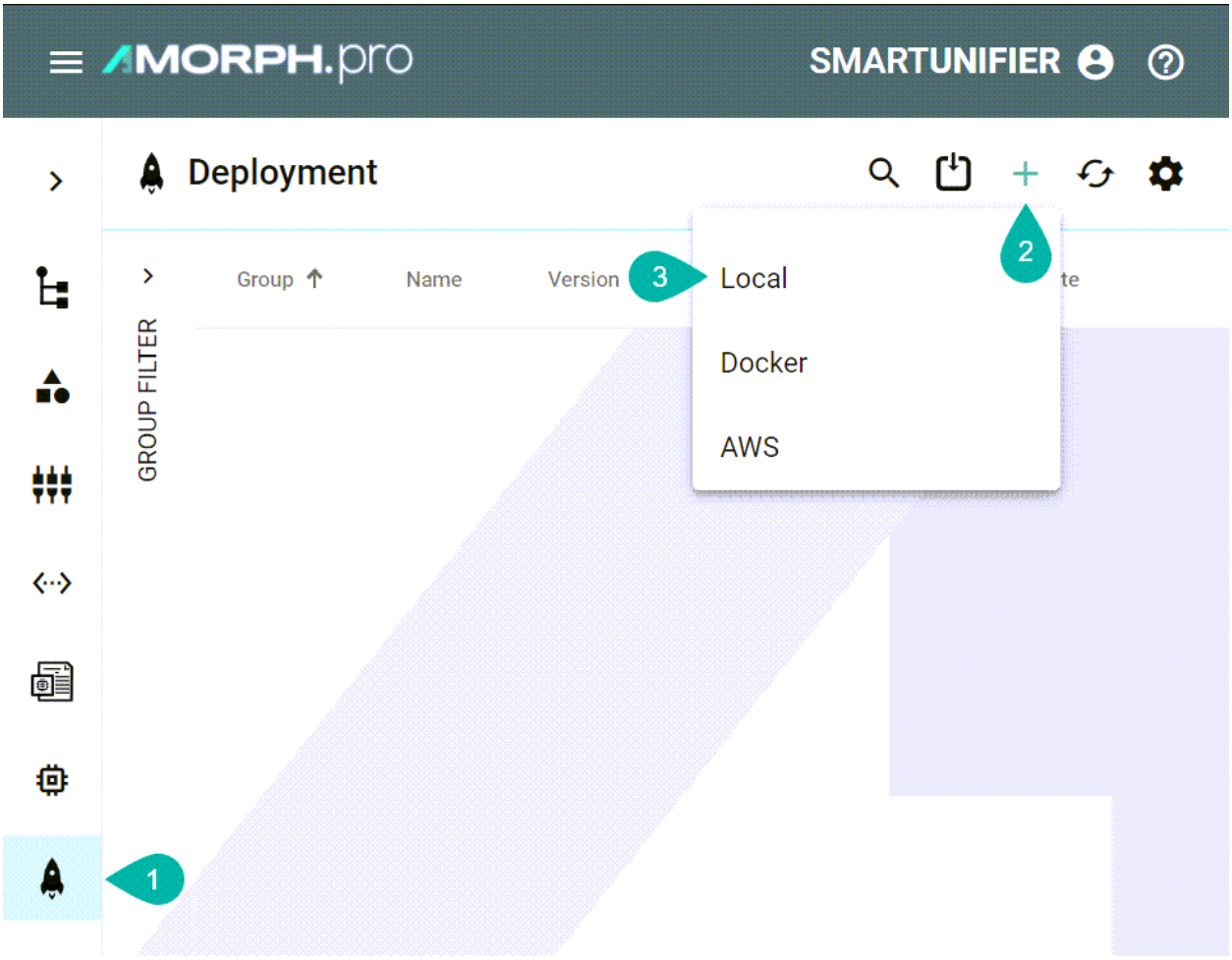
In order to run the Demonstrator you need to *Deploy and Start* the instance.

3.1 Run the Instance

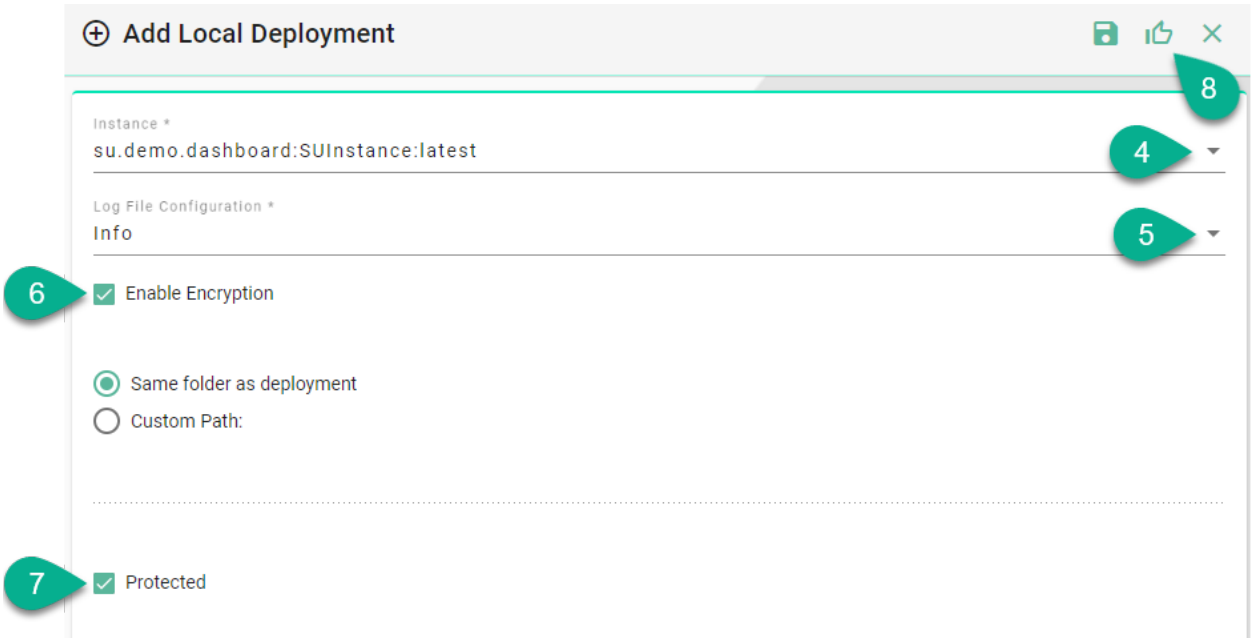
The communication between the MES system (Node-Red) and the Equipment Simulator is facilitated by the SMART**UNIFIER** Instance.

First add the Instance to a local deployment:

- Open “Deployments” section **(1)**.
- Click on the “Add” button **(2)**.
- Select the “Local” option **(3)**.



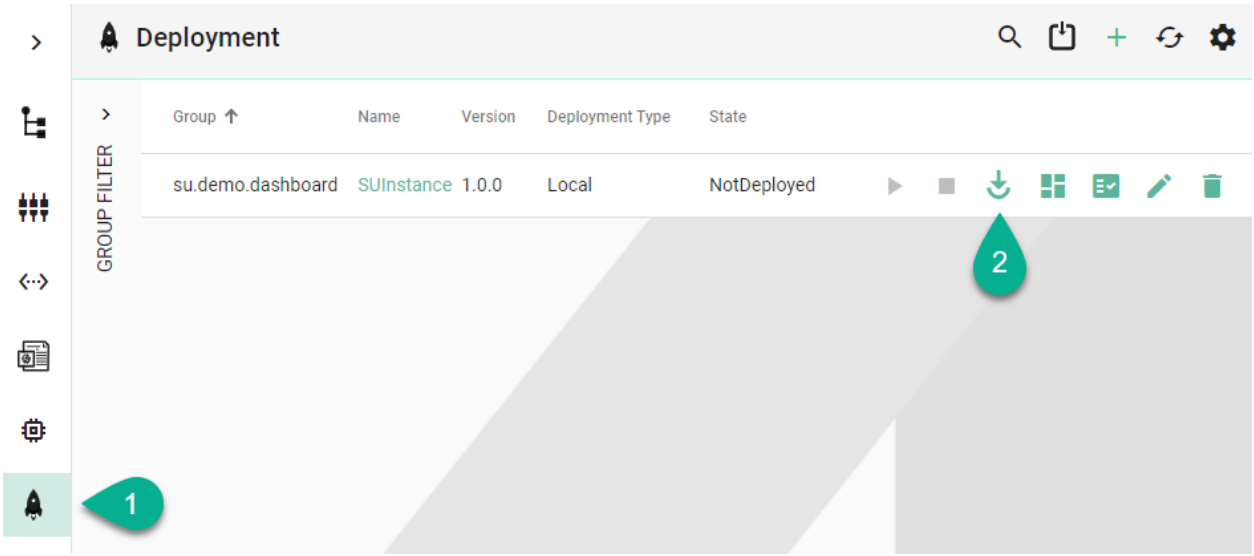
- Select the “Instance” from the dropdown **(4)**.
- Select “Info” from the “Log File Configuration” dropdown **(5)**.
- Check “Enable Encryption” **(6)** to have all credentials encrypted in the configuration files.
- Enable “Protected” **(7)** in order to secure the deployment - A confirmation will be required for further changes on the deployment (e.g., deploy, undeploy, start, stop).



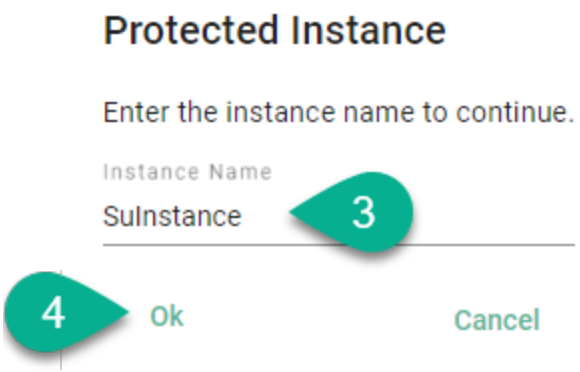
- Click the “Save and Close” button **(8)**.

Then run the Instance:

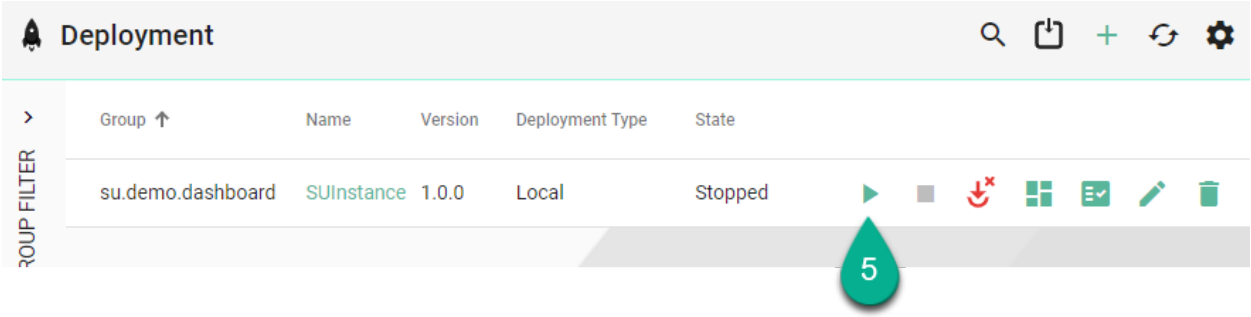
- Open the “Deployments” section **(1)**.
- Click the “Deploy” button **(2)**.



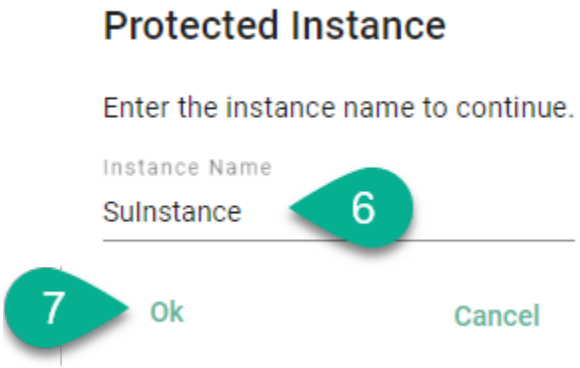
- Enter the Instance name **(3)** to confirm the action and click the “Ok” button **(4)**.



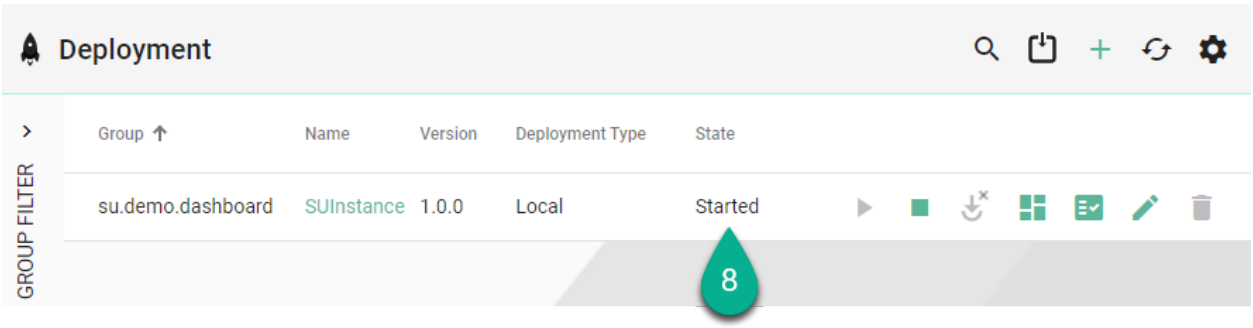
- Click the “Start” button **(5)** to run the Instance.



- Enter the Instance name **(6)** to confirm the action and click the “Ok” button **(7)**.



- The Instance is running **(8)**.


Deployment
GROUP FILTER
Group
Name
Version
Deployment Type
State
su.demo.dashboard
SUInstance
1.0.0
Local
Started
8

CHAPTER

FOUR

VISUALIZATION TOOLS

The Demonstrator visualization tools are:

- *Node-Red* application preconfigured to act as a MES Simulator.
- *Grafana* application that has a built-in dashboard displaying the key parameters sent by the Equipment Simulator.

4.1 Node-Red

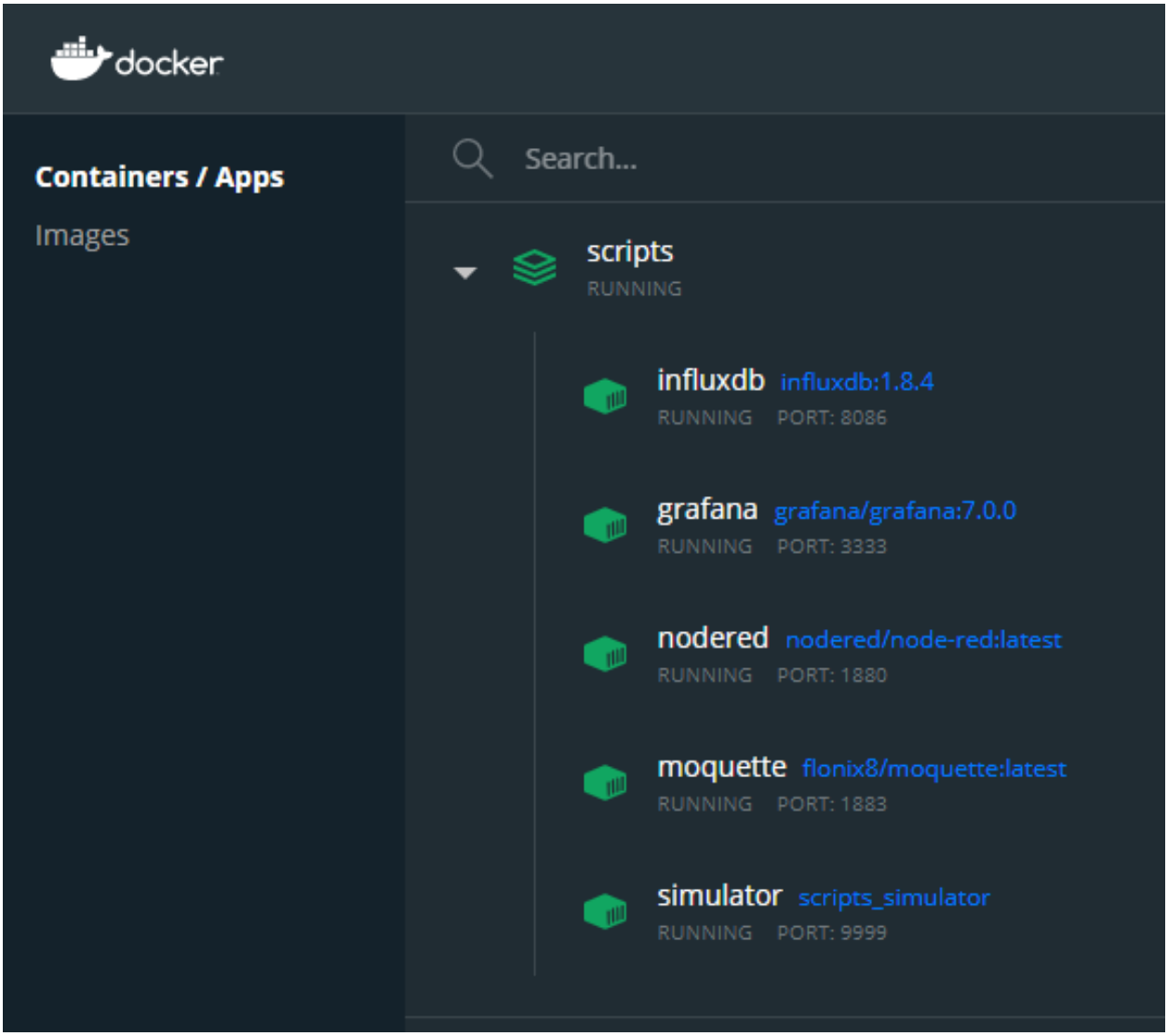
Node-RED is a tool for programming visually. It displays relations, functions and allows the user to program without having to code. Node-Red is a browser-based flow editor where you can add or remove nodes and wire them together in order to make them communicate with each other.

In the current demonstrator, Node-Red is preconfigured to act as a MES Simulator in order to provide commands/orders to the Equipment Simulator.

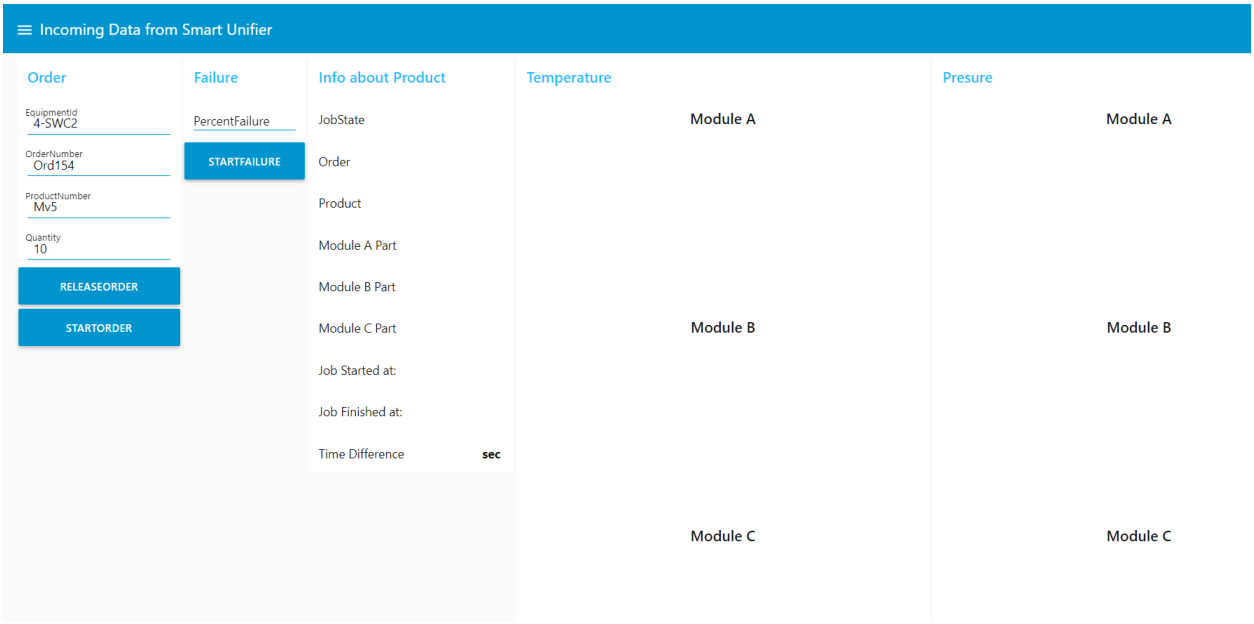
4.1.1 Access

Follow the steps below to access Node-Red:

- The Node-Red Docker container must be running.



- Open an Internet Browser (e.g., Chrome or Firefox) and navigate to http://localhost:1880/ui.



4.2 Grafana

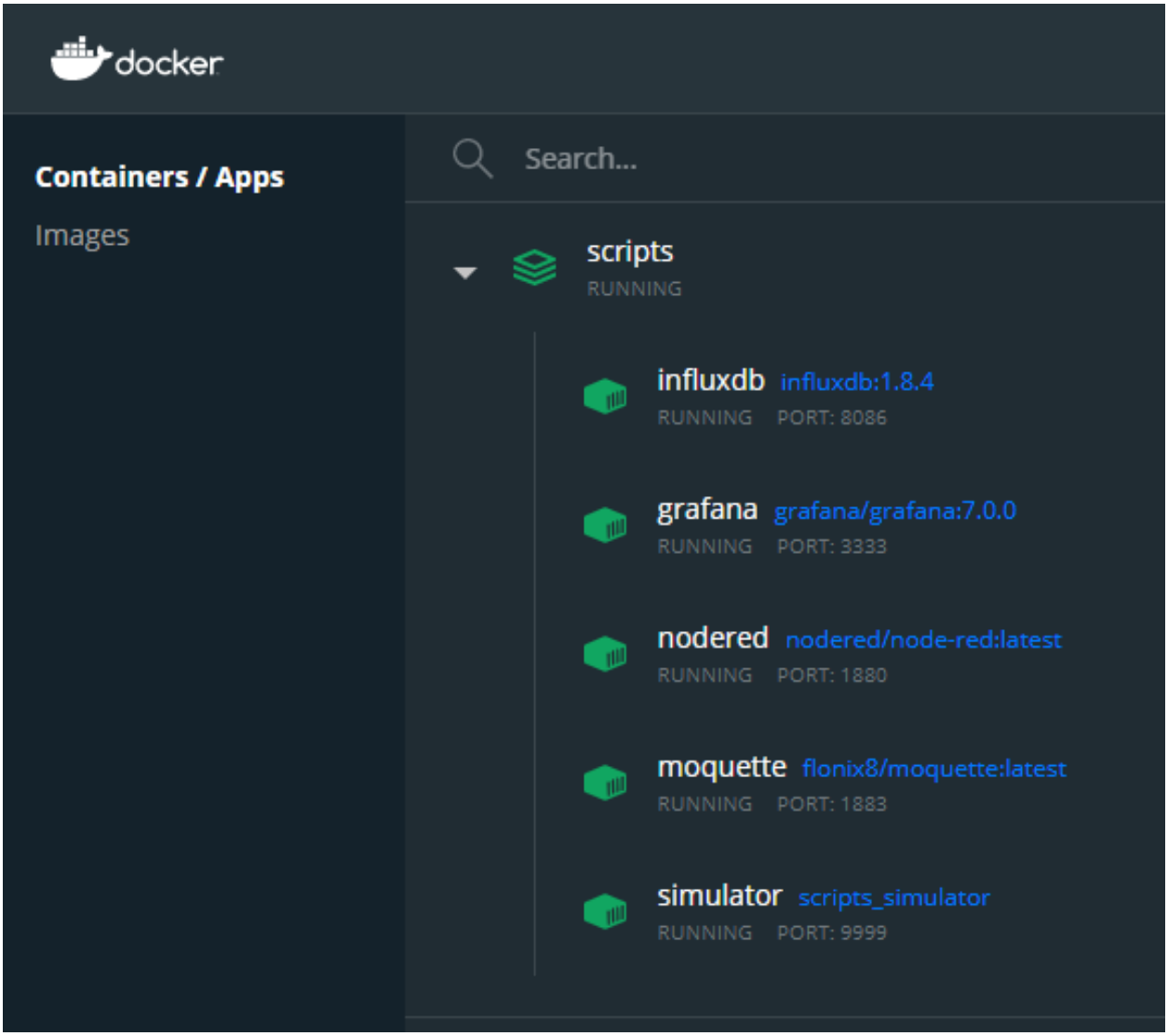
Grafana is an open source analytics and interactive visualization web application. It provides charts, graphs and alerts for the web when connected to the supported data sources.

In the current demonstrator, Grafana is used to display the key parameters sent by the Equipment Simulator.

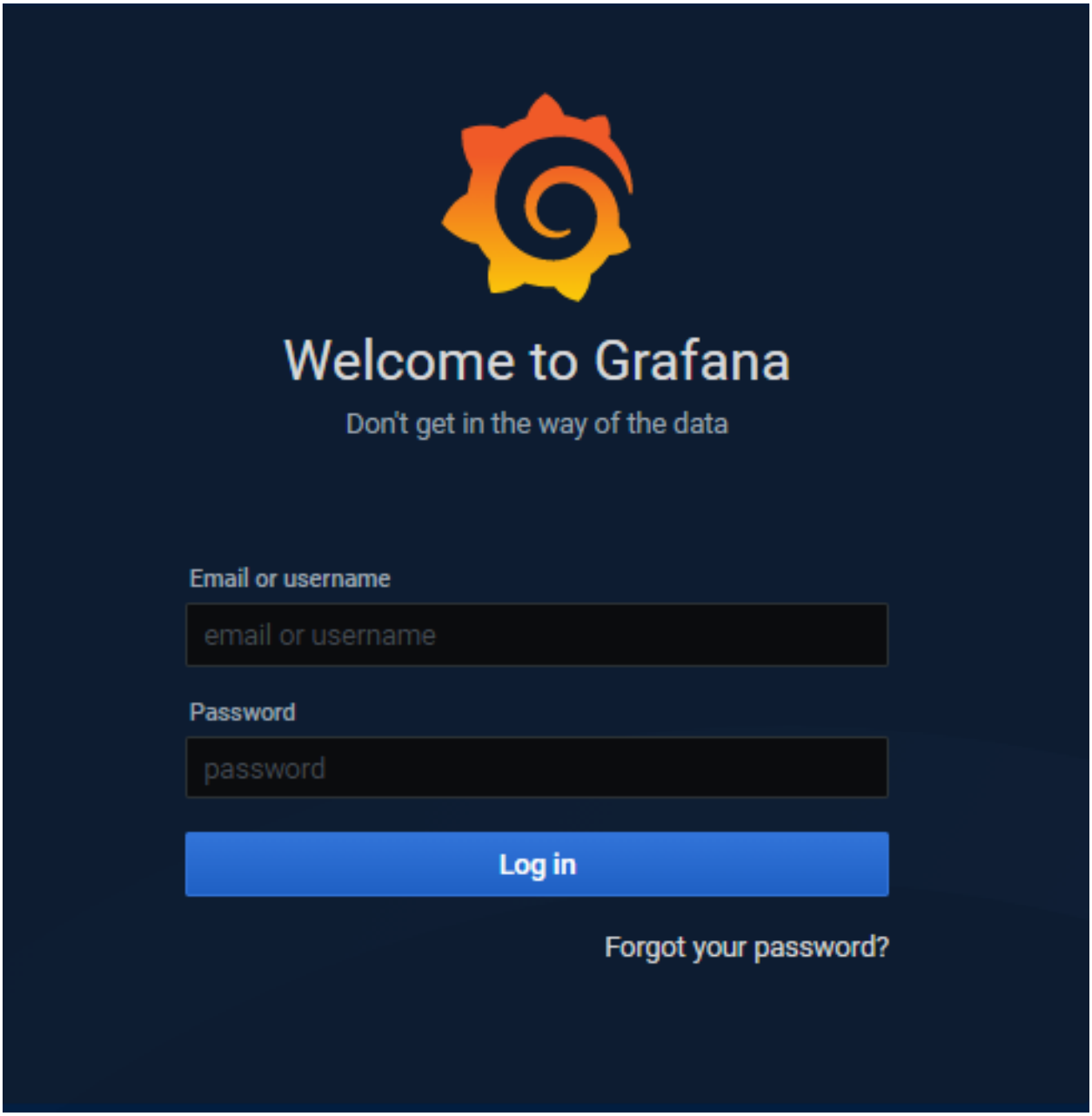
Access Grafana

Follow the steps bellow to access Grafana:

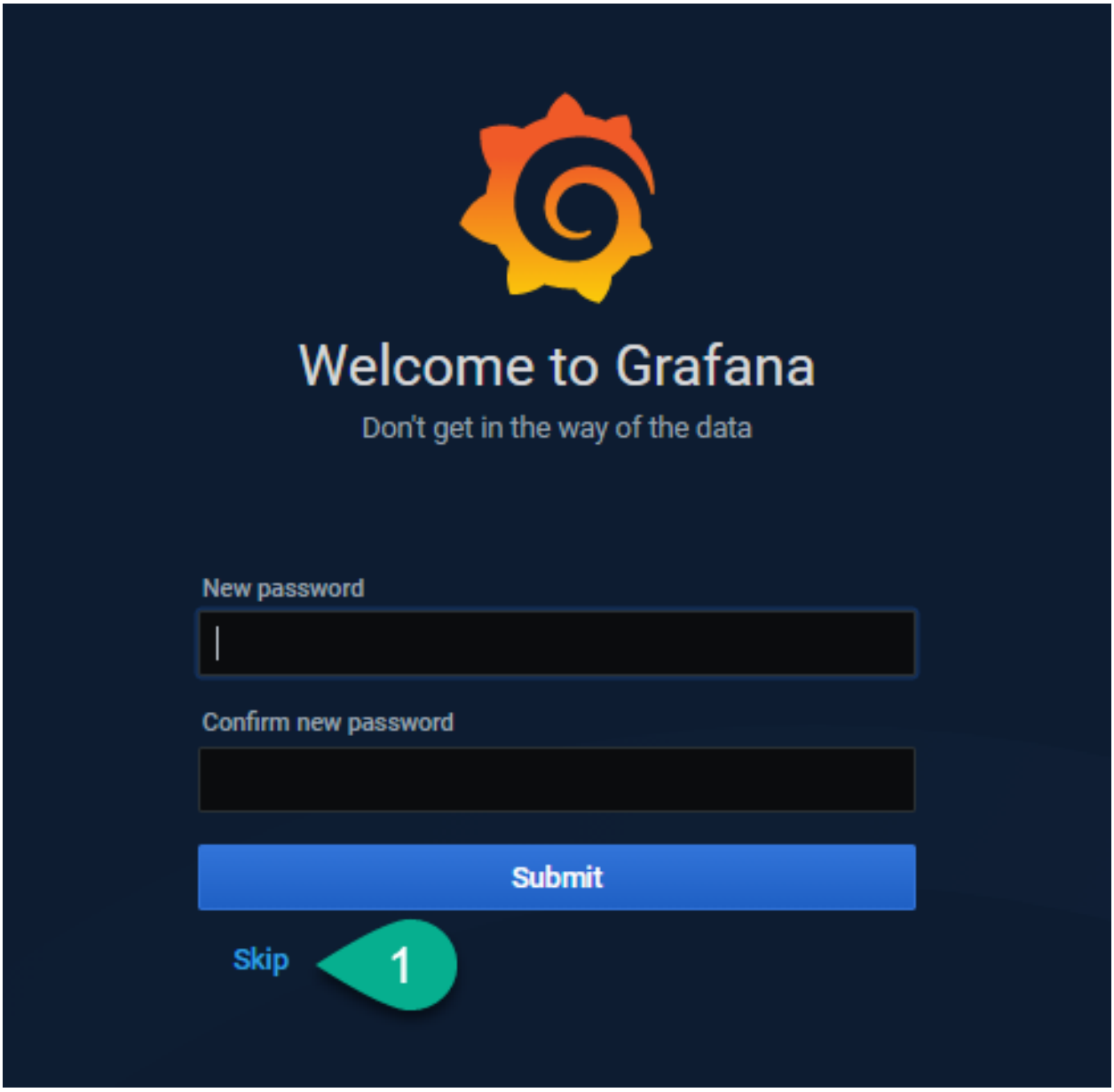
- Grafana Docker container must be running.



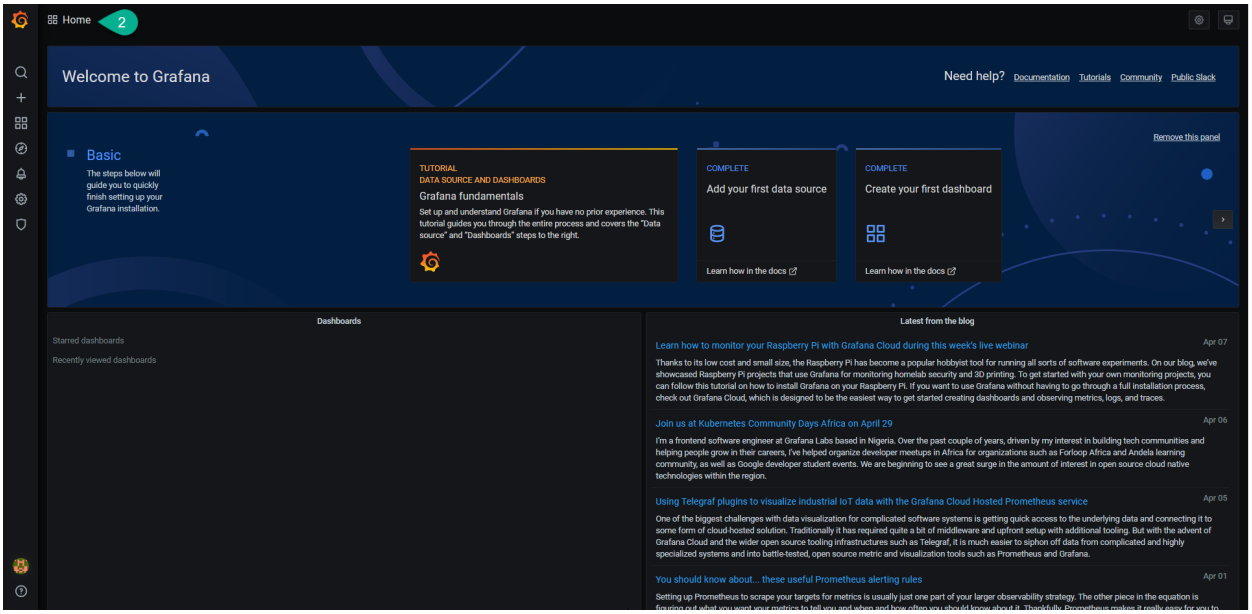
- Open an Internet Browser (e.g., Chrome or Firefox) and navigate to http://localhost:3333/. For "Username" and "Password" use **admin**.



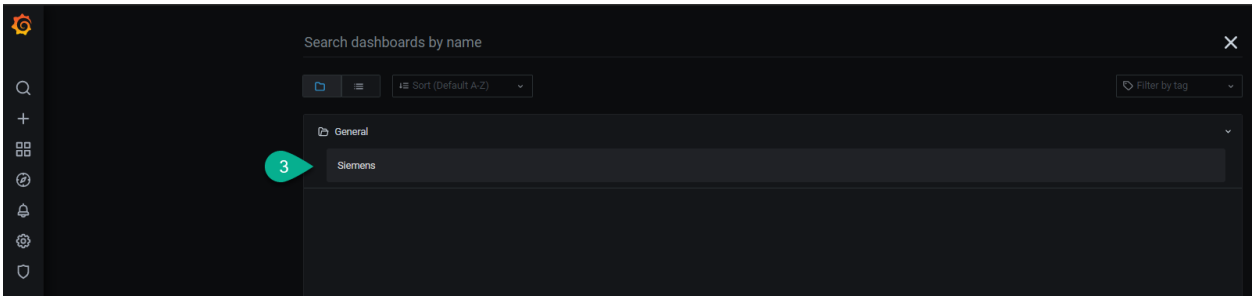
- Change the "Password" or just select the "Skip" button **(1)**.



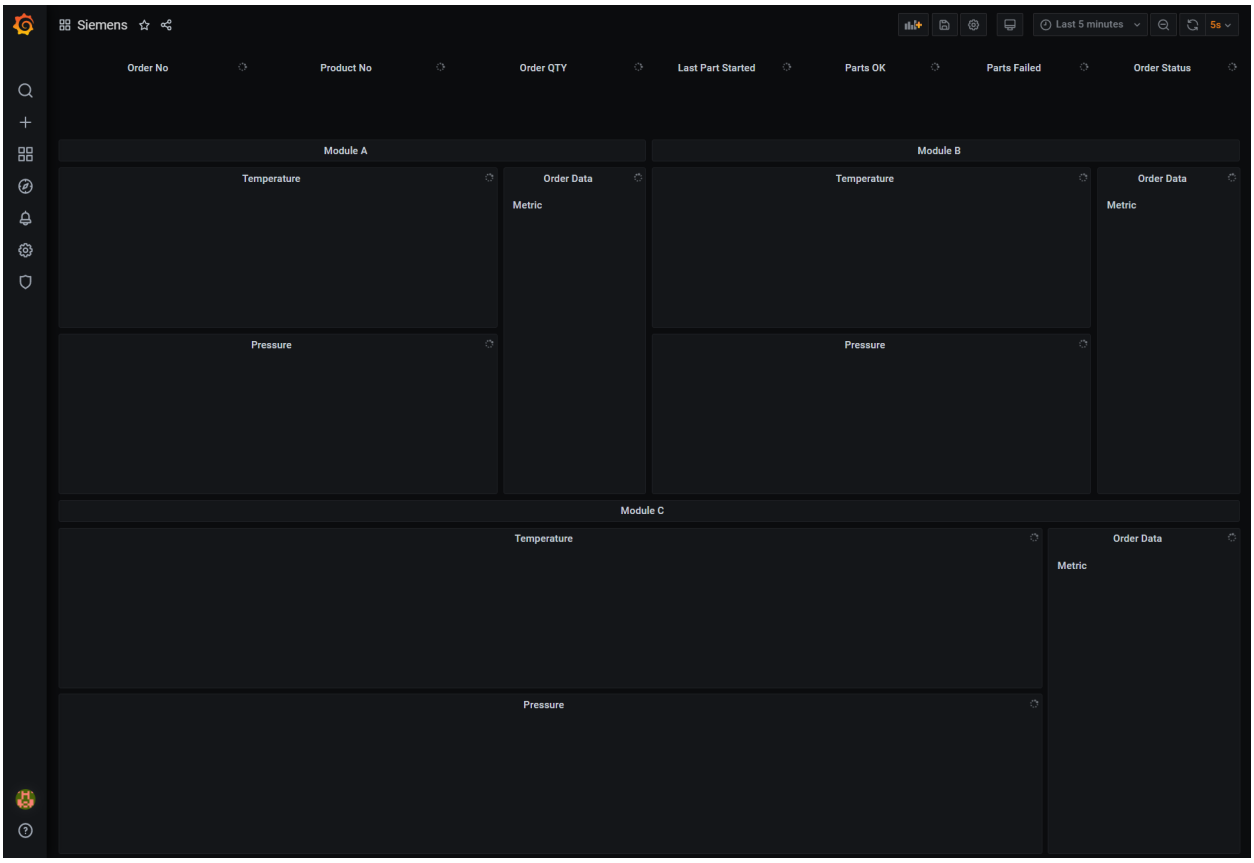
- Select the “Home” button **(2)**.



- Select “Siemens” options **(3)**.



- The Grafana Siemens Dashboard is visible.


Siemens
Last 5 minutes
5s
Order No
Product No
Order QTY
Last Part Started
Parts OK
Parts Failed
Order Status
Module A
Temperature
Order Data
Metric
Pressure
Module B
Temperature
Order Data
Metric
Pressure
Module C
Temperature
Order Data
Metric
Pressure

CHAPTER

FIVE

START THE SIMULATION SCENARIO

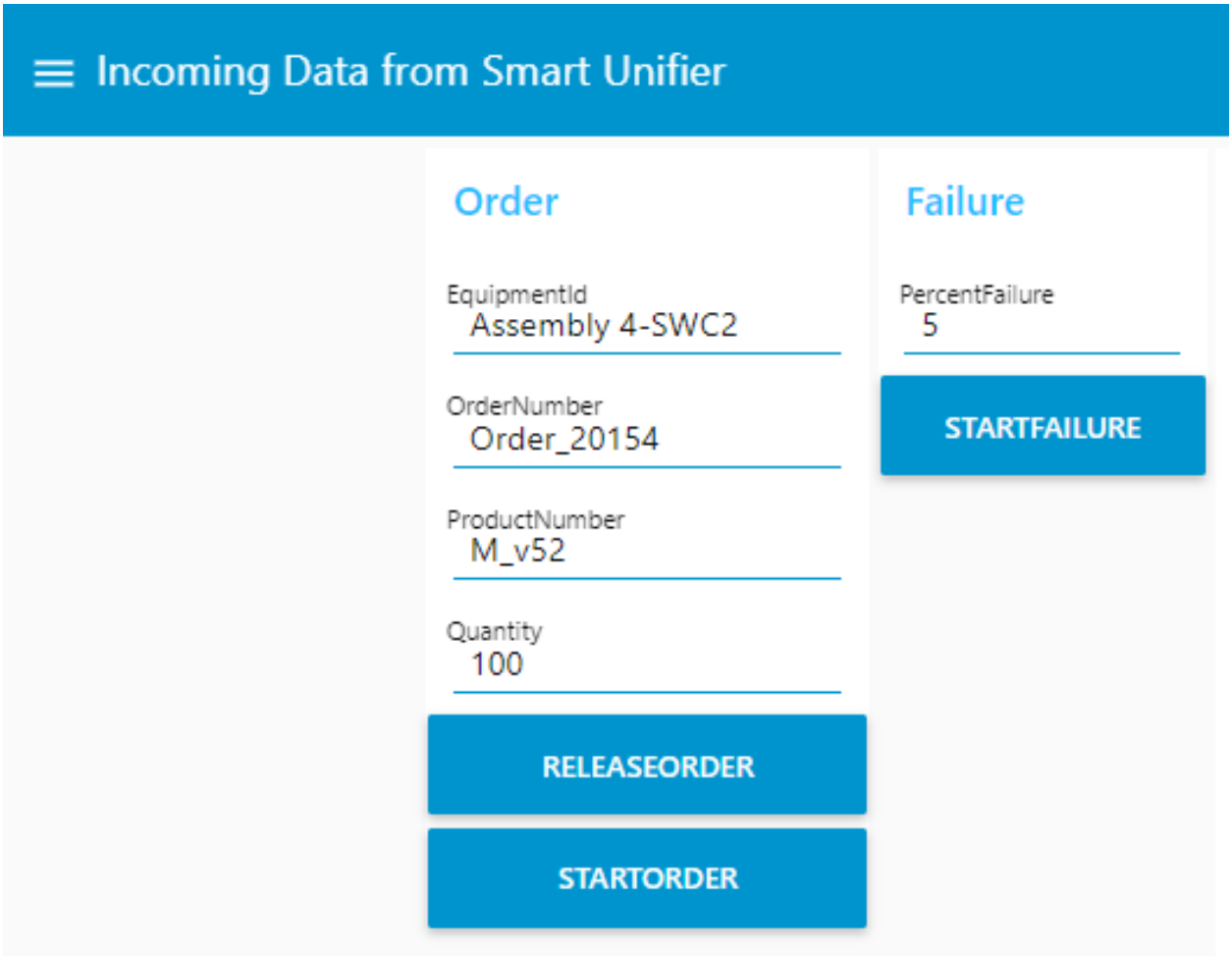
The simulation scenario flow:

1. *Input Data* - the order details.
2. *Release Order* - sending the order details to the Equipment Simulator.
3. *Start Order* - the Equipment Simulator starts the production.
4. *Start Failure* - the Equipment Simulator starts the production of failed parts.

5.1 Input Data

The input data is structured into two categories, as seen in the Node-Red dashboard (MES Simulator):

1. Order:
    - EquipmentId
    - OrderNumber
    - ProductNumber
    - Quantity
2. Failure:
    - PercentFailure



5.2 Start the Simulation

The Demonstrator use case illustrates the connection between a MES system and a production equipment, connection facilitated by the SMART**UNIFIER**.

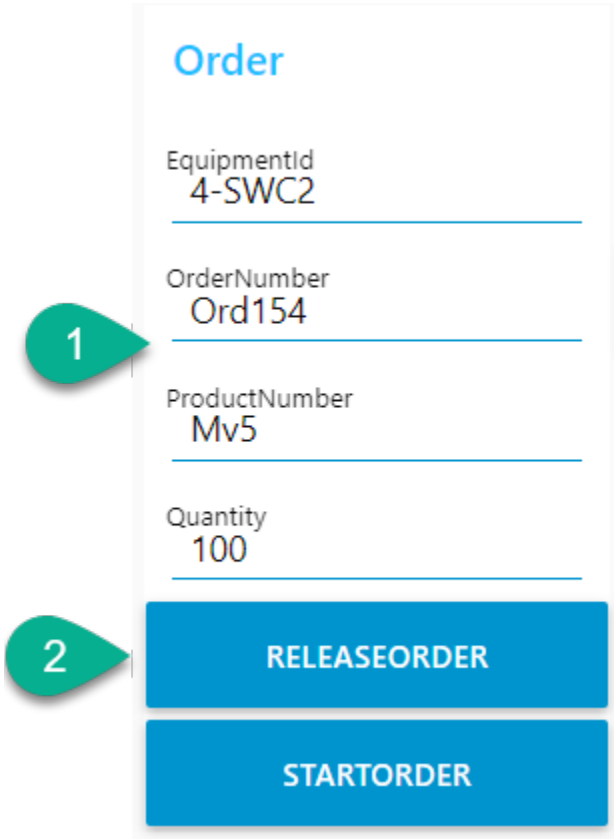
To run the simulation, three steps are required:

- Release Order
- Start Order
- Start Failure

Each simulation step can be initiated from the Node-Red Dashboard (MES Simulator). The simulation results will be visible in Grafana Dashboard.

5.2.1 Release Order

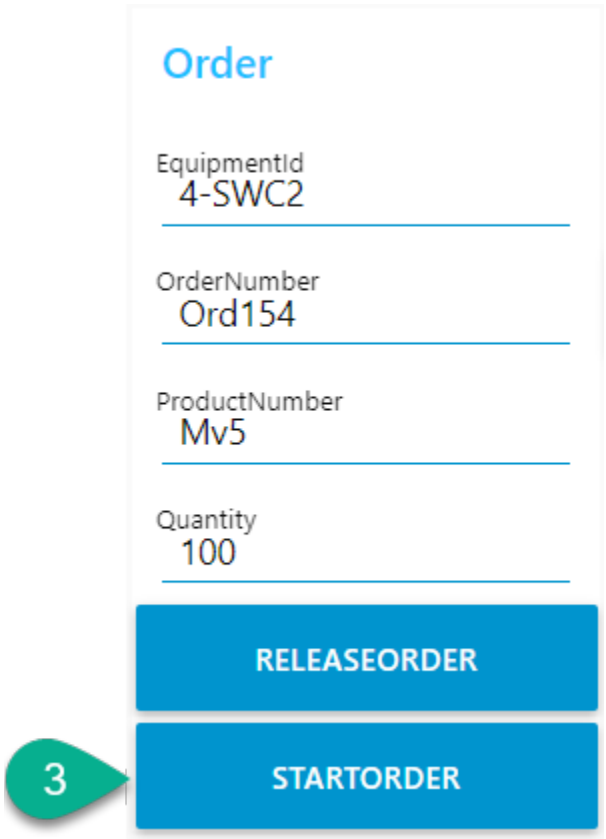
To start the simulation scenario, first input the order data **(1)** and release the order **(2)** from the Node-Red dashboard (MES Simulator).



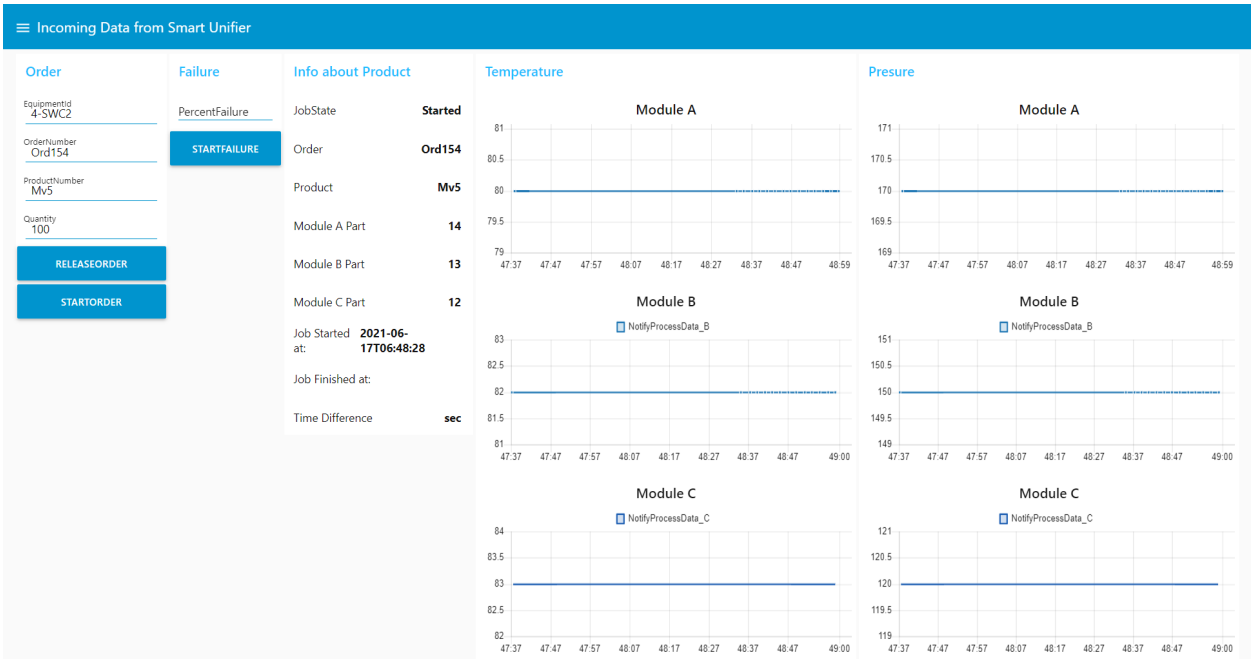
Releasing of the order posts a message on the MQTT Broker that is picked up by the SMART**UNIFIER** Instance and sent to the Equipment Simulator.

5.2.2 Start Order

Second step is to Start the order **(3)** from the Node-Red dashboard. A new message is posted on the MQTT Broker and via the SMART**UNIFIER** Instance sent to the Equipment Simulator instructing it to start the production.



The Equipment Simulator is built to have 3 production modules. The produced parts start in Module A continue production in Module B and finish on Module C. Once the Equipment Simulator starts to produce parts every Module will mark the production of a part.

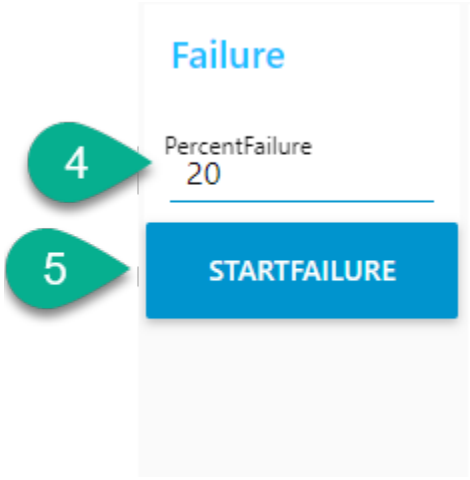


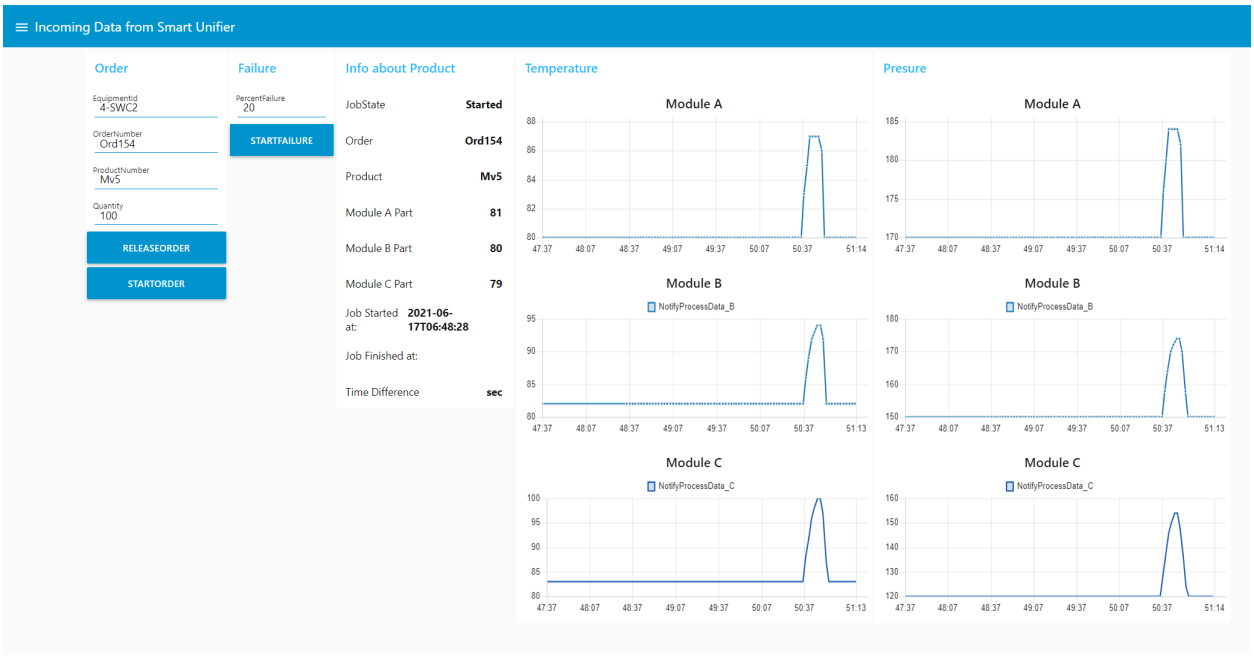
The change that happens on each module is picked up by the SMART**UNIFIER** and data is sent

simultaneously towards MQTT Broker and Influx Database in order to allow the follow up of the production using Grafana.

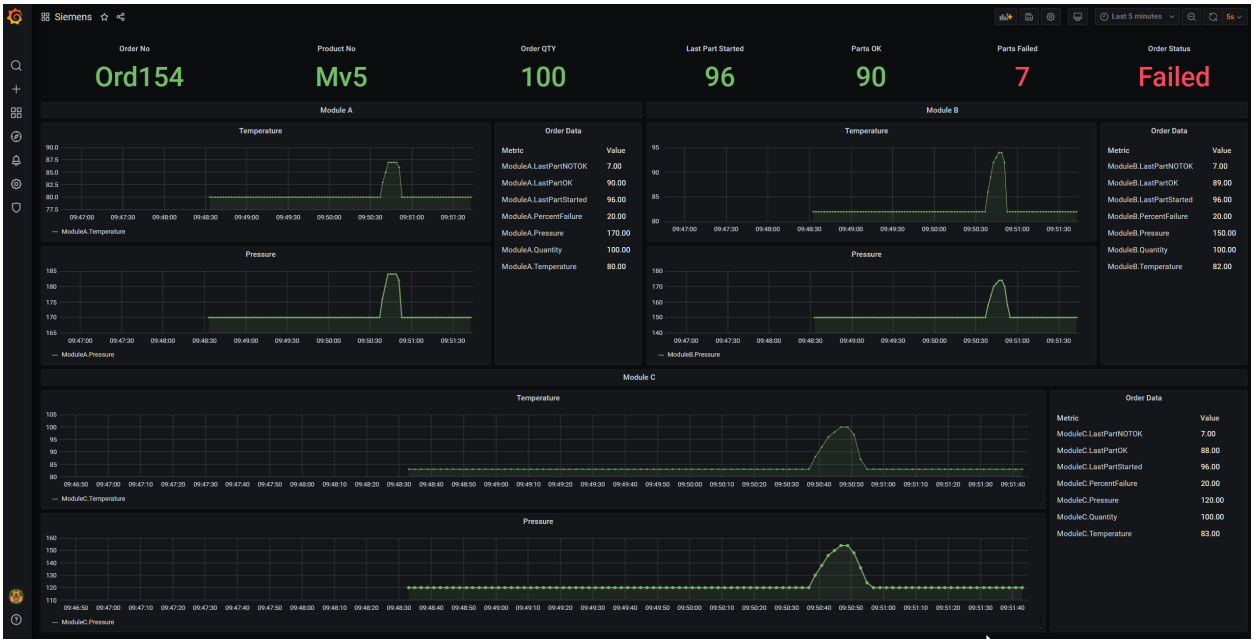
5.2.3 Start Failure

The Equipment Simulator is capable to simulate production of failed parts by accessing the option “StartFailure” from the Node-RED dashboard. Add a value (percentage of failed parts to be generated by the Equipment Simulator) to the input box **(4)** and click the “StartFailure” button **(5)**.





All the data provided by the Equipment Simulator is stored using the SMART**UNIFIER** in the Influx database and the overall process can be viewed on the Grafana dashboard.



5.3 Instance Setup

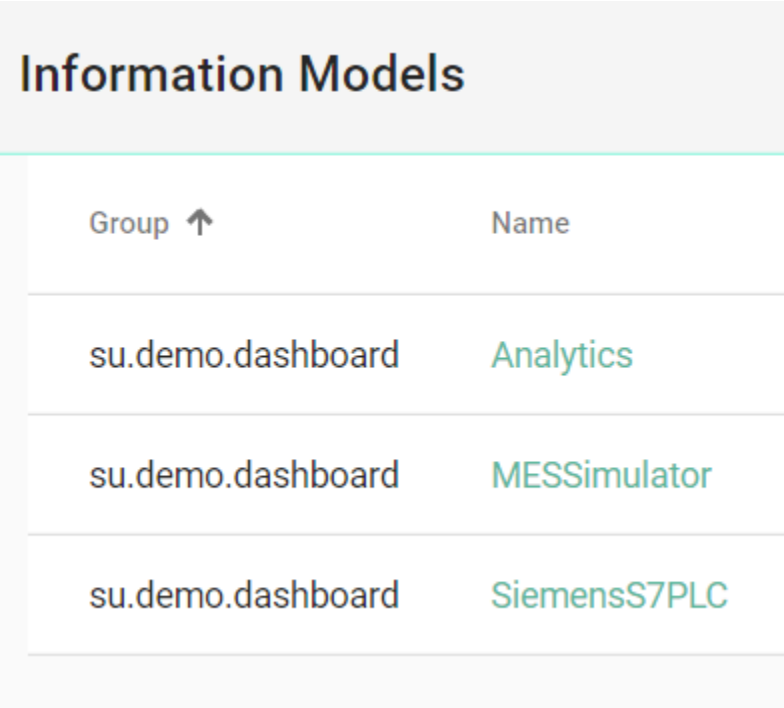
A SMART**UNIFIER** Instance is a dynamically created application that can be deployed to any suitable IT resource (e.g., Equipment PC, Server, Cloud), and which provides the connectivity functionality configured. Therefore, a SMART**UNIFIER** Instance uses one or multiple Mappings, selected Communication Channels and Information Models.

5.3.1 Information Models

Within the SMART**UNIFIER** an Information Model describes the communication related data that is available for a device or IT system. One device or one IT system therefore is represented by one Information Model.

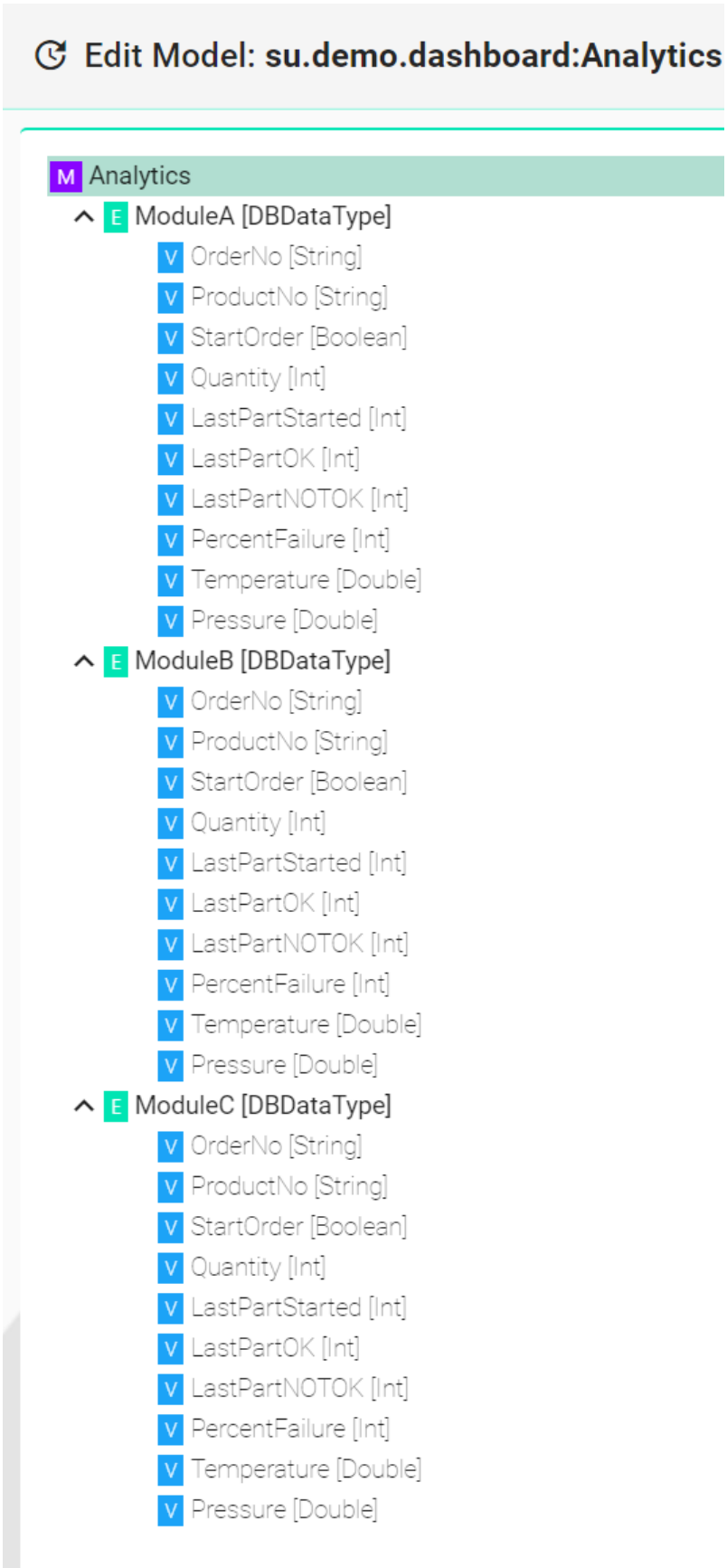
The Information Model perspective lists the information models currently configured within the SMART**UNIFIER** Manager:

1. **Analytics**
2. **MES Simulator**
3. **Siemens S7PLC**



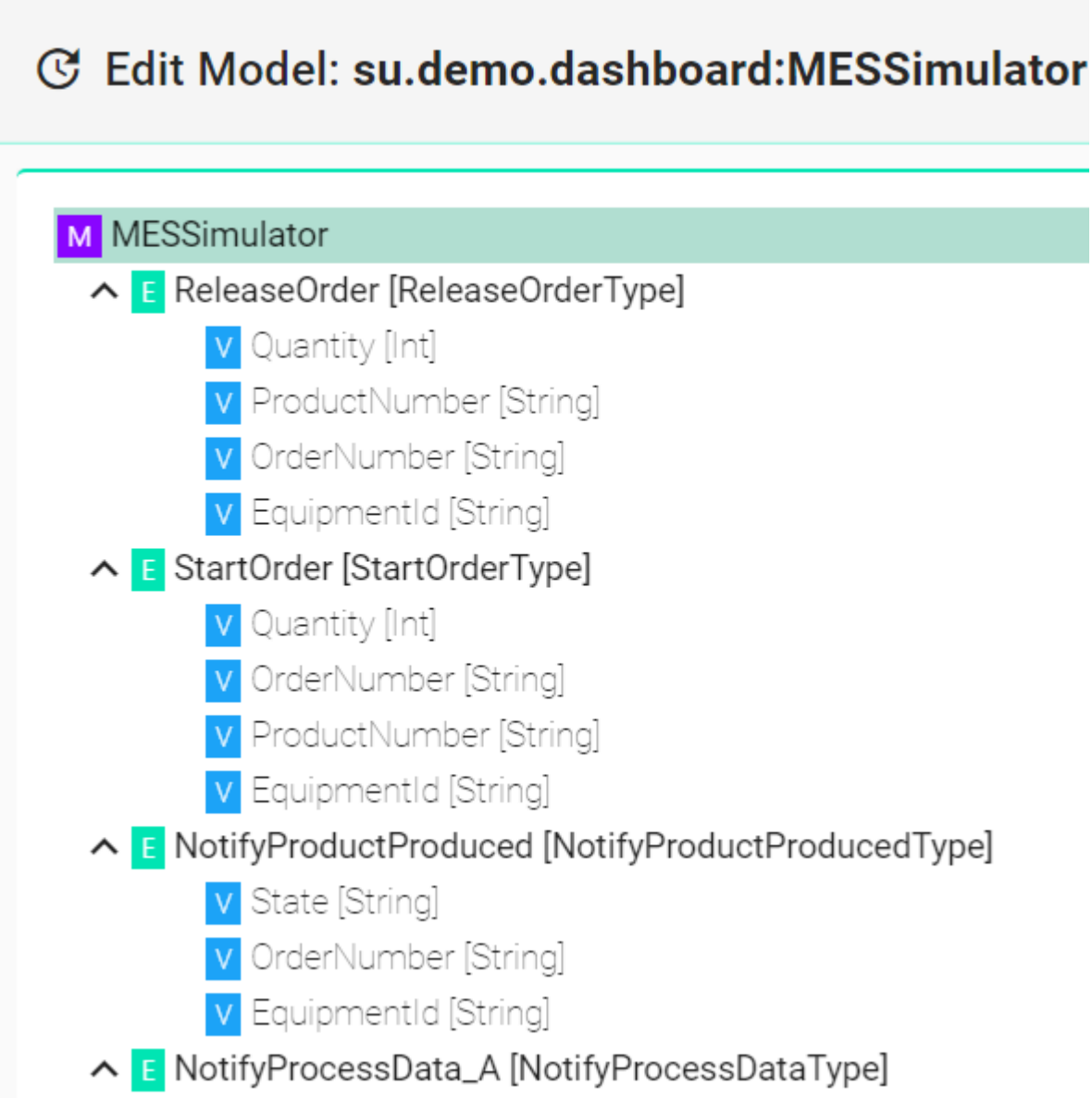
1. **Analytics**

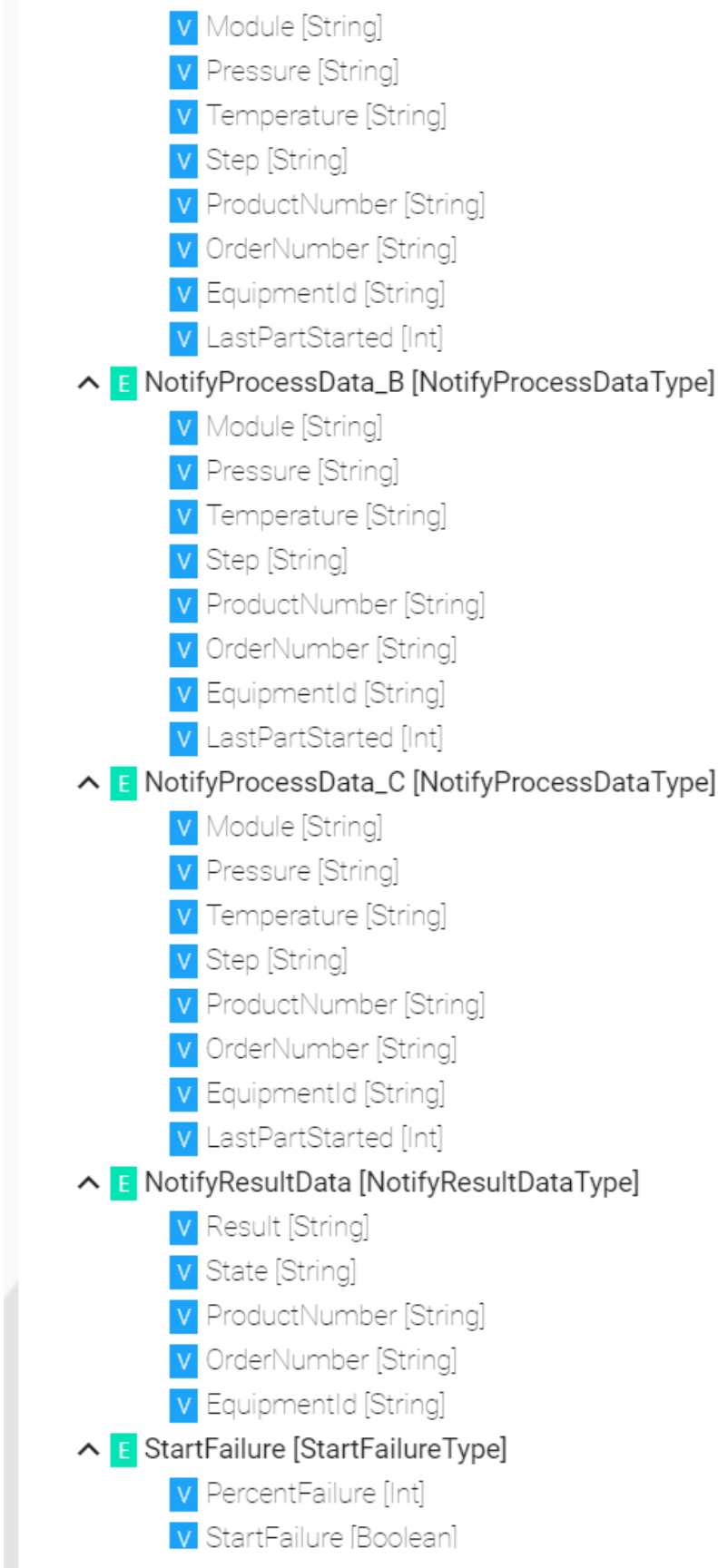
The Analytics represents the Influx database Information Model that stores data from the PLC of the production equipment (Equipment Simulator). The data provided by the Equipment Simulator (PLC) is stored using the SMART**UNIFIER** in the Influx database and the overall process can be viewed on the Grafana dashboard. As seen below, the data is structured for each production module.


Edit Model: su.demo.dashboard:Analytics
Analytics
ModuleA [DBDataType]
OrderNo [String]
ProductNo [String]
StartOrder [Boolean]
Quantity [Int]
LastPartStarted [Int]
LastPartOK [Int]
LastPartNOTOK [Int]
PercentFailure [Int]
Temperature [Double]
Pressure [Double]
ModuleB [DBDataType]
OrderNo [String]
ProductNo [String]
StartOrder [Boolean]
Quantity [Int]
LastPartStarted [Int]
LastPartOK [Int]
LastPartNOTOK [Int]
PercentFailure [Int]
Temperature [Double]
Pressure [Double]
ModuleC [DBDataType]
OrderNo [String]
ProductNo [String]
StartOrder [Boolean]
Quantity [Int]
LastPartStarted [Int]
LastPartOK [Int]
LastPartNOTOK [Int]
PercentFailure [Int]
Temperature [Double]
Pressure [Double]

1. **MES Simulator**

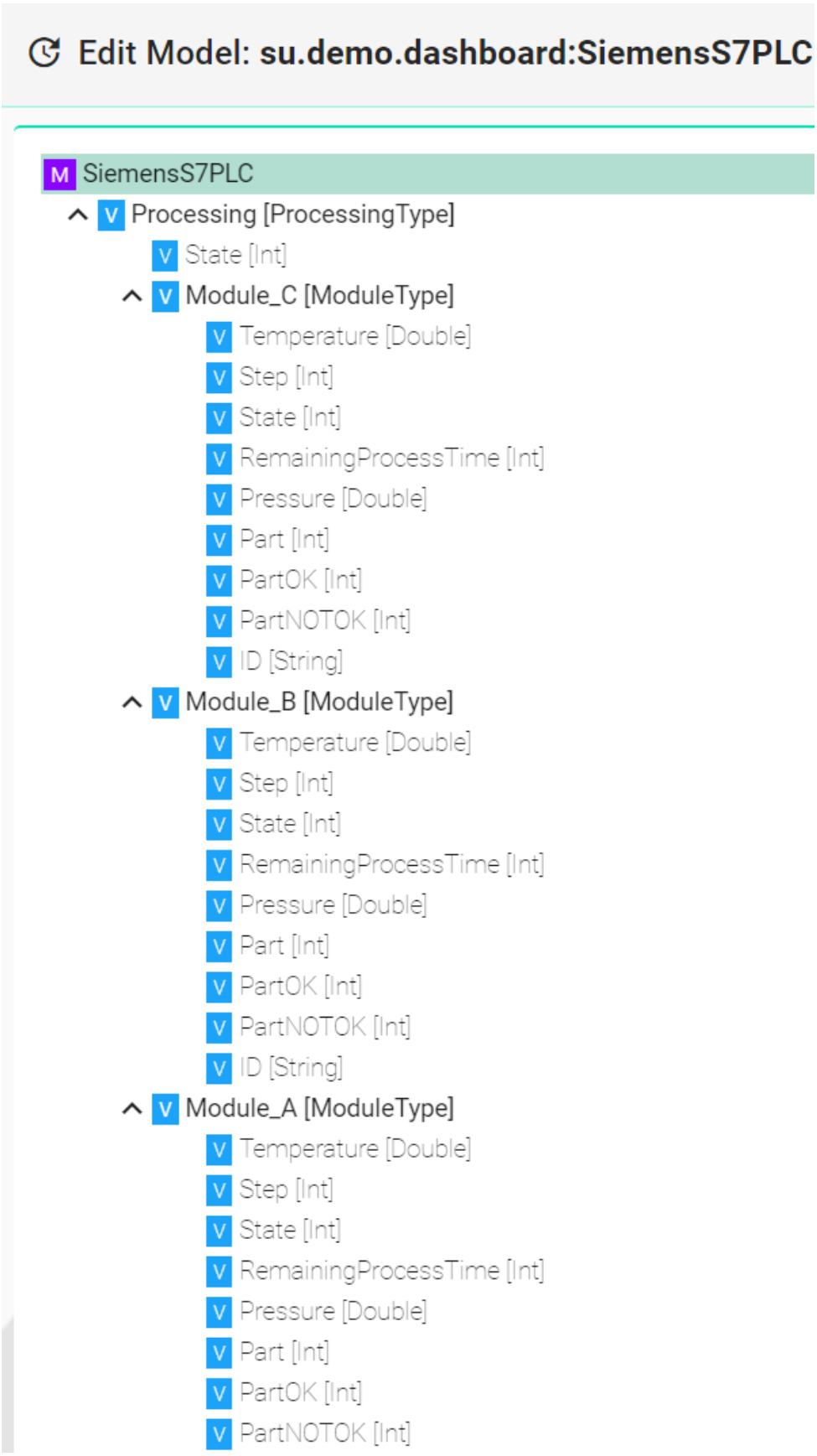
The MES Simulator Information Model represents the MES structure. It maps the production process flow, as seen below.

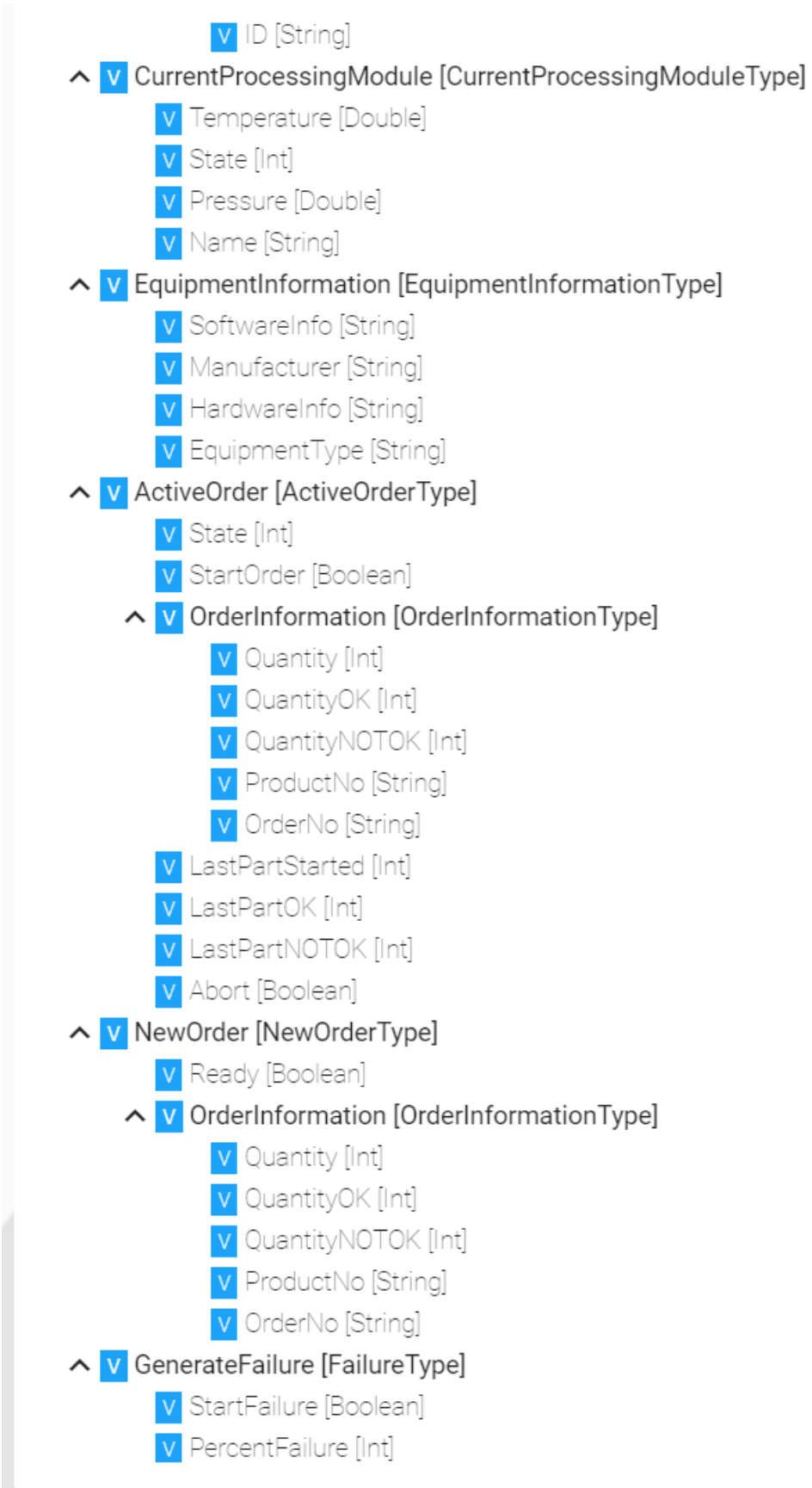



Module [String]
Pressure [String]
Temperature [String]
Step [String]
ProductNumber [String]
OrderNumber [String]
EquipmentId [String]
LastPartStarted [Int]
NotifyProcessData_B [NotifyProcessDataType]
Module [String]
Pressure [String]
Temperature [String]
Step [String]
ProductNumber [String]
OrderNumber [String]
EquipmentId [String]
LastPartStarted [Int]
NotifyProcessData_C [NotifyProcessDataType]
Module [String]
Pressure [String]
Temperature [String]
Step [String]
ProductNumber [String]
OrderNumber [String]
EquipmentId [String]
LastPartStarted [Int]
NotifyResultData [NotifyResultDataType]
Result [String]
State [String]
ProductNumber [String]
OrderNumber [String]
EquipmentId [String]
StartFailure [StartFailureType]
PercentFailure [Int]
StartFailure [Boolean]

1. **Siemens S7 PLC**

The Siemens S7 PLC Information Model represents the PLC blocks structure. As seen below, the Model contains the input and output data for each production module.


Edit Model: su.demo.dashboard:SiemensS7PLC
SiemensS7PLC
Processing [ProcessingType]
State [Int]
Module_C [ModuleType]
Temperature [Double]
Step [Int]
State [Int]
RemainingProcessTime [Int]
Pressure [Double]
Part [Int]
PartOK [Int]
PartNOTOK [Int]
ID [String]
Module_B [ModuleType]
Temperature [Double]
Step [Int]
State [Int]
RemainingProcessTime [Int]
Pressure [Double]
Part [Int]
PartOK [Int]
PartNOTOK [Int]
ID [String]
Module_A [ModuleType]
Temperature [Double]
Step [Int]
State [Int]
RemainingProcessTime [Int]
Pressure [Double]
Part [Int]
PartOK [Int]
PartNOTOK [Int]

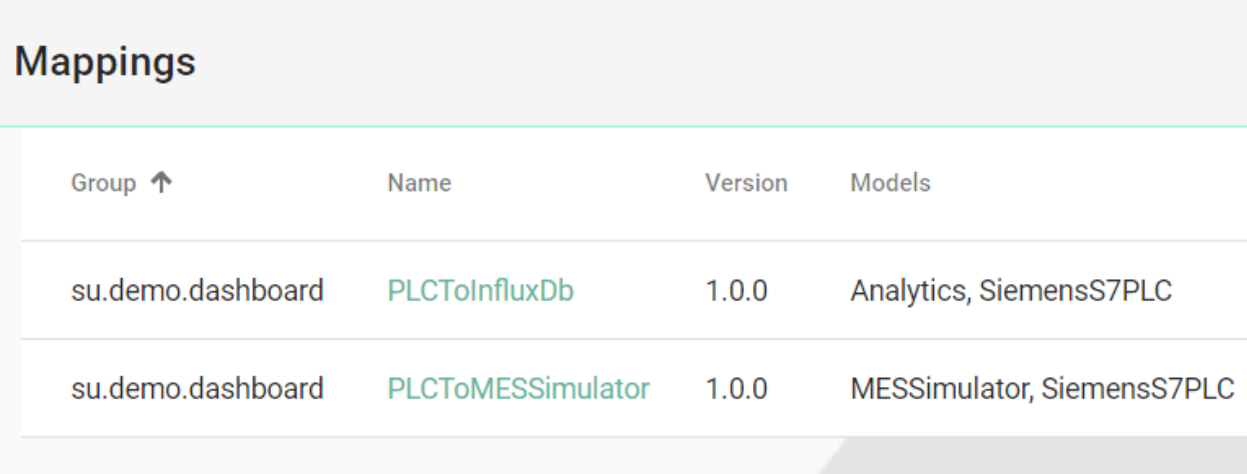

ID [String]
CurrentProcessingModule [CurrentProcessingModuleType]
Temperature [Double]
State [Int]
Pressure [Double]
Name [String]
EquipmentInformation [EquipmentInformationType]
SoftwareInfo [String]
Manufacturer [String]
HardwareInfo [String]
EquipmentType [String]
ActiveOrder [ActiveOrderType]
State [Int]
StartOrder [Boolean]
OrderInformation [OrderInformationType]
Quantity [Int]
QuantityOK [Int]
QuantityNOTOK [Int]
ProductNo [String]
OrderNo [String]
LastPartStarted [Int]
LastPartOK [Int]
LastPartNOTOK [Int]
Abort [Boolean]
NewOrder [NewOrderType]
Ready [Boolean]
OrderInformation [OrderInformationType]
Quantity [Int]
QuantityOK [Int]
QuantityNOTOK [Int]
ProductNo [String]
OrderNo [String]
GenerateFailure [FailureType]
StartFailure [Boolean]
PercentFailure [Int]

5.3.2 Mappings

The Mapping represents the SMART**UNIFIER** component that defines when and how to exchange/transform data between two or multiple Information Models. In other words, it is acting as a translator between the different Information Models. One Mapping consists of one or multiple Rules. A Rule contains a Trigger, which defines when the exchange/transformation takes place, and a list of actions that are defining how the exchange/transformation is done.

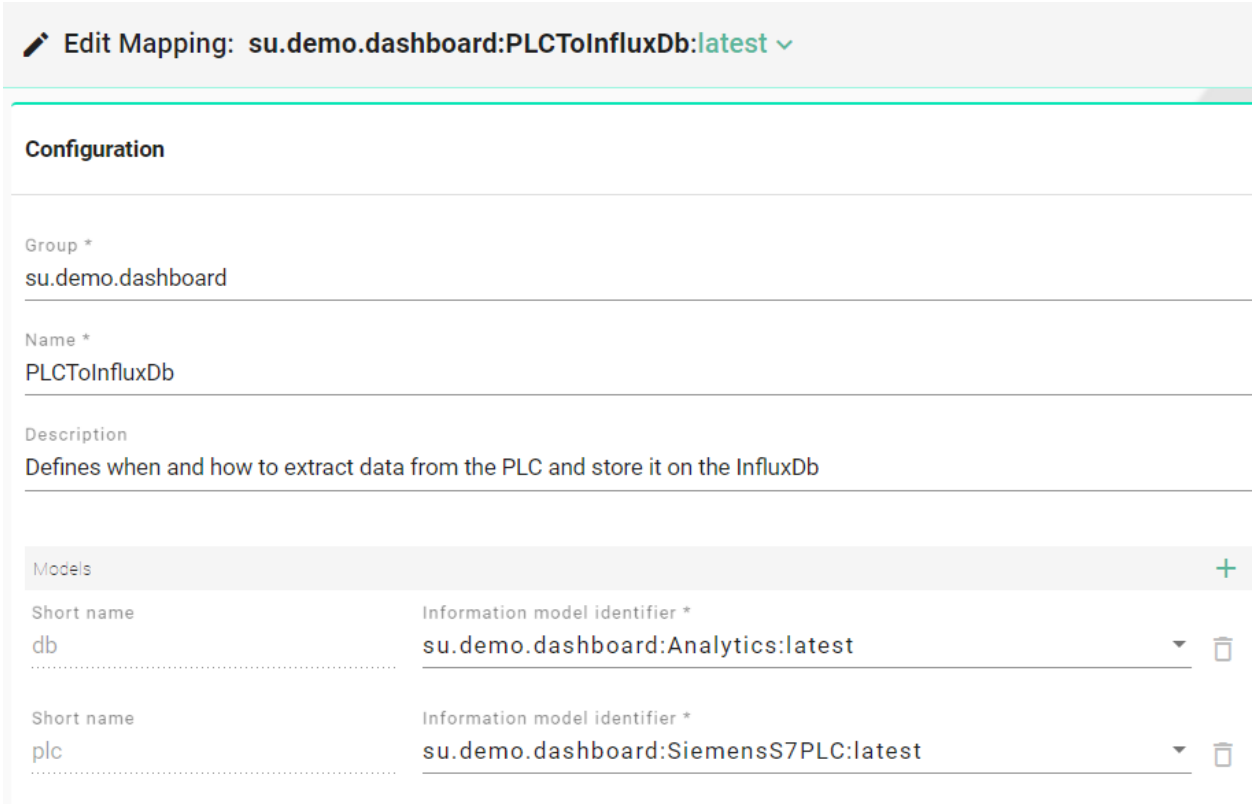
The Mapping perspective lists the Mappings currently configured within the SMART**UNIFIER** Manager:

1. **PLC to InfluxDb**
2. **PLC to MES Simulator**



1. **PLC to InfluxDb**

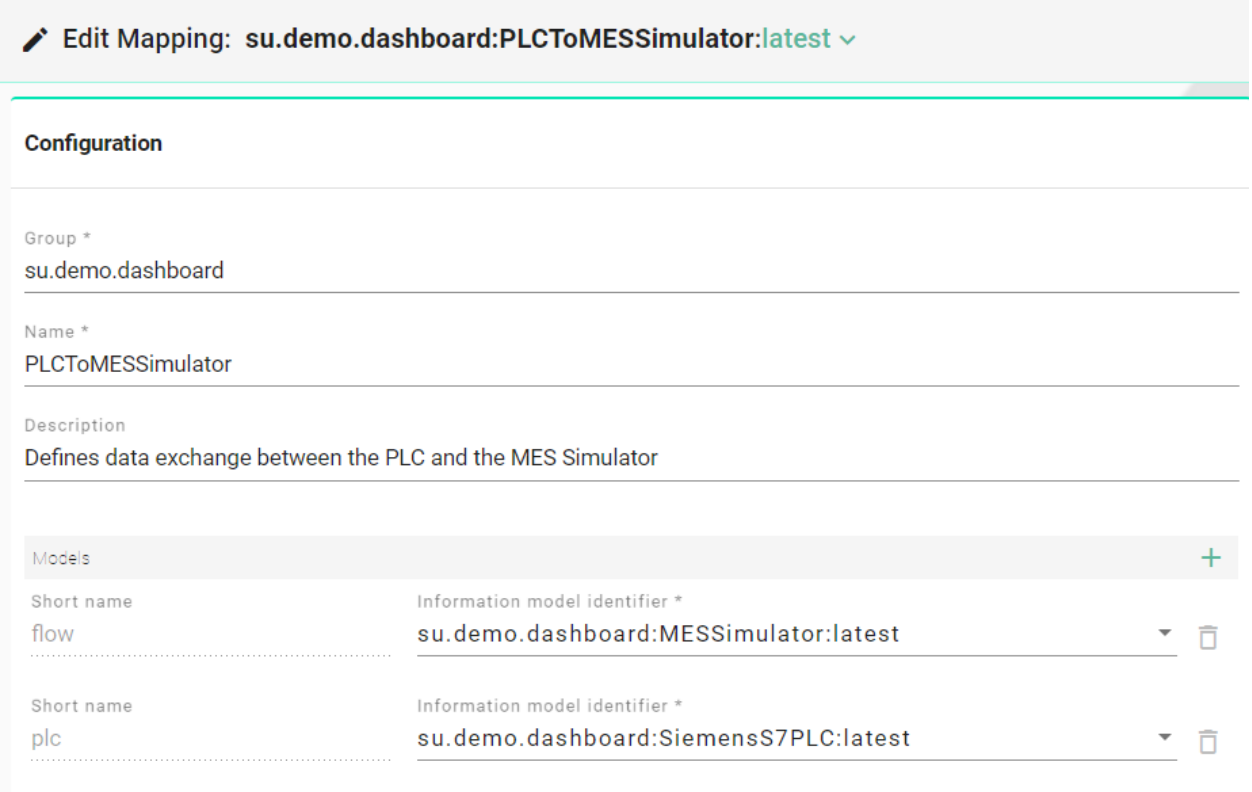
This Mapping defines when and how to extract data from the PLC of the production equipment (Equipment Simulator) and store it on the InfluxDb.


Edit Mapping: su.demo.dashboard:PLCToInfluxDb:latest
Configuration
Group *
su.demo.dashboard
Name *
PLCToInfluxDb
Description
Defines when and how to extract data from the PLC and store it on the InfluxDb
Models
Short name
db
Information model identifier *
su.demo.dashboard:Analytics:latest
Short name
plc
Information model identifier *
su.demo.dashboard:SiemensS7PLC:latest

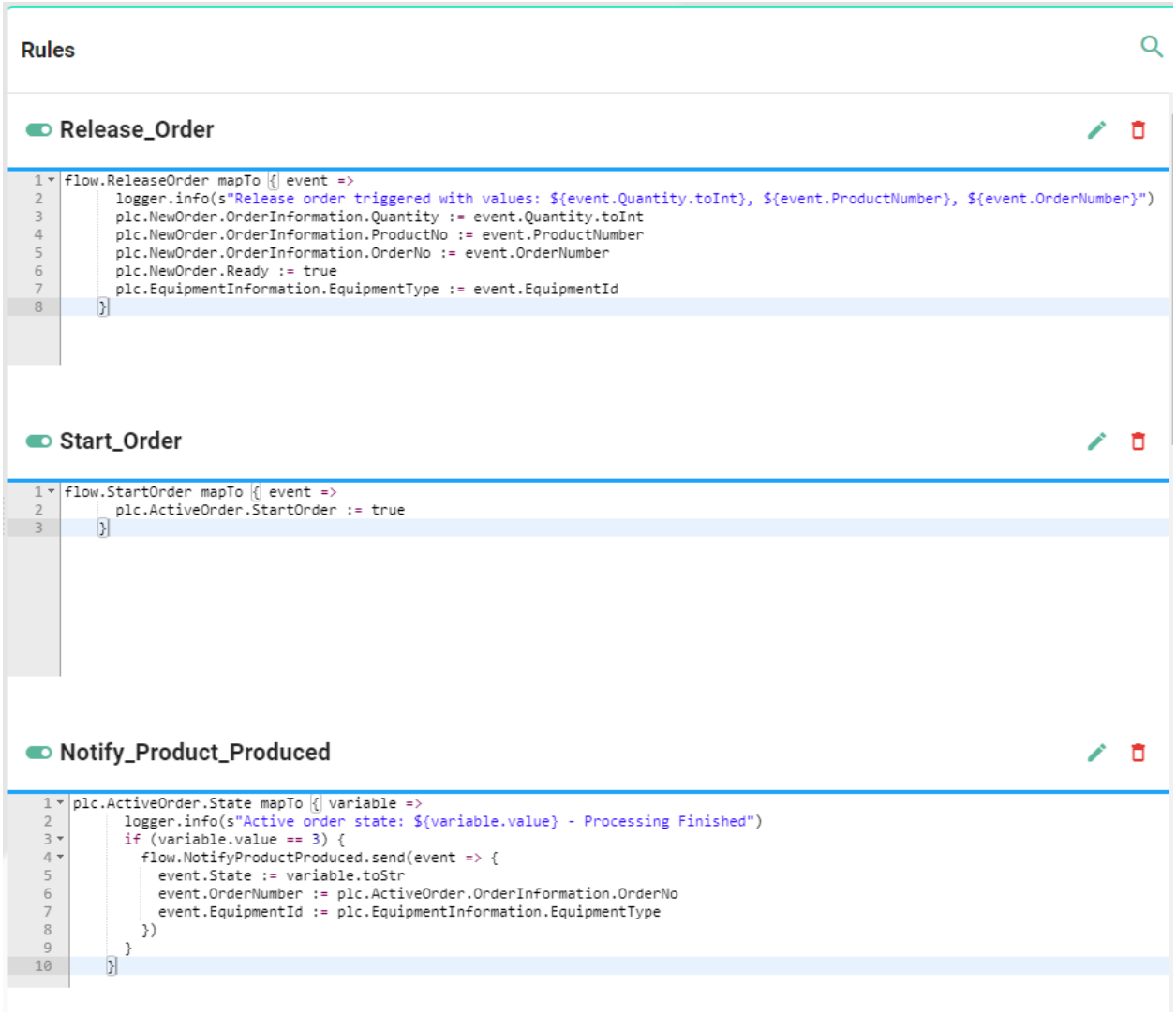
As seen above, for each production module, the rules created with Scala code lines define how the data exchange takes place.

1. **PLC to MES Simulator**

This Mapping defines data exchange between the PLC of the production equipment (Equipment Simulator) and the MES Simulator (Node-Red Dashboard).



As seen below, for each production process, the rules created with Scala code lines define how the data exchange takes place.


Rules
Release_Order
flow.ReleaseOrder mapTo { event =>
logger.info(s"Release order triggered with values: ${event.Quantity.toInt}, ${event.ProductNumber}, ${event.OrderNumber}")
plc.NewOrder.OrderInformation.Quantity := event.Quantity.toInt
plc.NewOrder.OrderInformation.ProductNo := event.ProductNumber
plc.NewOrder.OrderInformation.OrderNo := event.OrderNumber
plc.NewOrder.Ready := true
plc.EquipmentInformation.EquipmentType := event.EquipmentId
}
Start_Order
flow.StartOrder mapTo { event =>
plc.ActiveOrder.StartOrder := true
}
Notify_Product_Produced
plc.ActiveOrder.State mapTo { variable =>
logger.info(s"Active order state: ${variable.value} - Processing Finished")
if (variable.value == 3) {
flow.NotifyProductProduced.send(event => {
event.State := variable.toStr
event.OrderNumber := plc.ActiveOrder.OrderInformation.OrderNo
event.EquipmentId := plc.EquipmentInformation.EquipmentType
})
}
}

Notify_Result_Data

```
      logger.info(s"Active order state: ${variable.value} - Processing Finished")
      if (variable.value == 3) {
        flow.NotifyResultData.send(event => {
          event.Result := "t.b.n"
          event.State := variable.toStr
          event.ProductNumber := plc.ActiveOrder.OrderInformation.ProductNo
          event.OrderNumber := plc.ActiveOrder.OrderInformation.OrderNo
          event.EquipmentId := "t.b.n"
        })
      }
    }
```

Notify_Process_Data_A

```
plc.Processing.Module_A.Part mapTo { variable =>
      logger.info(s"Active order state: ${variable.value} - Processing")
      //    if (variable.value == 2) {
      flow.NotifyProcessData_A.send(event => {
        event.Module := "Module A"
        event.Pressure := plc.Processing.Module_A.Pressure.toStr
        event.Temperature := plc.Processing.Module_A.Temperature.toStr
        event.Step := plc.Processing.Module_A.Step.toStr
        event.ProductNumber := plc.ActiveOrder.OrderInformation.ProductNo
        event.OrderNumber := plc.ActiveOrder.OrderInformation.OrderNo
        event.LastPartStarted := plc.Processing.Module_A.Part
```

Notify_Process_Data_B

```
plc.Processing.Module_B.Part mapTo { variable =>
      logger.info(s"Active order state: ${variable.value} - Processing")

      flow.NotifyProcessData_B.send(event => {
        event.Module := "Module B"
        event.Pressure := plc.Processing.Module_B.Pressure.toStr
        event.Temperature := plc.Processing.Module_B.Temperature.toStr
        event.Step := plc.Processing.Module_B.Step.toStr
        event.ProductNumber := plc.ActiveOrder.OrderInformation.ProductNo
        event.OrderNumber := plc.ActiveOrder.OrderInformation.OrderNo
        event.LastPartStarted := plc.Processing.Module_B.Part
```

Notify_Process_Data_C

```
        event.Module := "Module C"
        event.Pressure := plc.Processing.Module_C.Pressure.toStr
        event.Temperature := plc.Processing.Module_C.Temperature.toStr
        event.Step := plc.Processing.Module_C.Step.toStr
        event.ProductNumber := plc.ActiveOrder.OrderInformation.ProductNo
        event.OrderNumber := plc.ActiveOrder.OrderInformation.OrderNo
        event.LastPartStarted := plc.Processing.Module_C.Part
        event.EquipmentId := plc.EquipmentInformation.EquipmentType
      })
      // }
    }
```

Start_Failure

```
flow.StartFailure mapTo { event =>
      logger.info(s"Trigger production of failed parts in a percentage of ${event.PercentFailure.value}")
      //  if (event.PercentFailure.value > 0) {

      plc.GenerateFailure.PercentFailure := event.PercentFailure
      plc.GenerateFailure.StartFailure := true
      //  }
    }
```