APPENDIX A

AI-CAMPUS COURSE: LERNFABRIK
LEARNING FACTORY 4.0 – CONTROL, MONITORING AND NN-MODEL

MORITZ HOHNEL

and

MATTIS TOM RITTER

Faculty of Mechanical and Manufacturing Engineering
Universiti Tun Hussein Onn Malaysia

JULY 2023

APPENDIX A

CONTENTS

APPENDIX A

APPENDIX A

APPENDIX A

APPENDIX A

CHAPTER 0 Introduction

Welcome to the course **Learning Factory 4.0 - Control, Monitoring and NN-Model**.

The fischertechnik learning factory is a learning factory that consists of the fischertechnik elements and is used to playfully simulate a digitized factory. The simulation maps the processes of ordering, production, delivery and storage in digitized process steps.

In the course, you will learn how to handle Internet of Things systems using the MQTT and OPC/UA protocols. You will learn how to handle data ingestion (ETL) routes with Nifi using exercise tasks to extract data from the production line, convert its form and write it to a database. Furthermore, you will perform data analysis using Python. At the end, we will take a look at a neural network for learning factory status detection.

APPENDIX A

CHAPTER 1 Module 1

Welcome to module 1.

In module 1 we describe the communication using MQTT and OPC/UA between the individual components of the learning factory and the university cloud. Your task is to connect to the individual systems via the university cloud.

Learning objectives

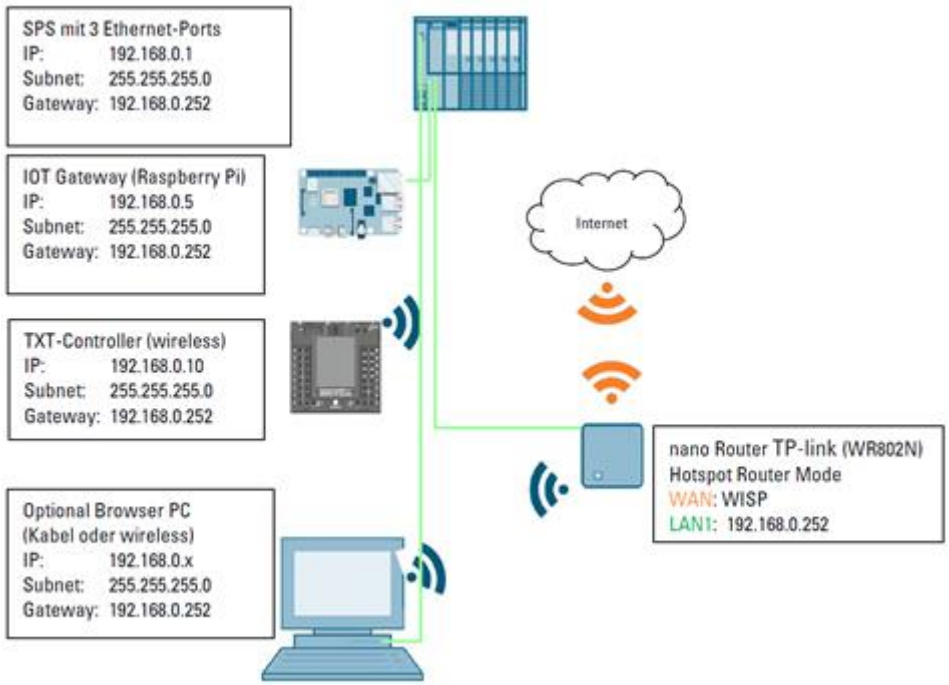
- Getting to know the structure of the learning factory.
- Create a communication to the individual systems of the learning factory.

MQTT is a system that can be used to transmit data to a server, which is called a broker here. A publisher writes the data and a subscriber reads the data for further processing.

OPC/UA is a protocol for monitoring PLC controllers.

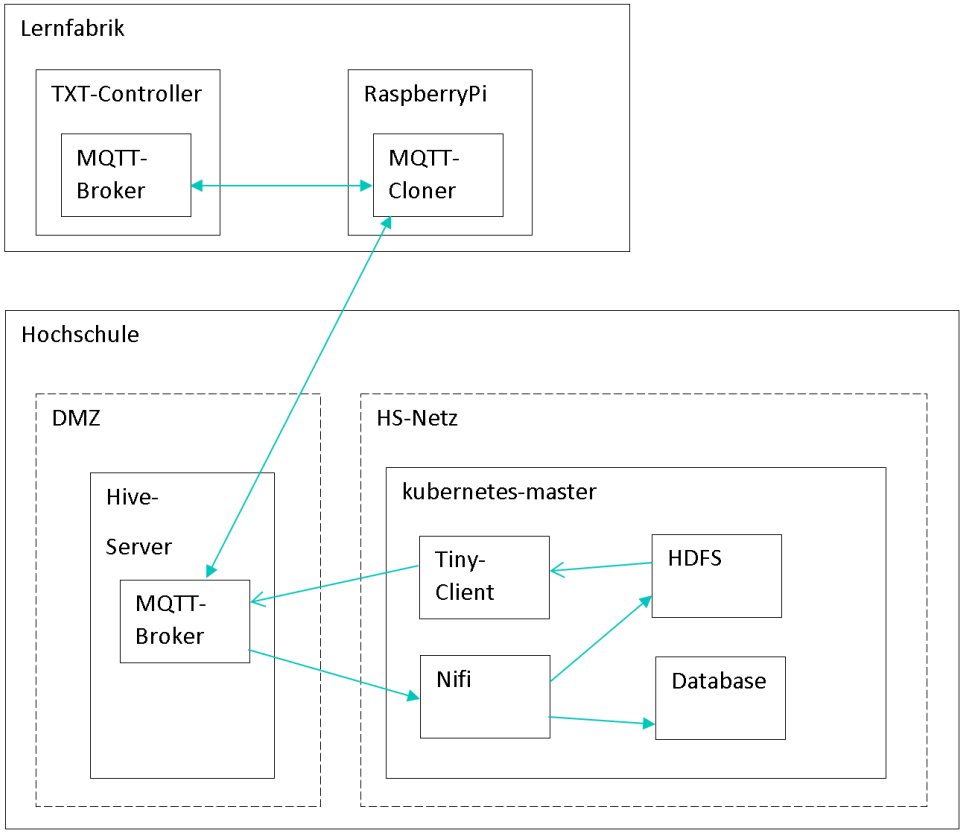
1.1 Communication structure of the learning factory

The figure describes the communication between the individual components of the learning factory. The components used are a PLC, a Raspberry PI and a controller. The Raspberry PI serves as a gateway to the university cloud. The PI runs a Python script to execute orders and send monitoring data to the university cloud. An MQTT broker running on the controller is used for data monitoring. The MQTT broker runs on the controller.



1.2 University communication structure

The figure shows the individual components in the HochschulCloud. A Hive and an MQTT broker are available in the demilitarized zone (DMZ) within the university cloud. The MQTT messages are sent from the learning factory to the MQTT server in the DMZ, where they are further processed using Apache Nifi as an ETL system and written to a relational database or Apache Hadoop. The MQTT cloner copies the MQTT communication that takes place on the controller to the university's MQTT broker in the DMZ. The cloner is a Python script that connects to both brokers and copies the messages



Apache Hadoop is a distributed file system.

Using Hive, files can be accessed and analyses can be created using SQL. SQL is a standard query language for database systems.

Apache Nifi is a data ingestion system, often called an ETL system.

ETL stands for Extract, Transfer, Load. In the Extract phase, data is loaded from a system, Transfer is used to modify the data, and Load is used to store the data back into a target system. In this system the data is read from the learning factory, transformed into the format of the target database and written into the database.

1.3 Exercise

Connect to the following systems to learn the system structure described in modules 1.1. and 1.2. You will perform exercises with these systems in Modules 2 through 5.

- Nifi
- Redshift via Python
- University Cloud
- Node-RED (OPC communication)

APPENDIX A

Note: If you have connection problems, please use a different browser.

System access

- Nifi

  You can access Nifi via the address hivemqserver.feste-ip.net:9443/nifi/.

  The access data are (user: lernfabrik / password: L5nf1br|k)

- Redshift via Python

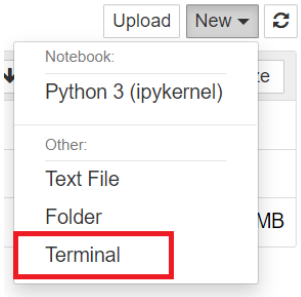
  You can run the Python scripts via university Jupyter notebooks in our Jupyter hub. To do this, access the address hivemqserver.feste-ip.net:9001.

  Redshift is a relational database in the AWS Cloud. In a relational database, data is stored in the form of tables.

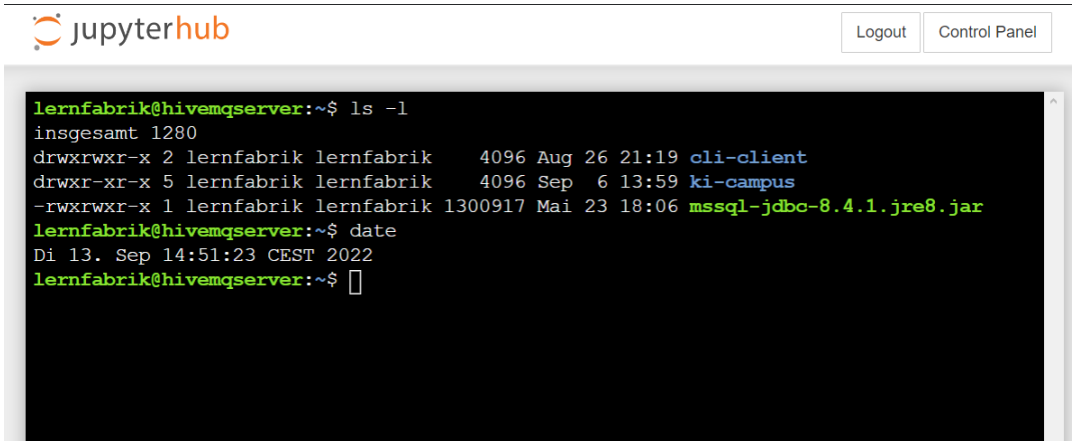
  The access data are (user: lernfabrik / password: L5nf1br|k)

- University cloud

  In order to use the access to the university server, log in to the JupyterHub server of the HS and start a terminal window. To do this, click on New in the upper right corner after logging in. And then click on Terminal.



A fully functional terminal window will open with which you can operate the CLI client, for example.



APPENDIX A

The scripts for the exercises can be found in the directory /home/lernfabrik/ki-campus. You do not have to execute the scripts yet.

- Node-Red (OPC communication)

  Node-RED is used to realize the OPC communication to the learning factory. Node-Red can be reached at the following address hivemqserver.feste-ip.net:1880.

  The access data are (user: lernfabrik / password: L5nf1br|k)

APPENDIX A

CHAPTER 2 Module 2

Welcome to module 2.

In this module we will learn about the Apache Nifi ETL system to read data from an MQTT broker, convert its form to SQL and write it to a database.

Learning Objectives

- To get to know Apache Nifi in order to read data from an MQTT broker, convert its form to SQL and write to a database.

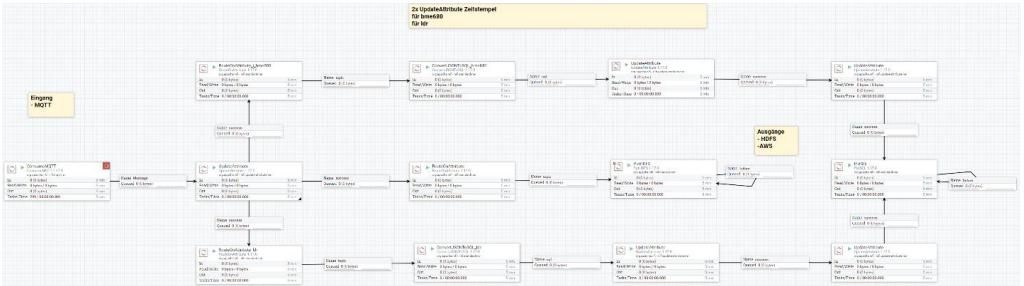
2.1 Flow Chart

Nifi is used to read messages from the learning factory and write them to a relational database.

Apache Nifi is a data ingestion system, often called an ETL system. ETL stands for Extract, Transfer, Load. In the Extract phase, data is loaded from a system, Transfer is used to modify the data, and Load is used to store the data back into a target system. In this system, the data is read from the learning factory, transformed into the format of the target database, and written to the database. The Nifi system is available in the university cloud at the URL hivemqserver.feste-ip.net:9443/nifi/. To the right of the Nifi logo are processors that can be used for a variety of data transformations.

Nifi is used to read the cloned messages and split them accordingly. Most of the messages are stored on HDFS. The messages that contain sensor values are written to a database. HDFS is a distributed file system (Hadoop Distributed Filesystem). XML- JSON or CSV files can be stored in HDFS.

The entire flow in Nifi is shown in the figure below:



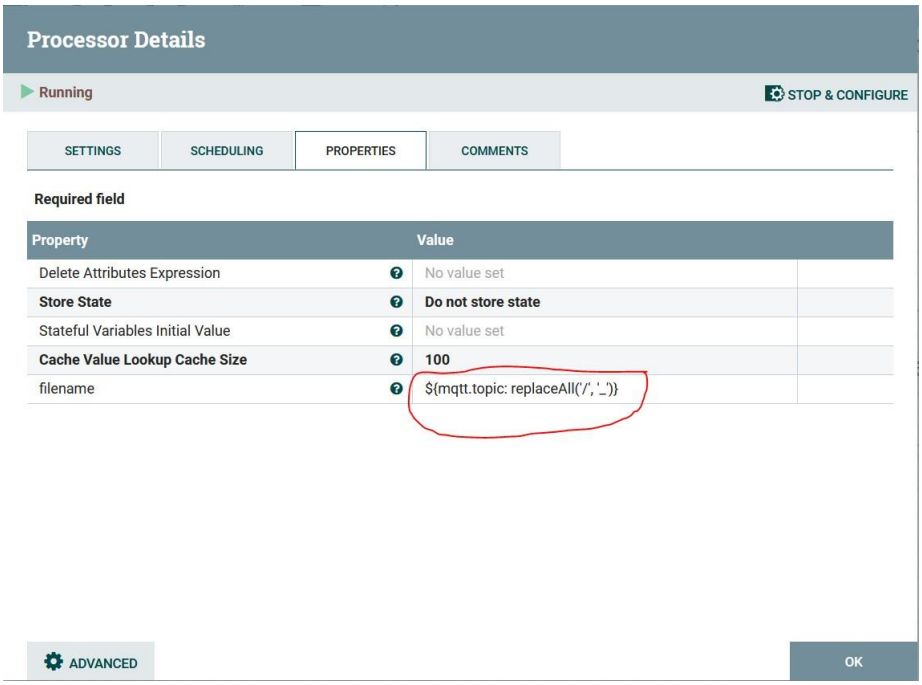
2.2 ConsumeMQTT

The ConsumeMQTT processor is used to read data from an MQTT server. Message Queue Telemetry Transport (MQTT) is an open message protocol for machine-to-machine (M2M) communication. It enables the transmission of telemetry data in the form of messages between devices, despite high delays or limited networks. Corresponding devices range from sensors and actuators, cell phones, embedded systems in vehicles or laptops to fully developed computers. The protocol was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Cirrus Link Solutions. In Nifi, the processor ConsumeMQTT is used as an MQTT subscriber that connects to an MQTT broker (server) under a specific topic.

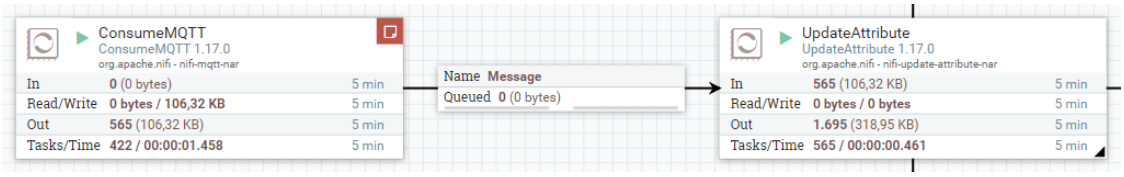
In Nifi there is the processor ConsumeMQTT where the following parameters have to be set:

Property	**Value**
Broker URI	tcp://141.87.109.227:1883
Client ID	nifi_lernfabrik_tls_client
Topic Filter	#

The data is changed using the UpdateAttribute process to swap the "/" in Topics for "_". The background is that otherwise the data is stored in HDFS directories. For this the character / is a hindrance. Thus e.g. from "i/ldr" -> "i_ldr" becomes. For this the following is entered in the property "filename":



View Chart.



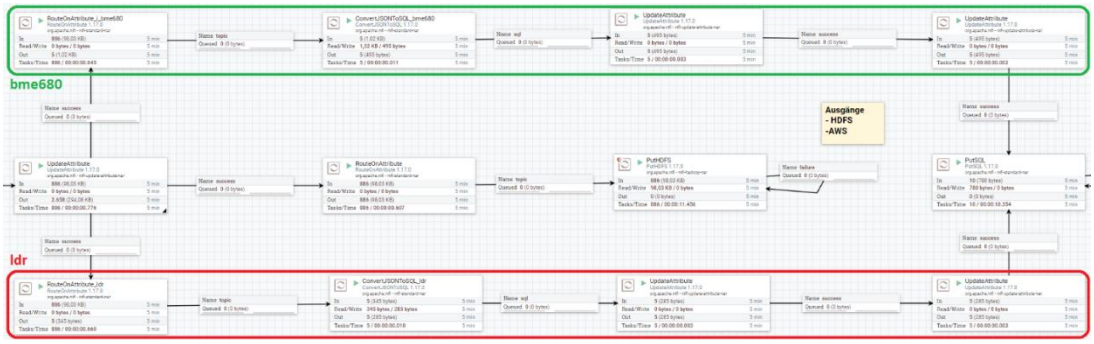
2.3 UpdateAttribute / ConvertJSONToSQL

After that the branch splits. The division is based on the data values. A distinction is made between environmental sensor values (type bme680), brightness sensor values (type ldr) and production data. Production data is written to HDFS, while the other values are written to a relational database, here AWS Redshift. AWS Redshift is a relational database in the Amazon Cloud (AWS).

Additionally, the relational data ldr and bme680 are split again to better distinguish the topics in the RouteOnAttribute process.

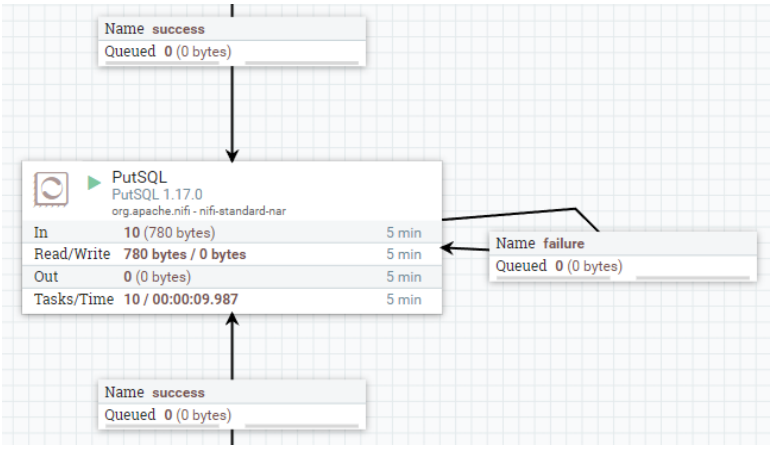
Therefore, the data is split by "i/bme680" and "i/ldr". The two branches are almost identical with the exception of the data processed in them. After converting the data to SQL using the ConvertJSONtoSQL process, two UpdateAttribute processes are still applied to it to adjust the timestamps so that AWS can store them. In the timestamp, the letters "Z" and "T" are removed using:

```
${sql.args.9.value: replace('Z', '')}
${sql.args.9.value: replace('T', '')}
```



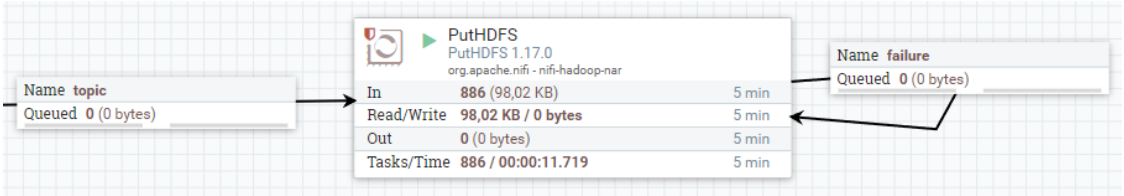
2.4 PutSQL

Finally, the data is written to the database using the PutSQL process.



2.5 PutHDFS

All other topics are written to HDFS using PutHDFS.



2.6 Exercises

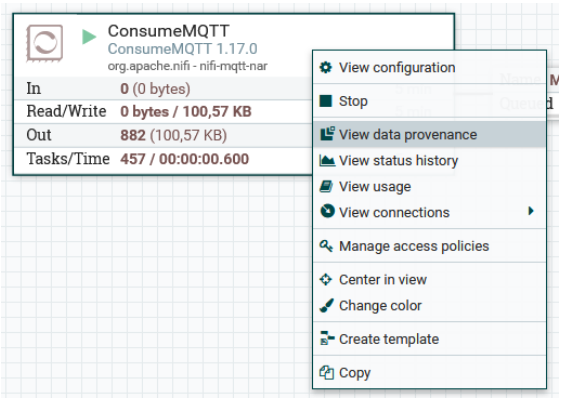
Connect to Apache Nifi at the following address https://hivemqserver.feste-ip.net:9443/nifi/. (user: lernfabrik / password: L5nf1br|k) and solve the following tasks:

a) Determine the address of the MQTT server in the university cloud (section 2.2).

b) In the MQTT processor, see how data is fetched cyclically from the facility in JSON format. Track the data by right-clicking on the "PutHDFS" or "PutSQL" processors and using "View data provenance" to view the contents.

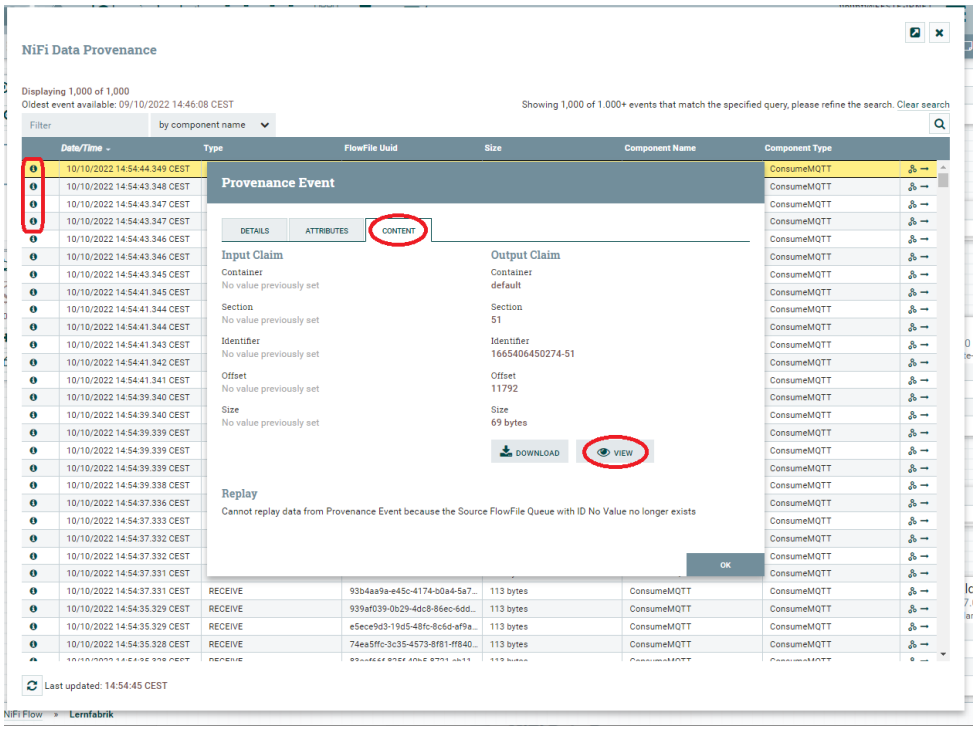
Use the following example as a guide:

Right-click on the ConsumeMQTT process. And then click on View data provenance.

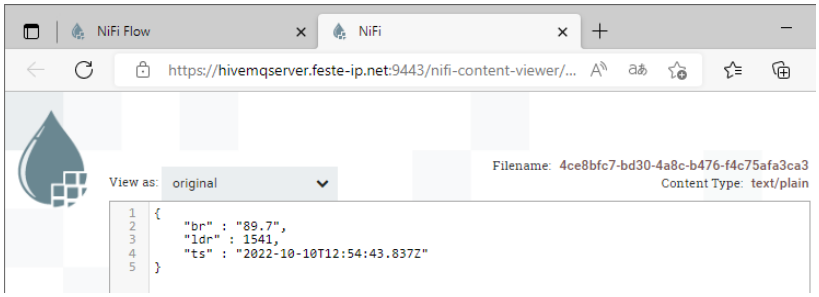


A list with all received MQTT messages will open. Click on one of the info icons on the far left of the table, Again a screen with three tabs opens, go to Content. And finally on View

APPENDIX A



The full MQTT message will we displayed.



APPENDIX A

CHAPTER 3 Module 3

Welcome to module 3.

In Module 3, we will learn a Python script that we can use to control the system such as placing orders, requesting the status of stations, and displaying camera images.

Learning Objectives

- Control a production plant using Python with MQTT.

Materials

- Python script to control the production line

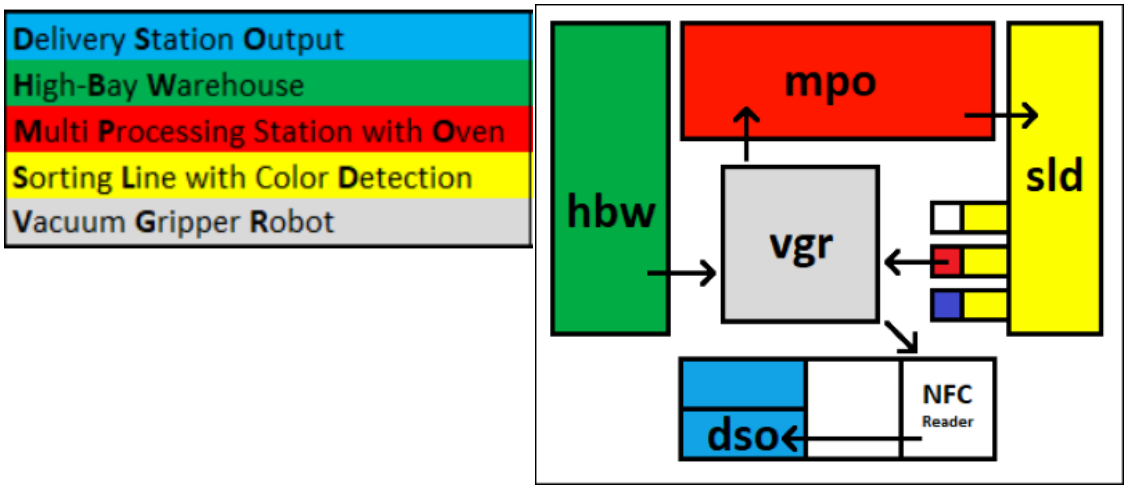
3.1 Ordering process

When an ordering process is triggered, the transport arm of the high-bay warehouse (HBW) moves to the storage system, picks up the ordered workpiece and places it in the input/output station. There it is conveyed to the pick-up position of the vacuum suction pad.

The vacuum gripper robot (VGR) picks up the workpiece from the input/output station of the high-bay warehouse (HBW) and places it on the slide of the furnace (MPO). There, the workpiece is pushed in, fired and moved out again. The transport carriage with vacuum suction pad then transports the workpiece to the "Mill" processing machine. There, the workpiece is placed on the rotary table. After the milling process, the workpiece is pneumatically pushed onto the conveyor belt (SLD).

On the conveyor belt, the workpiece passes through a colour recognition system. Depending on the colour detected, the workpiece is pneumatically pushed onto the corresponding material chute, where it is located in a pick-up area of the vacuum suction pad. From here, the workpiece is picked up by the vacuum gripper robot (VGR) and held for final labelling via the NFC reader, where the part is written with workpiece-relevant data such as order date, manufacturing or delivery data.

Finally, the workpiece is deposited in the output tray (DSO) of the input/output station.



3.2 storage process

Both the output tray (DSO) and the input tray (DSI) are mounted at an angle. After completion of the ordering process, the workpiece is placed in the output tray and slides directly into the input tray. Two light barriers are interrupted in the process. The first light barrier (DSO) signals the completion of the order. When the second light barrier (DSI) is interrupted, the infeed process is started automatically.

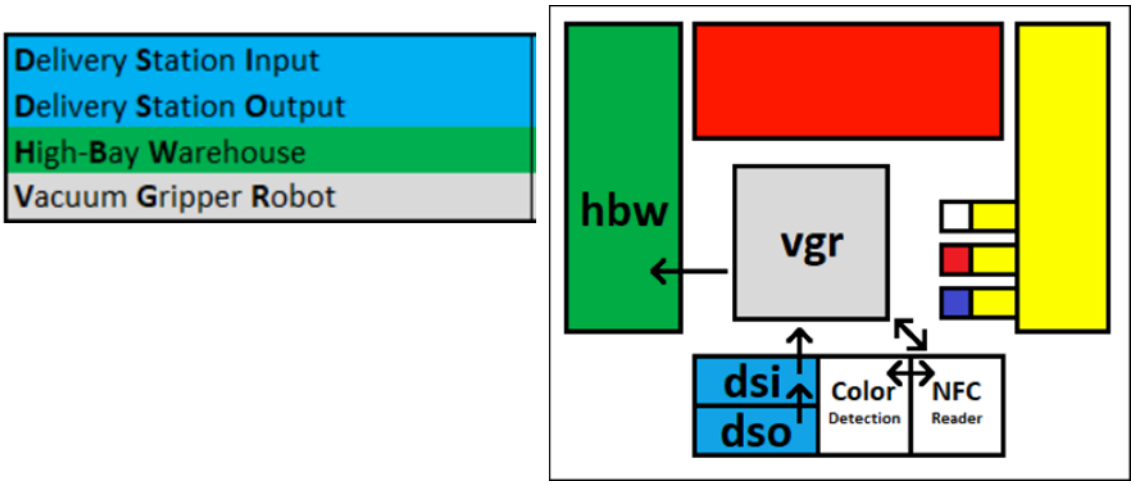
Here, the workpiece is picked up by the vacuum gripper robot and moved to the NFC reader, which deletes old data on the workpiece's NFC tag.

The workpiece is then moved to the colour sensor for colour detection.

Before storage, the determined colour, as well as other information such as delivery data, is written to the workpiece's NFC tag with the NFC reader.

Next, the transport arm of the high-bay warehouse picks up an empty container from the high-bay warehouse and transports it to the pickup position of the vacuum gripper robot.

Finally, the vacuum gripper robot places the workpiece in the waiting container, which is transported back to the high-bay warehouse by the transport arm of the high-bay warehouse and stored.



3.3 python script to control the factory

You can execute the script as described in module 1 with the command python3 lernfabrik.py -h. The individual options of the script are described in the following sections.

python3 lernfabrik.py -h

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -h
usage: lernfabrik.py [-h] [-o order] [-s state] [-t stock] [-c cam]

Benutzung python3 lernfabrik.py -XX z.B. lernfabrik.py -o RED z.B.
lernfabrik.py -s hbw z.B. lernfabrik.py -t 1

optional arguments:
  -h, --help            show this help message and exit
  -o {RED,BLUE,WHITE}, --order {RED,BLUE,WHITE}
                        Bestellung aufgeben.
  -s {hbw,vgr,mpo,sld,dsi,dso}, --state {hbw,vgr,mpo,sld,dsi,dso}
                        Stations Status abfragen.
  -t {0,1}, --stock {0,1}
                        Status des Hochregals abfragen
  -f, --follow          Sequentielles Abfragen aller Stationen.
  -c {0,1}, --cam {0,1}
                        Aktuelles Kamerabild anzeigen.
  -m movecam movecam, --movecam movecam movecam
                        Kamerasteuerung. Richtung: ['up', 'down',
'right',
                        'left']; Winkel: [2, 5, 10, 20]
```

3.3.1 Source code

The function parse_args() is a helper function to evaluate the passed arguments.

```
# Argument verwaltung
def parse_args():
    parser = argparse.ArgumentParser(
        description='Benutzung python3 lernfabrik.py -XX \n' \
                    'z.B. lernfabrik.py -o RED \n' \
                    'z.B. lernfabrik.py -s hbw \n' \
                    'z.B. lernfabrik.py -t 1'
    )
    parser.add_argument(
        '-o', '--order',
        choices=colors,
        help='Bestellung aufgeben.',
    )
    parser.add_argument(
        '-s', '--state',
        choices=stations,
        help='Stations Status abfragen.',
    )
    parser.add_argument(
        '-t', '--stock',
        choices=modi,
        type=int,
        help='Status des Hochregals abfragen.',
    )
    parser.add_argument(
        '-f', '--follow',
        action='store_true',
        help='Sequentielles Abfragen aller Stati.',
    )
    parser.add_argument(
        '-c', '--cam',
        choices=modi,
        type=int,
        help='Aktuelles Kamerabild abfragen.',
    )
    parser.add_argument(
        '-m', '--movecam',
        metavar='movecam',
        nargs=2,
        help='Kamerasteuerung. Richtung: ' + str(directions) + ';
Winkel: ' + str(angle),
    )
# end function parse_args
```

3.4 3.3 Request the states of the respective stations

The command python3 lernfabrik.py -s vgr displays the current state of the vacuum gripper (Vacuum Gripper Robot). There are the two states active (true) and inactive (false).
The command python3 lernfabrik.py -s hbw displays the current state of the High-Bay Warehouse. There are the two states active (true) and inactive (false).
The python3 lernfabrik.py -s mpo command displays the current state of the Multi Processing Station. There are the two states active (true) and inactive (false).
The command python3 lernfabrik.py -s sld displays the current state of the Sorting Line. There are the two states active (true) and inactive (false).
A detailed explanation of the codes can be found in module 6.2.

python3 lernfabrik.py -s vgr

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -s vgr

Status Station:         vgr
Letzter Zeitstempel:    2021-09-28T08:00:56.911Z
Letzter Status:         False
Code:                   1
```

3.4.1 Source code

The function createState() shows the current state of a specific station

```
# Returns state of a station
# {"ts":"2021-09-
22T09:32:45.436Z","station":"sld","code":1,"description":"","active
":false,"target":""}
def createState(station):
   value = getJSONFromHDFSByKey('f_i_state_' + station)
   state_dict = json.loads(value)

   print(' ')
   print('Status Station: \t' + str(state_dict['station']))
   print('Letzter Zeitstempel: \t' + str(state_dict['ts']))
   print('Letzter Status: \t' + str(state_dict['active']))
   print('Code: \t\t\t' + str(state_dict['code']))
   print('Description: \t\t\t' + str(state_dict['description']))
   print(' ')
# end function createState
```

3.5 State of high-bay warehouse

The command python3 lernfabrik.py -t 1 displays the fill levels of the high bay warehouse. The high bay warehouse is filled with 9 bins of the colours RED, BLUE or WHITE.

python3 lernfabrik.py -t 1

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -t 1

Status Hochregal
Letzter Zeitstempel:    2021-09-28T08:00:58.940Z

                |       Num 1           |       Num 2           |       Num 3
-------------------------------------------------------------------------------
        | id:   | 045e55a2186580        | 046c57a2186580        | 045159a2186580
 ROW A  | state:| RAW                   | RAW                   | RAW
        | type: | RED                   | WHITE                 | BLUE
-------------------------------------------------------------------------------
        | id:   | 046c57a2186580        | 045159a2186580        | 045b56a2186580
 ROW B  | state:| RAW                   | RAW                   | RAW
        | type: | WHITE                 | BLUE                  | RED
-------------------------------------------------------------------------------
        | id:   | 045159a2186580        | 045b56a2186580        | 044057a2186580
 ROW C  | state:| RAW                   | RAW                   | RAW
        | type: | BLUE                  | RED                   | WHITE
-------------------------------------------------------------------------------
```

3.5.1 Source code

The function createStock() shows the current state of the high-bay warehouse.

```
# Gibt den aktuellen Status des Hochregals zurueck
def createStock(stock):
   value = getJSONFromHDFSByKey( stock_topic )
   stock_dict = json.loads(value)

   print(' ')
   print(' Status Hochregal')
   print(' Letzter Zeitstempel: \t' + str(stock_dict['ts']))
   print(' ')

   if(stock == 1): # manuelle Ausgabe
      print('\t\t| \tNum 1 \t\t| \tNum 2 \t\t| \tNum 3')
      print('-------------------------------------------------------
---------------------------------')

      for i in [0, 3, 6]:
         # find empty storage
         for j in [0, 1, 2]:
```

APPENDIX A

```
            try:
               temp =
stock_dict['stockItems'][j+i]['workpiece']['id']
            except:
               stock_dict['stockItems'][j+i]['workpiece']['id'] =
"0"
               stock_dict['stockItems'][j+i]['workpiece']['state']
= " "
               stock_dict['stockItems'][j+i]['workpiece']['type'] =
" "


         print(' \t| id: \t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['id']) + \
                                  ('\t\t\t| ' if
str(stock_dict['stockItems'][i+0]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+1]['workpiece']['id']) + \
                                     ('\t\t\t| ' if
str(stock_dict['stockItems'][i+1]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+2]['workpiece']['id']) )
         print(' ROW ' + ('A' if i == 0 else ('B' if i == 1 else
'C')) +' \t| state:| ' + \

str(stock_dict['stockItems'][i+0]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['state']) )
         print(' \t| type:\t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['type']))
         print('-----------------------------------------------------
-----------------------------------')
   else: # Standard ausgabe JSON
      print(json.dumps(stock_dict, indent=3, sort_keys=True))
   print(' ')
# end function createStock
```

3.6 Ordering

The command python3 lernfabrik.py -o can be used to order a container with the colours WHITE, RED or BLUE. The container is provided in the output tray and then automatically stored again so that it can be ordered again.

python3 lernfabrik.py -o RED

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -o RED

Es wird ein Werkstück mit der Farbe "RED" bestellt.
Connected to broker via TCP
Bestellvorgang war erfolgreich!
```

3.6.1 Source code

The function createOrder() is used to trigger an order process.

```
# Gibt einen Bestellvorgang auf
# {"ts":"YYYY-MM-DDThh:mm:ss.fffZ", "type":"BLUE"}
def createOrder(color):
   print(' ')
   print('Es wird ein Werkstück mit der Farbe \"' + color +'\"
bestellt.')
   print(' ')

   # get flag
   if(getSimulateFlag(1,False)):
      print(' Bestellung kann nicht ausgefuehrt werden da im Moment
ein Bauteil bestellt wird.')
      print(' ')
      return 1

   # MQTT Verbindung aufbauen
   client = paho.Client(client_id)
   client.on_publish = on_publish
   client.on_connect = on_connect
   client.connect(broker, port=port)
   client.loop_start()

   while connected != True:
      time.sleep(0.1)

   # aktueller Zeitstempel
   the_time = datetime.datetime.now()
   time_stamp = the_time.strftime(format = "%Y-%m-%dT%H:%M:%S.%f")
```

```
   order = '{\"ts\":\"' + time_stamp[:-3] + 'Z\", \"type\":\"' +
color + '\"}'
   client.publish(order_topic, order, qos=1)
   time.sleep(1)
   print(' ')
# end function createOrder
```

After placing an order, you can use the "-f" argument to track the status of each station.

python3 learningfactory.py -f

3.7 Tyni-Client source code

lernfabrik.py

```
import argparse
import sys
import os
import json
import paho.mqtt.client as paho
import datetime
import time
import http.server
import socketserver

hdfsMainDir = '/user/lernfabrik'
colors = ['RED', 'BLUE', 'WHITE']
stations = ['hbw', 'vgr', 'mpo', 'sld', 'dsi']
modi = [0, 1]
image_json = ''

# MQTT-Broker
connected = False
broker = '141.87.109.227'
port = 1883
http_port = 8000
client_id = 'hs_client_order_lernfabrik'
order_topic = 'f/o/order'<
```

The MyHttpRequestHandler() class has the do_GET() function. This writes the HTML page to the HTTPRequestHandler. This in turn is used by the web server to send the web page on request.

```
class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
   def do_GET(self):
      # Sending an '200 OK' response
      self.send_response(200)
```

```
      # Setting the header
      self.send_header("Content-type", "text/html")
      # Whenever using 'send_header', you also have to call
'end_headers'
      self.end_headers()

      global image_json

      # Some custom HTML code, possibly generated by another
function
      html = '<html><head></head><body><h1>current camera
image</h1><div>time stamp: ' + str(image_json['ts']) + \
             '</div><div><img src=\"' + str(image_json['data']) +
'\" alt=\"camera image\"
style=\"width:640px;height:480px;\"/></div></body></html>'

      # Writing the HTML contents with UTF-8
      self.wfile.write(bytes(html, "utf8"))

      return
# end Class MyHttpRequestHandler
```

The function startWebServer() starts the web server.

```
# Startet HTTP Server
def startWebServer():
   try:
      #handler = http.server.SimpleHTTPRequestHandler
      handler = MyHttpRequestHandler
      with socketserver.TCPServer(("", http_port), handler) as
httpd:
         #print("Server started at localhost:" + str(http_port))
         httpd.serve_forever()
   except:
      print("Fehler Bei HTTP Server: ", sys.exc_info()[0])
      raise
# end function startWebServer
```

The function getJSONFromHDFSByKey() is a helper function, which reads the data to a specific topic from HDFS.

```
# Holt einen Wert in HDFS by Key
def getJSONFromHDFSByKey(key):
   try:
      loc = hdfsMainDir + '/' + key
      value = os.popen("hdfs dfs -cat " + loc).readlines()

      value2 = ''
      for val in value:
         value2 += val
```

```
      return value2
   except:
      print("Fehler Bei HDFS Abfrage: ", sys.exc_info()[0])
      raise
# end function getJSONFromHDFSByKey
```

The function craeteState() shows the current state of a specific station.

```
# Gibt Status zur einer Station zurueck
# {"ts":"2021-09-
22T09:32:45.436Z","station":"sld","code":1,"description":"","active
":false,"target":""}
def createState(station):
   value = getJSONFromHDFSByKey('f_i_state_' + station)
   state_dict = json.loads(value)

   print(' ')
   print('Status Station: \t' + str(state_dict['station']))
   print('Letzter Zeitstempel: \t' + str(state_dict['ts']))
   print('Letzter Status: \t' + str(state_dict['active']))
   print('Code: \t\t\t' + str(state_dict['code']))
   print('Description: \t\t\t' + str(state_dict['description']))
   print(' ')
# end function createState
```

The function createSTock() shows the current filling level of the high-bay warehouse.

```
# Gibt den aktuellen Status des Hochregals zurueck
def createStock(stock):
   value = getJSONFromHDFSByKey( stock_topic )
   stock_dict = json.loads(value)

   print(' ')
   print(' Status Hochregal')
   print(' Letzter Zeitstempel: \t' + str(stock_dict['ts']))
   print(' ')

   if(stock == 1): # manuelle Ausgabe
      print('\t\t| \tNum 1 \t\t| \tNum 2 \t\t| \tNum 3')
      print('-----------------------------------------------------
---------------------------------')

      for i in [0, 3, 6]:
         # Leeres Regalfach abfangen
         for j in [0, 1, 2]:
            try:
               temp =
stock_dict['stockItems'][j+i]['workpiece']['id']
            except:
               stock_dict['stockItems'][j+i]['workpiece']['id'] =
"0"
               stock_dict['stockItems'][j+i]['workpiece']['state']
```

```
= " "
                stock_dict['stockItems'][j+i]['workpiece']['type'] =
" "


         print(' \t| id: \t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['id']) + \
                                    ('\t\t\t| ' if
str(stock_dict['stockItems'][i+0]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+1]['workpiece']['id']) + \
                                      ('\t\t\t| ' if
str(stock_dict['stockItems'][i+1]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+2]['workpiece']['id']) )
         print(' ROW ' + ('A' if i == 0 else ('B' if i == 1 else
'C')) +' \t| state:| ' + \

str(stock_dict['stockItems'][i+0]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['state']) )
         print(' \t| type:\t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['type']))
         print('-----------------------------------------------------
-----------------------------------')
   else: # Standard ausgabe JSON
      print(json.dumps(stock_dict, indent=3, sort_keys=True))
   print(' ')
# end function createStock
```

The function on_connect() is a callback, which gets called as soon as the MQTT-Broker is connecting.

```
# Callback Funktion beim connecten
def on_connect(client, userdata, flags, rc):
   if rc == 0:
      print("Connected to broker via TCP")
      global connected            # Use global variable
      connected = True            # Signal connection
   else:
```

```
      print("Connection failed via TCP")
# end function on_connect
```

The function on_publish() gets called, when a message is published.

```
# Callbak Funktion beim publishen
def on_publish(client, userdata, result):
   print('Bestellvorgang war erfolgreich!')
# end fcuntion on_publish
```

The function createOrder() triggers a ordering process.

```
# Gibt einen Bestellvorgang auf
# {"ts":"YYYY-MM-DDThh:mm:ss.fffZ", "type":"BLUE"}
def createOrder(color):
   print(' ')
   print('Es wird ein Werkstück mit der Farbe \"' + color +'\"
bestellt.')
   print(' ')

   # Blockierungs Flag holen
   if(getSimulateFlag(1,False)):
      print('Bestellung kann nicht ausgefuehrt werden da im Moment
ein Bauteil bestellt wird.')
      print(' ')
      return 1

   # MQTT Verbindung aufbauen
   client = paho.Client(client_id)
   client.on_publish = on_publish
   client.on_connect = on_connect
   client.connect(broker, port=port)
   client.loop_start()

   while connected != True:
      time.sleep(0.1)

   # aktueller Zeitstempel
   the_time = datetime.datetime.now()
   time_stamp = the_time.strftime(format = "%Y-%m-%dT%H:%M:%S.%f")
   order = '{\"ts\":\"' + time_stamp[:-3] + 'Z\", \"type\":\"' +
color + '\"}'
   client.publish(order_topic, order, qos=1)
   time.sleep(1)
   print(' ')
# end function createOrder
```

The function createCam() shows the current camera image.

```
# Gibt das aktuelle Kamerabild zurueck
def createCam(cam):
   try:
```

```
      value = getJSONFromHDFSByKey('i_cam')
      global image_json
      image_json = json.loads(value)
      print(' ')

      if(cam == 1):
         print('Um sich das Kamerabild anzeigen zu lassen öffnen
sie folgende URL im Browser:')
         print(' ')
         print('\thttp://kubernetes-master:' + str(http_port))
         print(' ')
         print('Anzeige muss manuell mit str+c abgebrochen
werden!')
         print(' ')
         startWebServer()
      else:
         print('Letzter Zeitstempel: \t' + str(image_json['ts']))
         print('Daten: \t\t\t' + str(image_json['data'][:60]))
      print(' ')
   except:
      print("Fehler Bei HDFS Abfrage: ", sys.exc_info()[0])
      raise
# end function createCam
```

The function parse_arg() is a helper function to evaluate the passed arguments.

```
# Argumente verwaltung
def parse_args():
    parser = argparse.ArgumentParser(
        description='Benutzung python3 lernfabrik.py -XX \n' \
                    'z.B. lernfabrik.py -o RED \n' \
                    'z.B. lernfabrik.py -s hbw \n' \
                    'z.B. lernfabrik.py -t 1'
    )
    parser.add_argument(
        '-o', '--order',
        choices=colors,
        help='Bestellung aufgeben.',
    )
    parser.add_argument(
        '-s', '--state',
        choices=stations,
        help='Stations Status abfragen.',
    )
    parser.add_argument(
        '-t', '--stock',
        choices=modi,
        type=int,
        help='Status des Hochregals abfragen.',
    )
    parser.add_argument(
        '-f', '--follow',
```

```
        action='store_true',
        help='Sequentielles Abfragen aller Stati.',
    )
    parser.add_argument(
        '-c', '--cam',
        choices=modi,
        type=int,
        help='Aktuelles Kamerabild anzeigen.',
    )
    parser.add_argument(
        '-m', '--movecam',
        metavar='movecam',
        nargs=2,
        help='Kamerasteuerung. Richtung: ' + str(directions) + ';
Winkel: ' + str(angle),
    )
# end function parse_args
```

The main() function.

```
def main():
    try:
        args = parse_args()

        # Bestellung pruefen
        if(args.order): #!= None):
           createOrder(args.order)

        # Status einer Station pruefen
        if(args.state): #!= None):
           createState(args.state)

        # Status des Hochregals pruefen
        if(args.stock != None):
           createStock(args.stock)

        # Sequentielles Abfragen der Stati
        if(args.follow): #!= None):
           createFollow()

        # Letztes Kamerabild anzeigen
        if(args.cam != None):
           createCam(args.cam)

        # Kamera bewegen
        if(args.movecam != None):
           if(args.movecam[0] in directions and
int(args.movecam[1]) in angle):
              moveCam(args.movecam[0], args.movecam[1])
           else:
              print('Bitte die Parameter beachten: Richtung: ' +
str(directions) + '; Winkel: ' + str(angle))
```

```
    except KeyboardInterrupt:
        print(" Interrupted")
        exit(0)

if __name__ == "__main__":
    main()
```

3.8 Exercises

Connect to the university cloud as described in Module 1 and solve the following tasks:

(a) Trigger an ordering process using the Tiny client. (Section 3.1.1 / 3.5)

b) Then, cyclically determine the status of each station and the high-bay warehouse to see the passage of the workpiece. (Section 3.4)

APPENDIX A

CHAPTER 4 Module 4

Welcome to module 4.

In module 4 the data analysis with Python is explained.

Monitoring data is read from the learning factory via Apache Nifi and stored in a relational database. AWS Redshift is used for this purpose. The Redshift database runs in the Amazon Cloud (AWS). Data analysis is performed using a Python script. The script is located in a Jupyter notebook in the university cloud at the following URL: https://hivemqserver.feste-ip.net:9001 The credentials are: User: lernfabrik Password: L5nf1br|k

Learning Objectives

- Data analysis using Python for IoT data.

Materials

- Python script source code brightness sensor visualization
- Python script source code environmental sensor visualization

4.1 AWS Redshift Connector

To access the AWS Redshift database, the RedShift Connector is required. The connector is already installed. The connector can be installed with pip install redshift_connector in the Jupyter notebook.

Installation:

```
python3.9 -m pip install redshift_connector
```

With the method redshift_connector_connect a connection to the RedShift database in the AWS Cloud is established. Connection data:

```
# Verbindungsdaten
conn = redshift_connector.connect(
    host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
    port=5439,
    database='dev',
    user='lernfabrik',
    password=L5nf1br|k
)
```

4.2 SQL statement ldr

In the table lernfabrik_ldr the brightness values within the plant are stored.

The column Brightness indicates the brightness in %. The sensor value column specifies the value measured by the sensor.

The time period for which the sensor data is to be created is specified, as well as the access data of the Connector.

```
cursor: redshift_connector.Cursor = conn.cursor()
cursor.execute("select * from lernfabrik.lernfabrik_ldr where ts >
'" + \
         zeitstempel_von + "' and ts < '" + zeitstempel_bis + "'
order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()
conn.close()
```

Output of the sensor values

```
df.columns = ["Zeitstempel", "Helligkeit", "Sensorwert"]
df
```

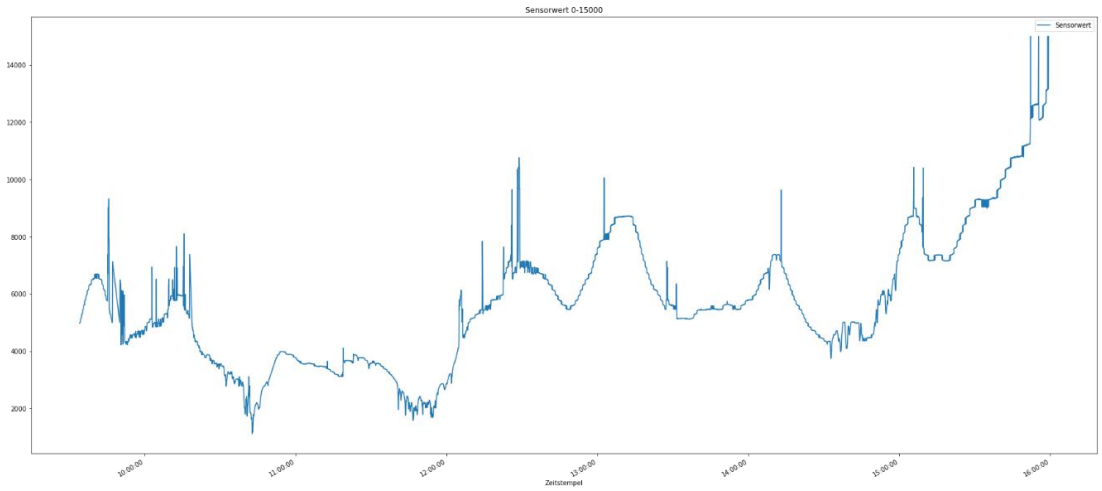
	Time stamp	Brightness	Sensor value
0	2022-02-06 15:43:42.470	0.0	15000
1	2022-02-06 15:43:39.454	12.3	13156
2	2022-02-06 15:43:36.450	12.3	13161
3	2022-02-06 15:43:27.470	0.0	15000
4	2022-02-06 15:43:24.424	12.3	13151
...	...	...	...

4.3 Brightness sensor value

The sensor value is the raw value of the brightness sensor. This is a photoresistor and has the unit Ohm. The value is between 0 - 15.000. The higher the resistor value the darker.

The select statement is executed and the data sets are returned as DataFrame. Thus the data can be visualized directly. Furthermore, more meaningful names are assigned to the column names.

```
# Sensorwert Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Sensorwert'], figsize=(30,14),
title='Sensorwert 0-15000')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



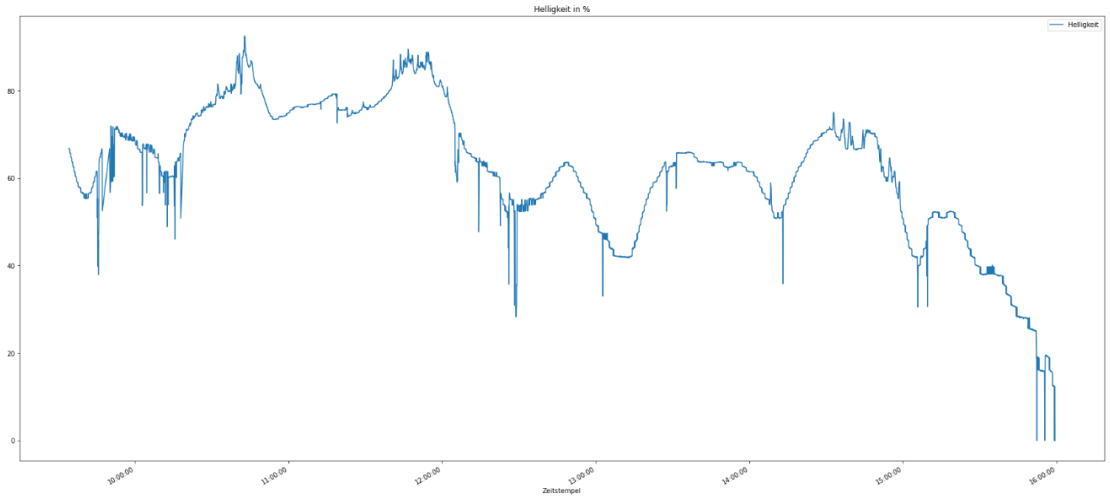
4.4 Brightness in %

Conversely, the percentage display is mirror-inverted, since for us 100% is bright and 0% is dark. Brightness in percent is a percentage representation of the raw value of the sensor.

Brightness in % = 1 – (Sensor value / 15000)

Therefore the graphs are identical, but mirrored.

```
# Helligkeit in %
ax0 = df.plot( x='Zeitstempel', y=['Helligkeit'], figsize=(30,14),
title='Helligkeit in %')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



4.5 Brightness sensor source code

lernfabrik_helligkeitssensor.py

```
import redshift_connector
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from matplotlib.dates import DateFormatter

# Zeitstempel fuer die Grafiken
zeitstempel_von = "2021-10-07 00:00:00.000"
zeitstempel_bis = "2021-10-08 00:00:00.000"

# Verbindung
conn = redshift_connector.connect(
   host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
   port=5439,
   database='dev',
   user='lernfabrik',
   password='L5nf1br|k'
)

cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_ldr where ts >
'" + zeitstempel_von + "' and ts \
   < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()
```

```
df.columns = ["Zeitstempel", "Helligkeit", "Sensorwert"]
df

%matplotlib inline

# Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Helligkeit'],
                      figsize=(30,14), title='Helligkeit in %')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)

# Sensorwert Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Sensorwert'], figsize=(30,14),
title='Sensorwert 0-15000')
ax0.xaxis.set_major_formatter(date_form)
```

4.6 SQL-Statement bme680

In the table lernfabrik_bme680 air data is saved.

CREATE TABLE
  lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
	Zeitstempel
	Sensorwert (gas resistance [Ohm])
	Luftqualität Genauigkeit [0-3]
	Luftqualität [index air quality 0-500]
	Luftdruck [hPa]
	Luftfeuchtigkeit [raw-value]
	Luftfeuchtigkeit [%]
	Temperatur [raw-value]
	Temperatur [°C]

The connector data is identical to those from the brightness sensor.

The select statement is executed. The records are returned as DataFrame. Here, the column names are also assigned more meaningful names.

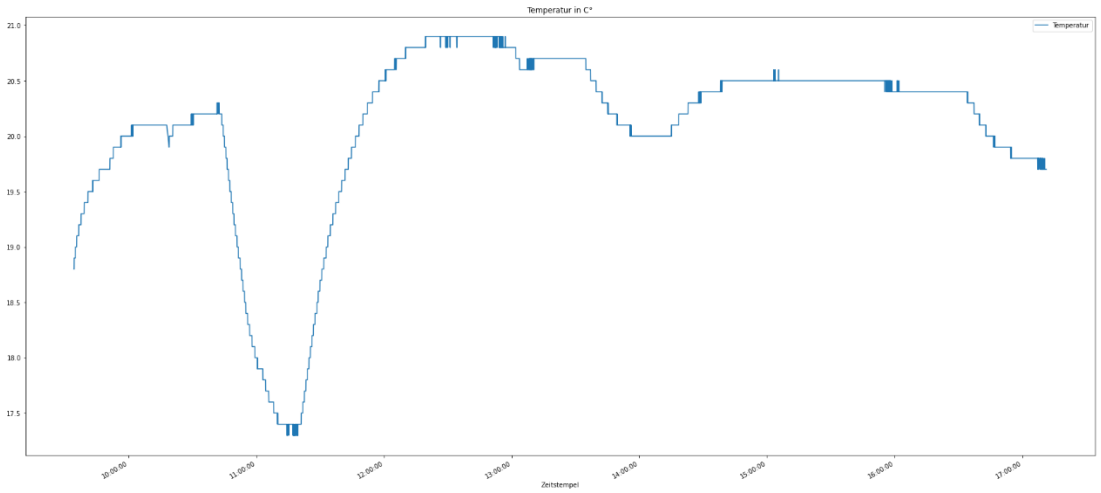
```
cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_bme680 where ts
> '" + zeitstempel_von + \
       "' and ts < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()
```

```
df.columns = ["Zeitstempel", "aq", "gr", "Luftfeuchtigkeit",
"Luftqualitaet", "Luftdruck", "rh", "rt", "Temperatur"]
df
```

	Time stamp	aq	gr	Humidity	Quality	Pressure	rh	rt	Temperature
0	2022-02-06 15:43:42.470	1	819622.0	42.7	30.0	935.3	34.77	16.20	13.0
1	2022-02-06 15:43:39.454	1	817629.0	42.7	31.0	935.3	34.77	16.20	13.0
...	...	...	...	...	...	...	...	...	...

4.7 4.7. Temperature

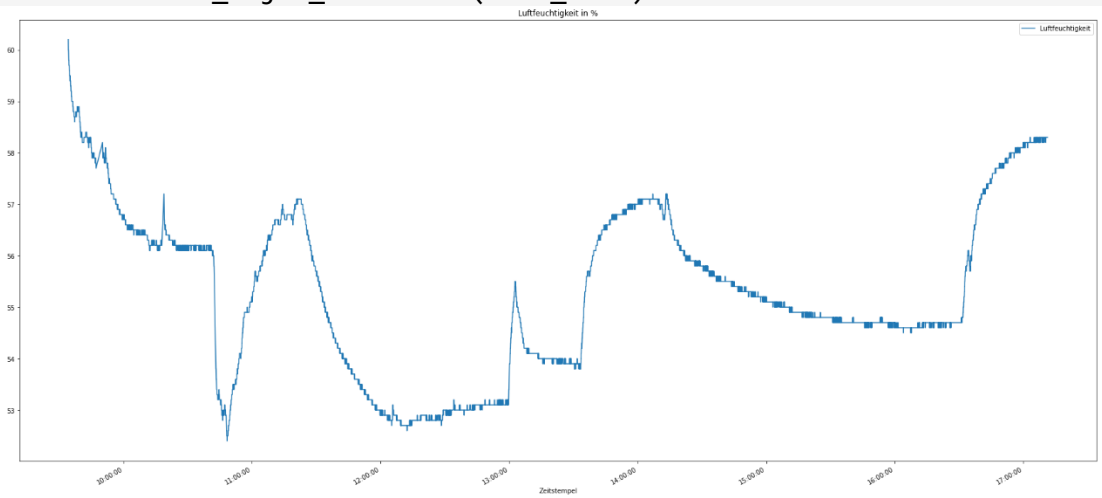
```
The following python script visualizes the temperature changes.
%matplotlib inline

# Temperatur
ax0 = df.plot( x='Zeitstempel', y=['Temperatur'],
                      figsize=(30,14), title='Temperatur in C°')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



4.8 Humidity

The following python script visualizes the humidity development.

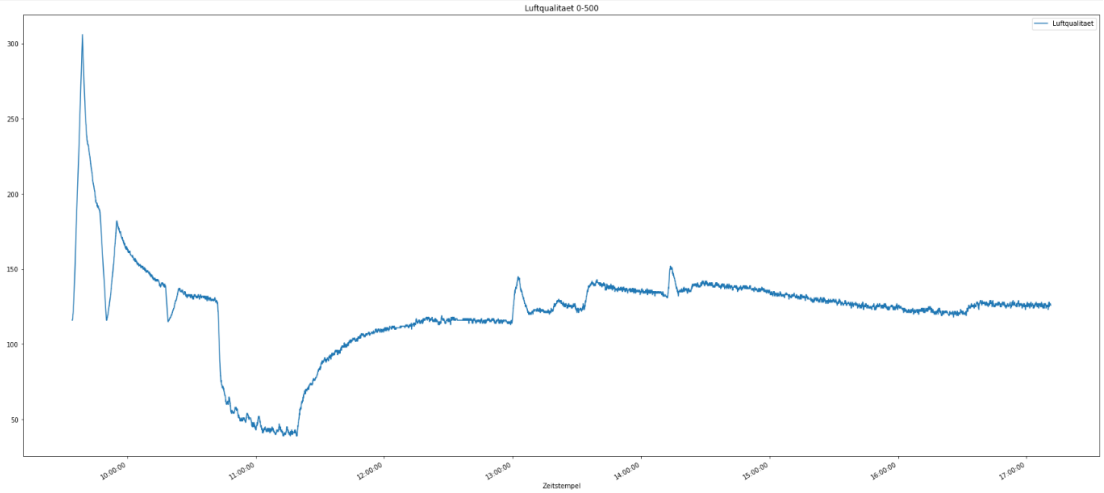
```
# Luftfeuchtigkeit
ax0 = df.plot( x='Zeitstempel', y=['Luftfeuchtigkeit'],
                        figsize=(30,14), title='Luftfeuchtigkeit in
%')
ax0.xaxis.set_major_formatter(date_form)
```



4.9 Air quality

The following python visualizes the air quality development.

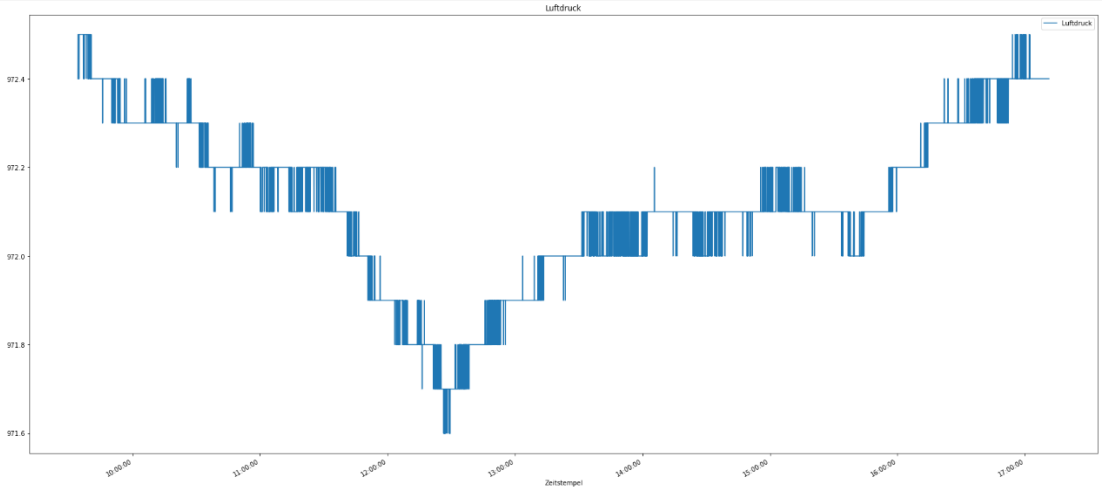
```
# Luftqualitaet
ax0 = df.plot( x='Zeitstempel', y=['Luftqualitaet'],
                        figsize=(30,14), title='Luftqualitaet 0-500')
ax0.xaxis.set_major_formatter(date_form)
```



4.10 Air pressure

The following python script visualizes the air pressure development.

```
# Luftdruck
ax0 = df.plot( x='Zeitstempel', y=['Luftdruck'],
                        figsize=(30,14), title='Luftdruck')
ax0.xaxis.set_major_formatter(date_form)
```



4.11 Environmental sensor source code

lernfabrik_umweltsensor.py

```
import redshift_connector
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from matplotlib.dates import DateFormatter

# Zeitstempel fuer die Grafiken
zeitstempel_von = "2021-10-07 00:00:00.000"
zeitstempel_bis = "2021-10-08 00:00:00.000"

# Verbindung
conn = redshift_connector.connect(
   host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
   port=5439,
   database='dev',
   user='lernfabrik',
   password='L5nf1br|k'
)
```

```
cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_bme680 where ts
> '" + zeitstempel_von + \
       "' and ts < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()

df.columns = ["Zeitstempel", "aq", "gr", "Luftfeuchtigkeit",
"Luftqualitaet", "Luftdruck", "rh", "rt", "Temperatur"]
df

%matplotlib inline

# Temperatur
ax0 = df.plot( x='Zeitstempel', y=['Temperatur'],
                      figsize=(30,14), title='Temperatur in C°')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)

# Luftfeuchtigkeit
ax0 = df.plot( x='Zeitstempel', y=['Luftfeuchtigkeit'],
                      figsize=(30,14), title='Luftfeuchtigkeit in
%')
ax0.xaxis.set_major_formatter(date_form)

# Luftqualitaet
ax0 = df.plot( x='Zeitstempel', y=['Luftqualitaet'],
                      figsize=(30,14), title='Luftqualitaet 0-500')
ax0.xaxis.set_major_formatter(date_form)

# Luftdruck
ax0 = df.plot( x='Zeitstempel', y=['Luftdruck'],
                      figsize=(30,14), title='Luftdruck')
ax0.xaxis.set_major_formatter(date_form)
```

4.12 Exercises

Display the sensor values for the period of the ordering process graphically. Use the scripts of this module as an orientation.

APPENDIX A

CHAPTER 5 Module 5

Welcome to module 5.

In module 5 we will get to know the OPC/UA communication between Node-RED and the production plant. The production plant is controlled by a Siemens PLC using OPC-UA.

Learning objectives

- Introduction to OPC/UA communication between Node-RED and a PLC.
- Read of the learning factory with Node-RED.

Materials

- Node-RED

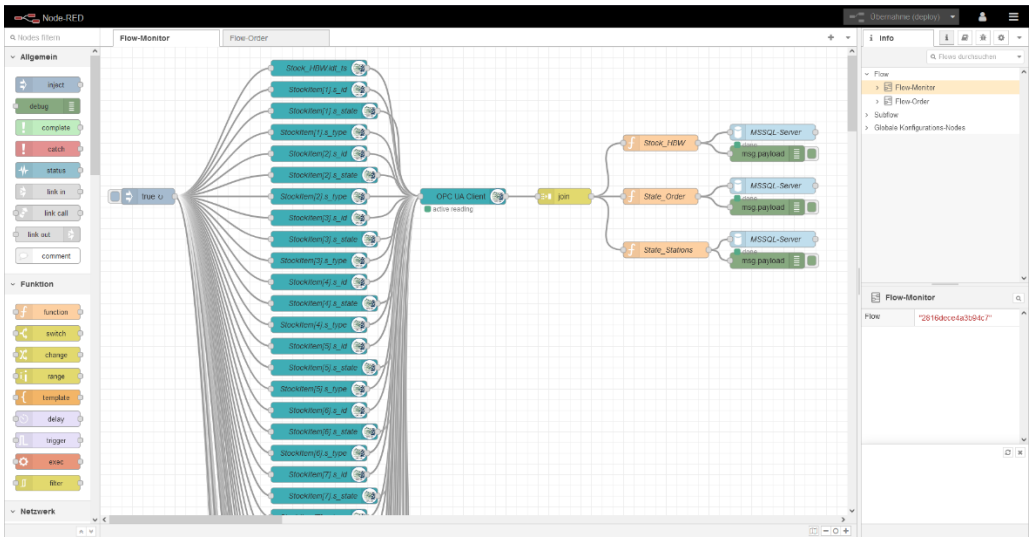
5.1 Introduction

Node-RED is a graphical open-source development tool from IBM, which makes it possible to implement various use cases in the field of IoT. For this purpose, a flow editor is available in the web browser, with which JavaScript functions can be created. In addition, numerous modules for the most common services and technologies are provided, which are connected with each other according to a simple modular principle. Each module symbolizes a node and has different tasks as input, output or processing node. The runtime environment is based on Node.js and the created flows are stored in JSON, which can be easily imported/exported or shared with others.

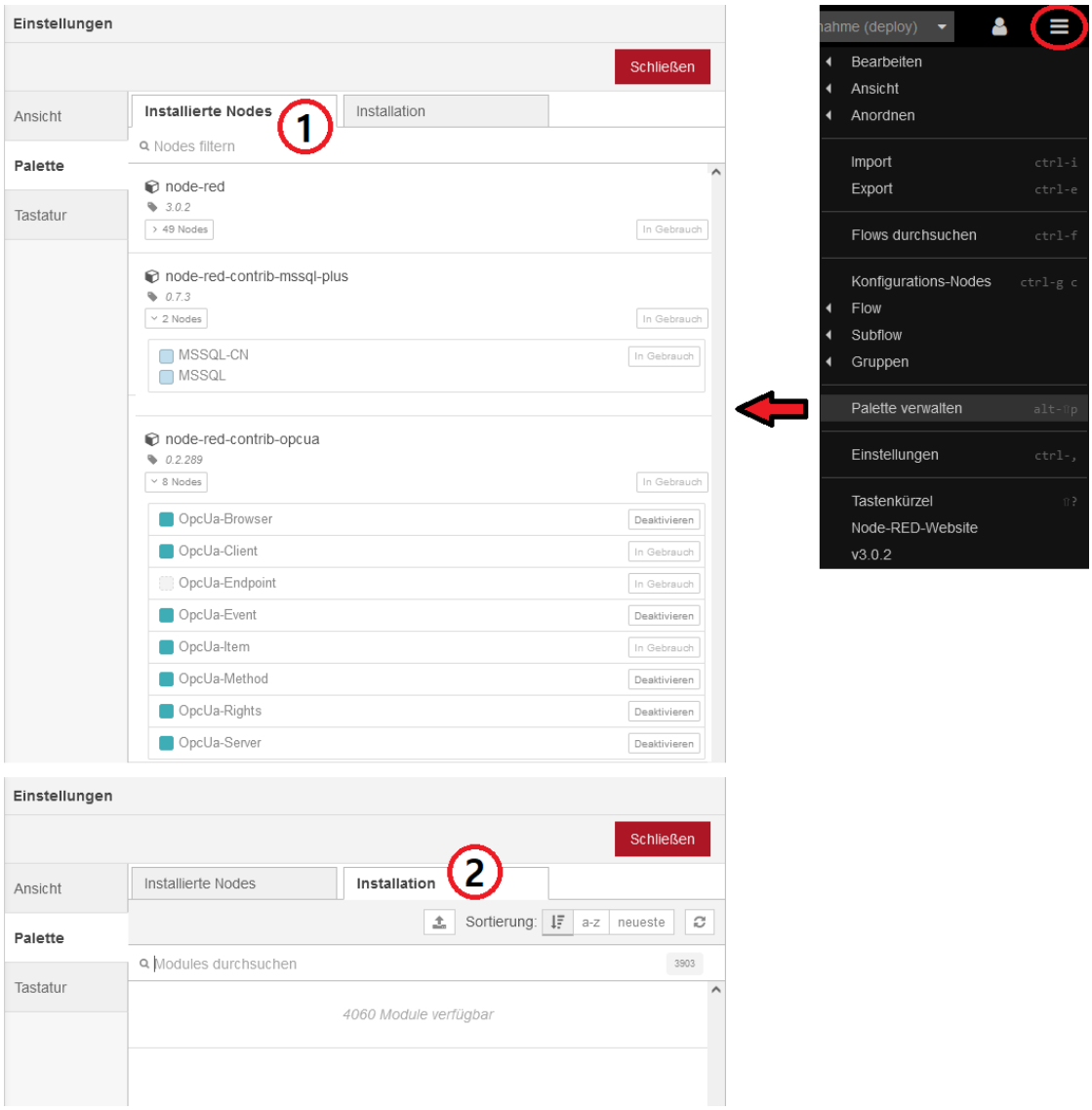
5.1.1 Flow editor

Node-RED is accessed in the web browser via the address https://hivemqserver.feste-ip.net:1880 (user: lernfabrik / password: L5nf1br|k). On the left side are the installed nodes, which are dragged and dropped into the flow editor. With a double click the

node specific settings can be defined and by connecting the different nodes the data flow can be defined.



Via the main menu (top/right icon with three stripes) under “Palette verwalten” (“manage palette”) you can access the settings of the installed nodes.



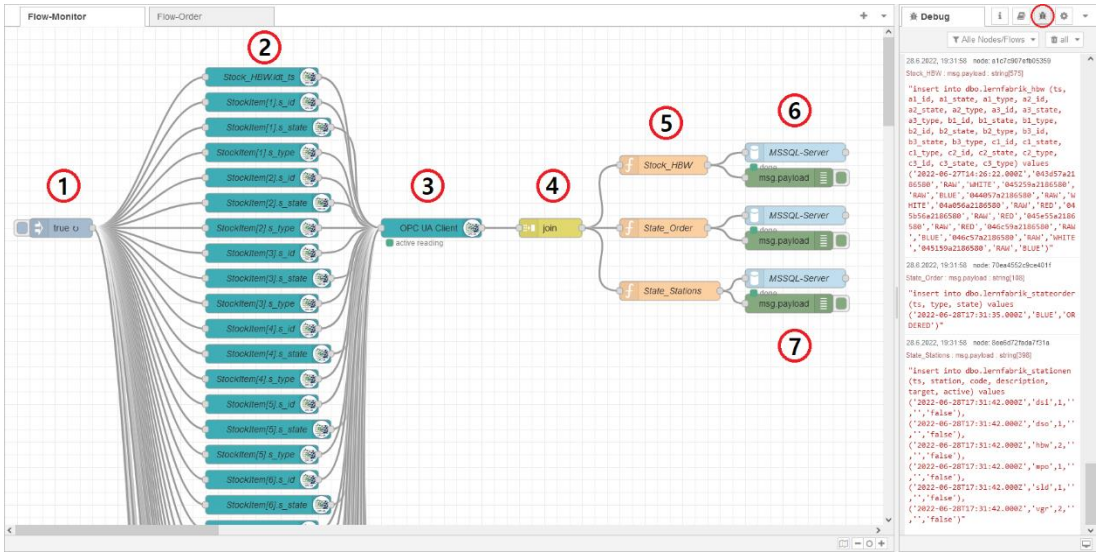
Here you get an overview of the installed nodes and can see which ones are currently in use (1). By default, 49 nodes of "node-red" are available. For our project, the nodes "node-red-contrib-opcua" for the OPC/UA communication with the learning factory and "node-red-contrib-mssql-plus" for the storage of the OPC values in our MSSQL database were additionally installed.

Additional nodes can be searched and installed in the second tab "Installation" (2). There are currently more than 4000 modules available for this purpose.

5.2 Reading from the learning factory

For reading the learning factory the flow “Flow-Monitor” was created. It consists of the following nodes:

1. inject
2. OpcUa-Item
3. OpcUa-Client
4. join
5. function
6. MSSQL
7. debug



The flow is started every 10 seconds with the "inject" node (1). The values from the "OPC items" (2) are read out with the "OPC-UA client" node (3) and written to an array with a "join" node (4).

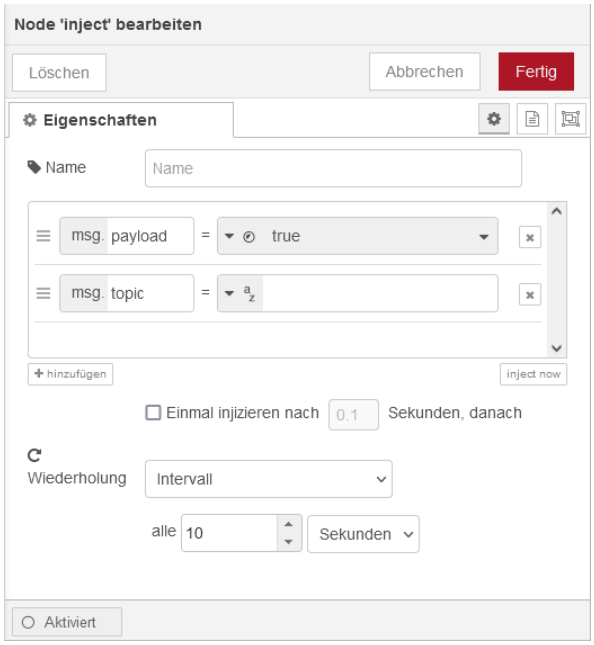
With the help of the "function" nodes (5), the values are read from the array and the corresponding SQL insert string is assembled. This is then passed to an "MSSQL" node (6) as a query, where the SQL insert is finally executed in the corresponding table.

With the "debug" nodes (7) the passed SQL insert strings are output in the right debug tab. The debug tab button is located on the right/top (fig. above - circled in red).

5.2.1 inject

The "inject" node starts the flow. This can be injected either manually with a click on the left button, at specific times or at regular time intervals. For our project a time interval of 10 seconds was chosen.

In "msg.payload" the payload of the flow is stored during the injection.



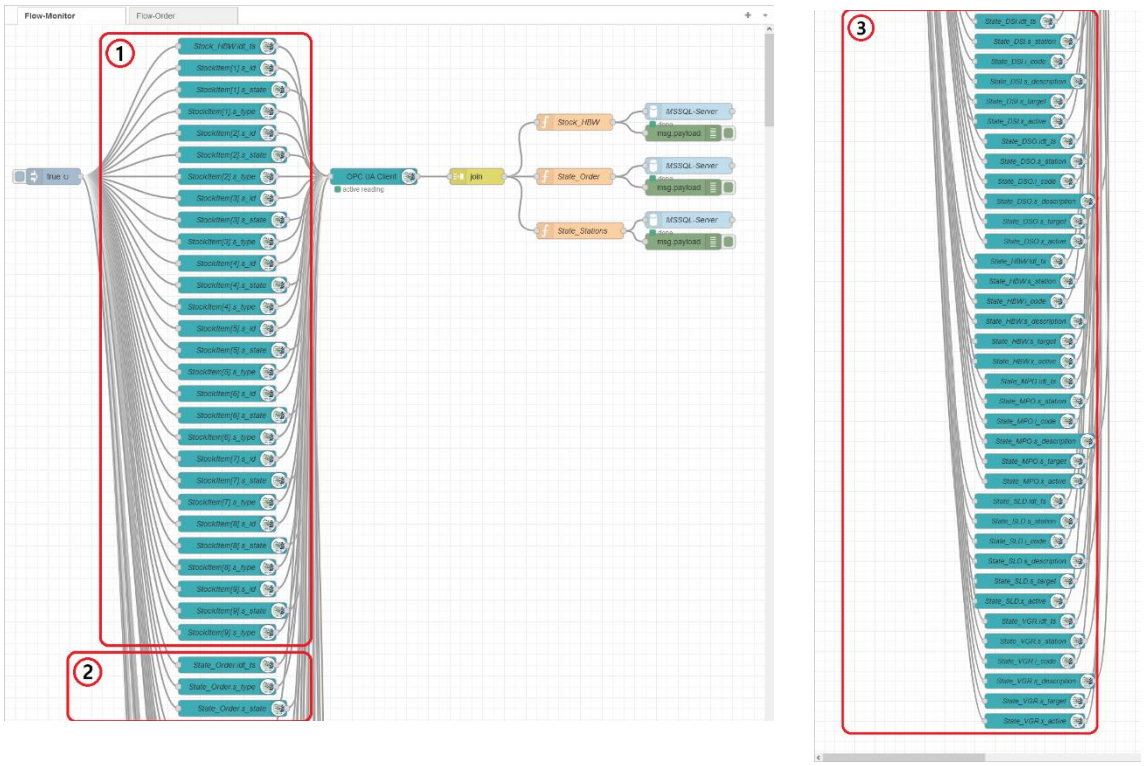
5.2.2 OpcUa item

Every single OPC data value to be read out is defined with an "OpcUa-Item" node.

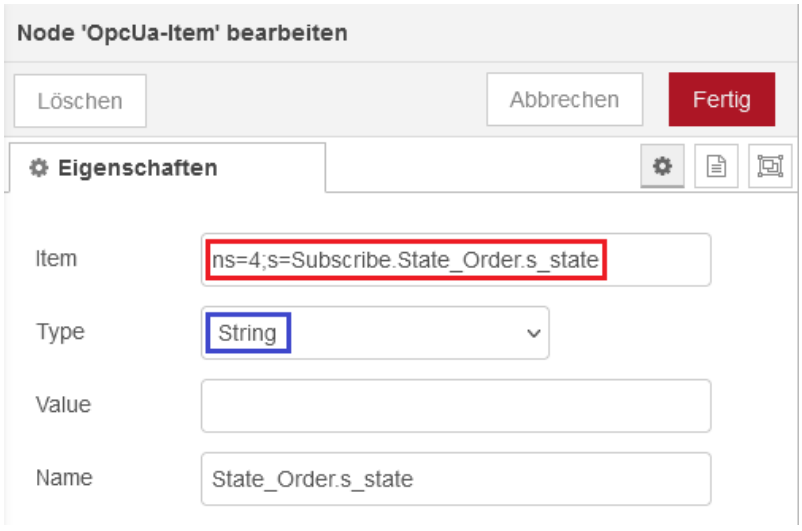
In our flow, a total of 67 values are read out. These are:

1. Stock of the high bay warehouse
2. Current order status
3. Status data of the various learning factory stations

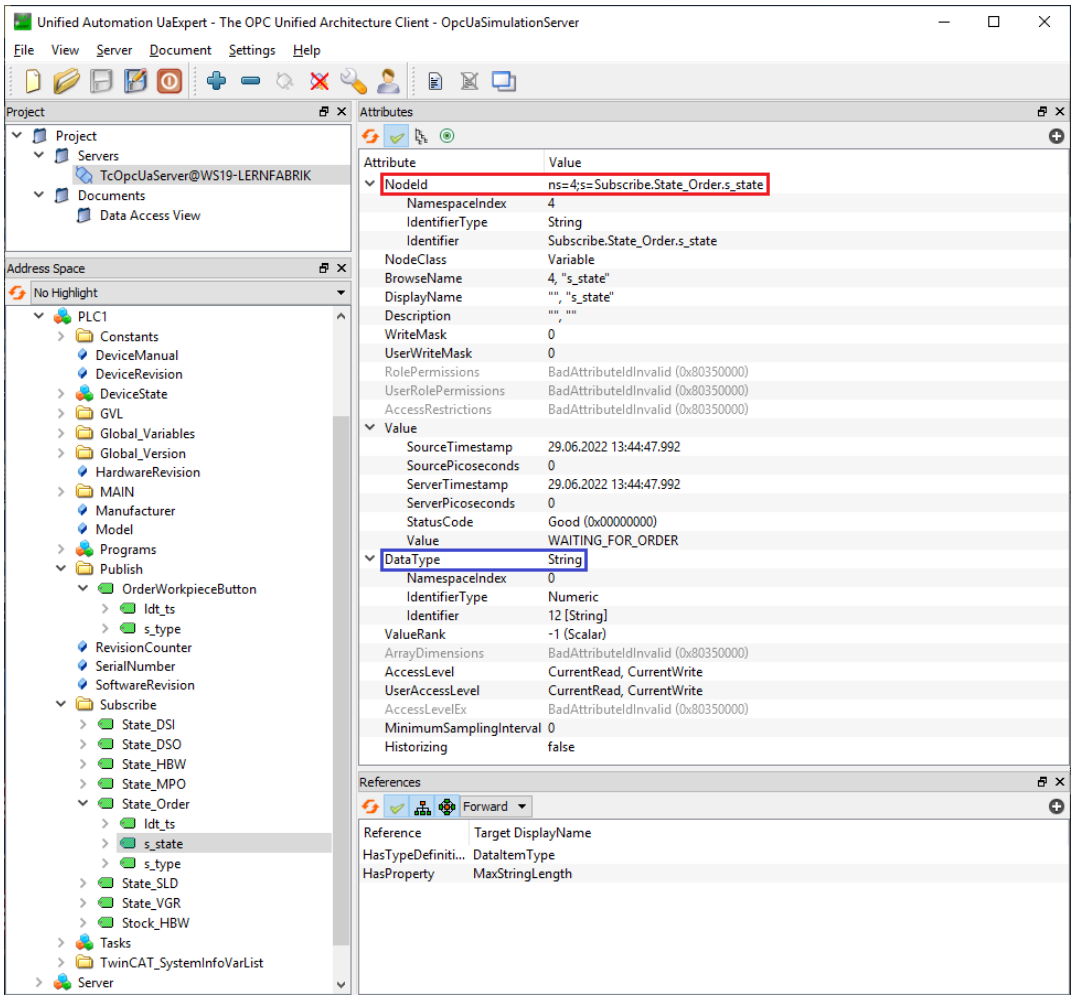
APPENDIX A



In the node settings the NodeID is given under “Item”. The NodeID consists of the Namespace-Index and the Identifier-String. Under “Type” the data type is given.

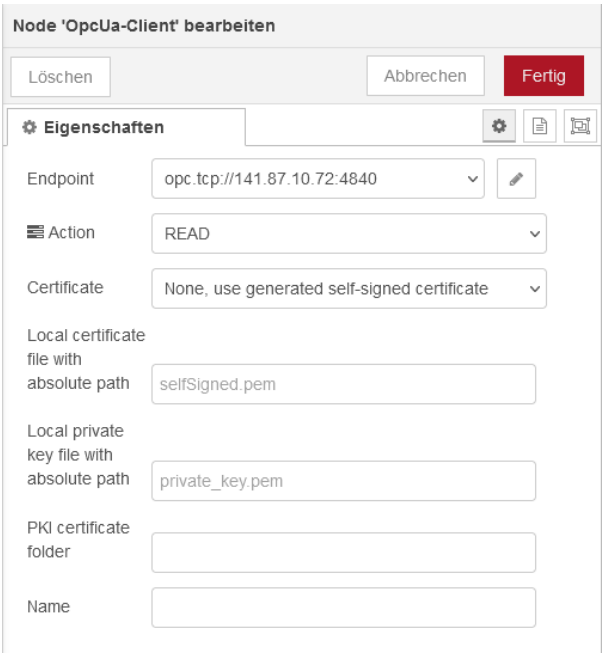


The "UA Expert" software was used to determine the values. This lists all OPC data points and values and also provides a comprehensive detailed overview:



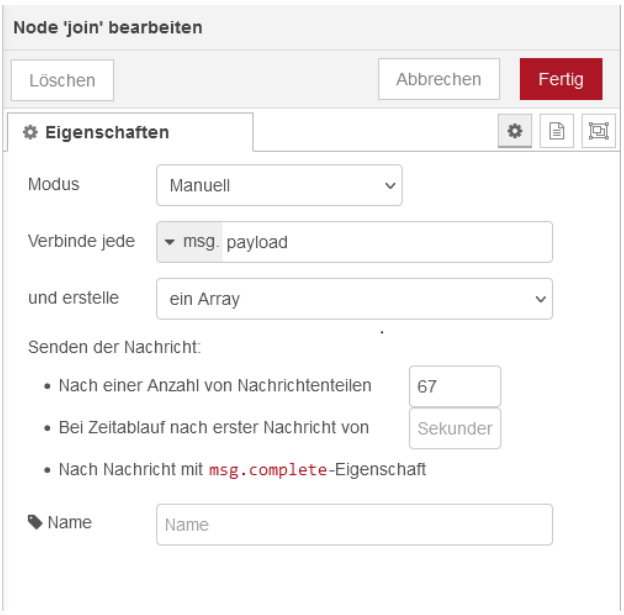
5.2.3 OpcUa-Client

The communication with the OPC/UA server is realized with the node “OpcUa-Client”. Therefore under “Endpoint” the OPC/UA server address is provided an finally the wished action is selected – in our case “READ” to read the values.



5.2.4 join

With the help of the “join” nodes the single 67 read OPC values are combined to an array “msg.payload[0-66]”.



5.2.5 function

With the "function" nodes the values are read from the array "msg.payload[0-66]", the corresponding SQL insert string is created, and finally forwarded as a message object (msg1). The message object contains a "topic" with the node name to make the output in the debug tab clearer and a "payload" with the created SQL insert string.

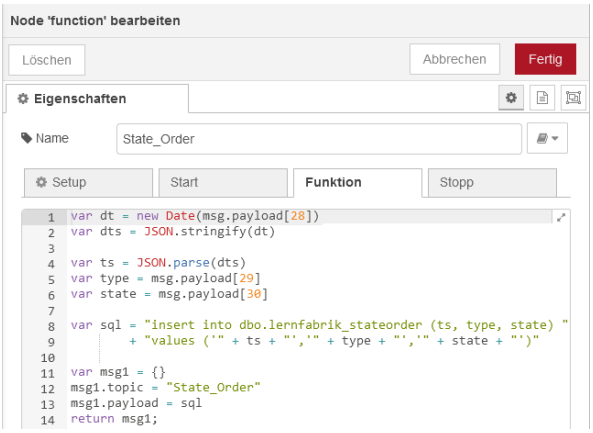
Stock_HBW:

APPENDIX A


```
var dt = new Date(msg.payload[0])
var dts = JSON.stringify(dt)

var ts = JSON.parse(dts)
var a1_id = msg.payload[1]
var a1_state = msg.payload[2]
var a1_type = msg.payload[3]
var a2_id = msg.payload[4]
var a2_state = msg.payload[5]
var a2_type = msg.payload[6]
var a3_id = msg.payload[7]
var a3_state = msg.payload[8]
var a3_type = msg.payload[9]
var b1_id = msg.payload[10]
var b1_state = msg.payload[11]
var b1_type = msg.payload[12]
var b2_id = msg.payload[13]
var b2_state = msg.payload[14]
var b2_type = msg.payload[15]
var b3_id = msg.payload[16]
var b3_state = msg.payload[17]
var b3_type = msg.payload[18]
var c1_id = msg.payload[19]
var c1_state = msg.payload[20]
var c1_type = msg.payload[21]
var c2_id = msg.payload[22]
var c2_state = msg.payload[23]
var c2_type = msg.payload[24]
var c3_id = msg.payload[25]
var c3_state = msg.payload[26]
var c3_type = msg.payload[27]

var sql = "insert into dbo.lernfabrik_hbw (ts, "
        + "a1_id, a1_state, a1_type, "
        + "a2_id, a2_state, a2_type, "
        + "a3_id, a3_state, a3_type, "
        + "b1_id, b1_state, b1_type, "
        + "b2_id, b2_state, b2_type, "
        + "b3_id, b3_state, b3_type, "
        + "c1_id, c1_state, c1_type, "
        + "c2_id, c2_state, c2_type, "
        + "c3_id, c3_state, c3_type) "
        + "values ('" + ts
        + "','" + a1_id + "','" + a1_state + "','" + a1_type
        + "','" + a2_id + "','" + a2_state + "','" + a2_type
        + "','" + a3_id + "','" + a3_state + "','" + a3_type
        + "','" + b1_id + "','" + b1_state + "','" + b1_type
        + "','" + b2_id + "','" + b2_state + "','" + b2_type
        + "','" + b3_id + "','" + b3_state + "','" + b3_type
        + "','" + c1_id + "','" + c1_state + "','" + c1_type
        + "','" + c2_id + "','" + c2_state + "','" + c2_type
        + "','" + c3_id + "','" + c3_state + "','" + c3_type + "')

var msg1 = {}
msg1.topic = "Stock_HBW"
msg1.payload = sql
return msg1;
```

State_Order:

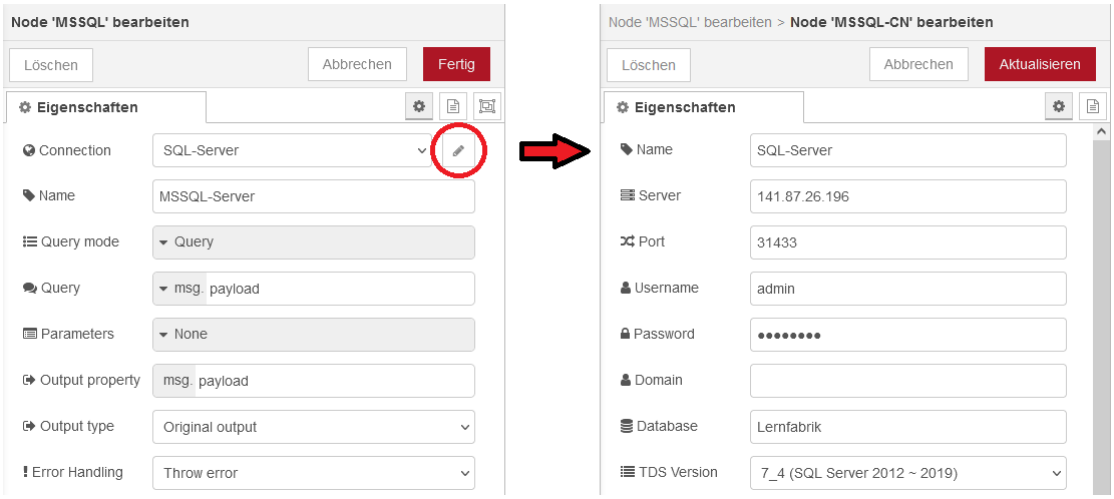
APPENDIX A



State_Stations:

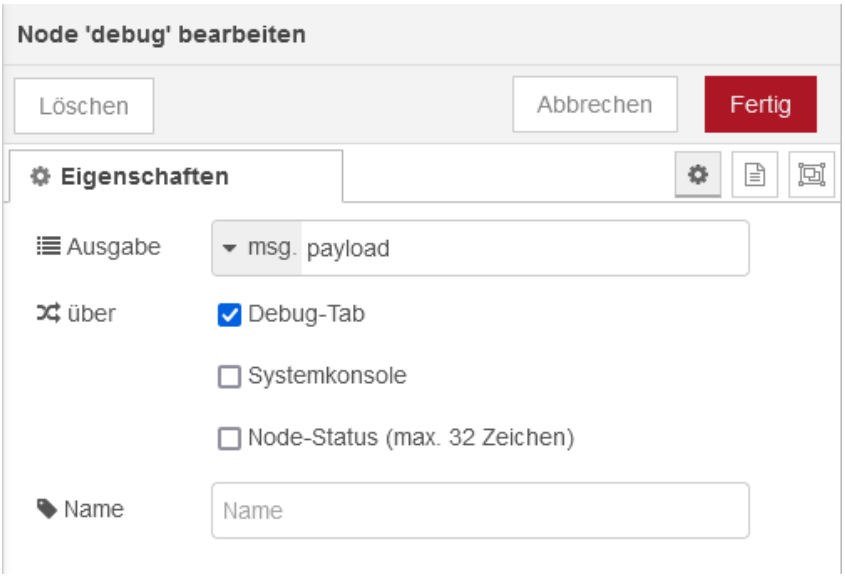
5.2.6 MSSQL

In the "MSSQL" node, the connection to the MSSQL server is defined via an MSSQL ConnectionNode. The edit button leads to the connection properties where the server and access data with the database to be used are specified.
Finally "Query" is selected as mode and with "msg.payload" the SQL statement to be executed from the "payload" string of the transferred message object is specified.



5.2.7 debug

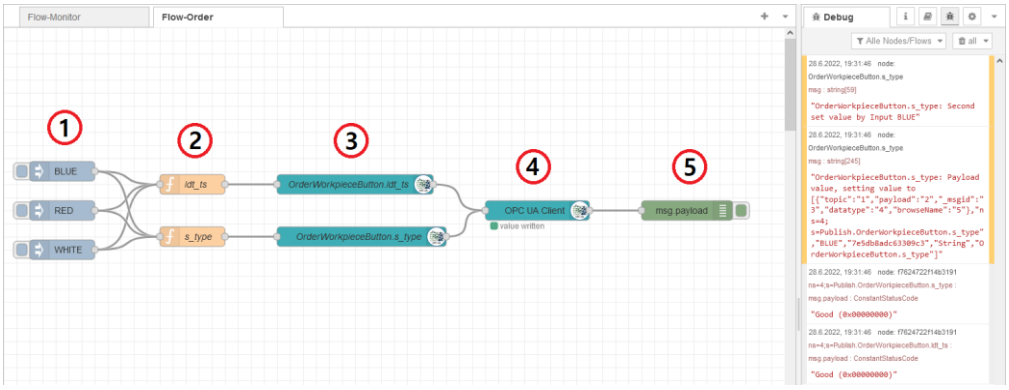
In the “debug” node the default settings with the output “msg.payload” are used by selecting the “Debug-Tab”.



5.3 Controlling the learning factory

For the open loop control of the learning factory the flow “Flow-Order” was created, that can trigger a ordering process. It consists of the following nodes:

1. inject
2. function
3. OpcUA-Item
4. OpcUa-Client
5. debug

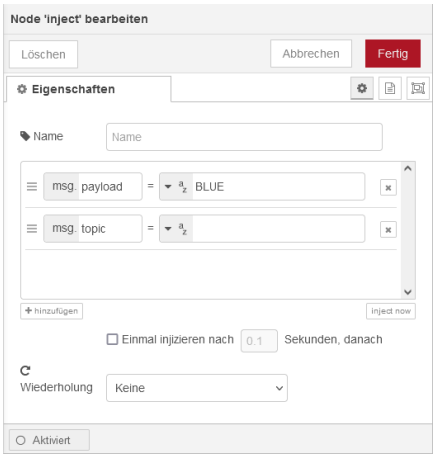


Three "inject" nodes (1) are available for the ordering process. Clicking on the left button of an "inject" node starts the ordering process with the desired workpiece color ("BLUE"/"RED"/"WHITE"). Via the function nodes (2), the time stamp (ldt_ts) and the workpiece color (s_type) are transferred to the corresponding OPC items (3), which are then sent to the OPC/UA server of the learning factory via the "OPC-UA Client" node (4).

Finally, the transferred values are output in the debug window using the "debug" node (5).

5.3.1 inject

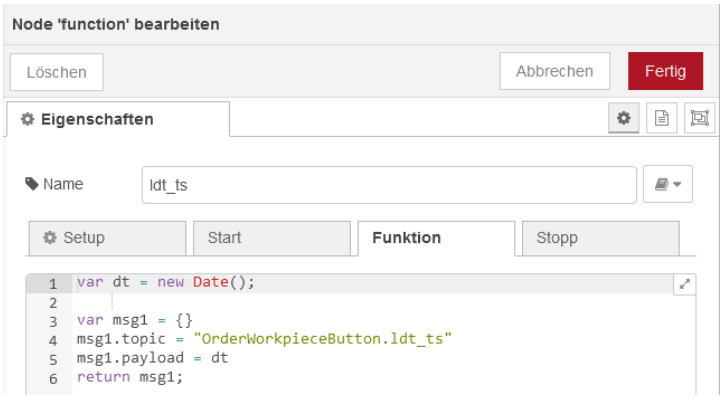
The "inject" nodes start the flow. They contain in "msg.payload" a string with the desired workpiece color. For our order flow only the manual injection is selected by switching off the automatic repetition.



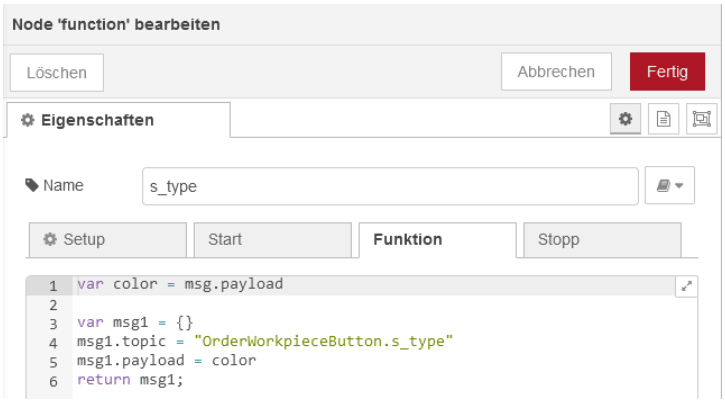
5.3.2 function

The "function" nodes are used to pass on the values required for the ordering process as a message object (msg1). These are the current time stamp in the "ldt_ts" node and the desired color, which is determined in the "s_type" node from "msg.payload".

ldt_ts:

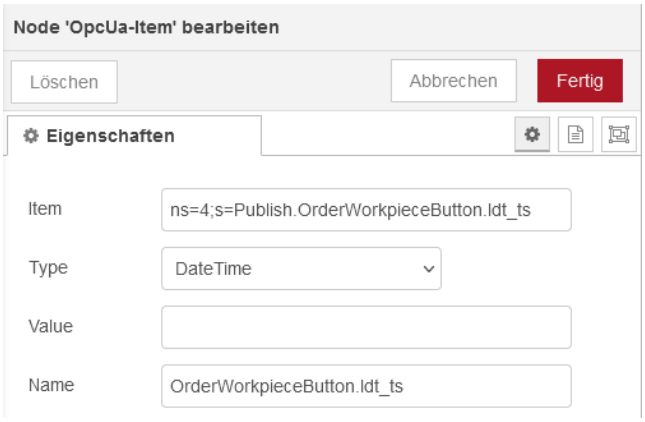


s_type

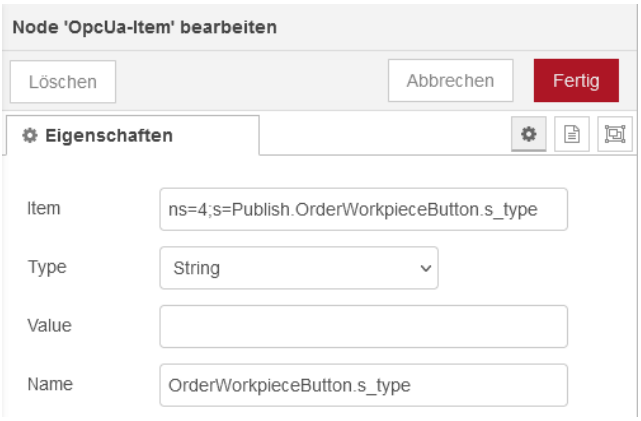


5.3.3 OpcUa-Item

OrderWorkpieceButton.ldt_ts:

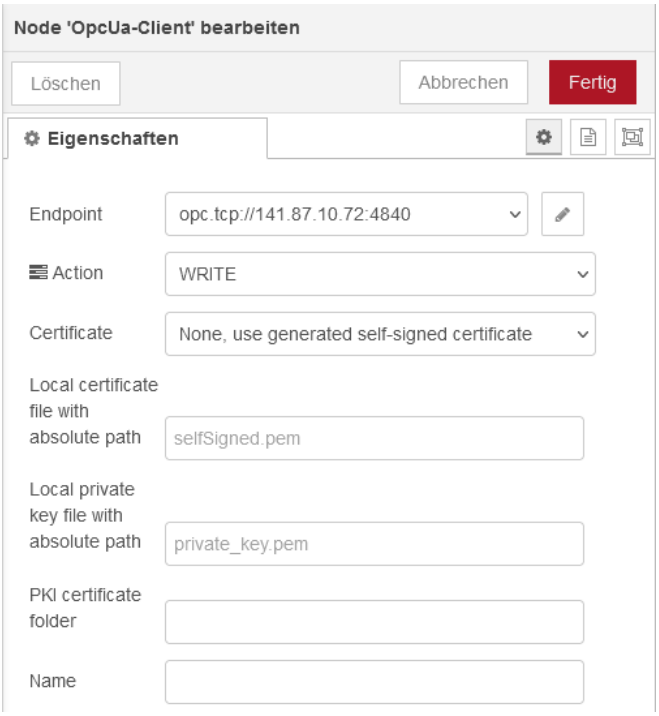


OrderWorkpieceButton.s_type:



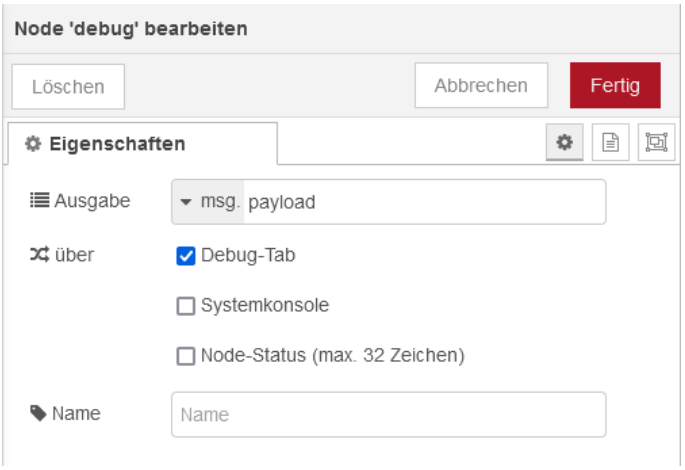
5.3.4 opcUa client

With the node "OpcUa-Client" the communication to the OPC/UA server of the learning factory simulation is realized. For this purpose, the OPC/UA server address is specified under "Endpoint" and finally the desired action is selected - in our case "WRITE" for writing the values.



5.3.5 debug

In the “debug” node the default settings with the output “msg.payload” are used by selecting the “Debug-Tab”.



5.4 Exercises

Connect to Node-Red at the following address https://hivemqserver.feste-ip.net:1880 (user: lernfabrik / password: L5nf1br|k) and solve the following tasks:

(a) Trigger an ordering process with Node-RED. (Section 5.3)

b) Determine the total duration of the order process including the putaway process using the values of the order state (State_Order) in the debug tab. (Section 5.2 / 5.3)

c) Track the transferred state changes during an order and putaway process. (Section 5.2 / 5.3)

APPENDIX A

CHAPTER 6 Module 6

Welcome to module 6.

Module 6 describes the database systems used and the data schema.

6.1 Nifi

The metadata for the two tables lernfabrik_bme680 and lernfabrik_ldr are described here.

Before the data is saved to the database using Nifi, the tables must be created. The tables are already created, the statements are only shown for completeness.

```
CREATE SCHEMA lernfabrik;
```

Contains all values of the environmental sensor

Enthält alle Werte des Umweltsensors

```
CREATE TABLE
lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Sensorwert (gas resistance [Ohm])
Luftqualität Genauigkeit [0-3]
Luftqualität [index air quality 0-500]
Luftdruck [hPa]
Luftfeuchtigkeit [raw-value]
Luftfeuchtigkeit [%]
Temperatur [raw-value]
Temperatur [°C]
```

Time stamp
Sensor value
Air quality precision
Air quality index
Air pressure
Raw humidity
Humidity
Raw temperature
Temperature

lernfabrik_bme680 (environmental sensor)

ts	gr	aq	iaq	p	rh	h	rt	t
29.09.2021 09:56:07	185450	3	144	966,799987792969	49,5200004577637	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:16	186598	3	142	966,900024414063	49,5	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:28	185762	3	144	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:34	185867	3	142	966,900024414063	49,5099983215332	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:49	185762	3	143	966,799987792969	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:58	187554	3	141	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:57:07	187013	3	142	966,900024414063	49,5	60	25,5900001525879	22,3999996185303

lernfabrik_ldr (brightness sensor)

APPENDIX A

Contains all values of the brightness sensor

Enthält alle Werte des Helligkeitssensors	

CREATE TABLE
lernfabrik.lernfabrik_ldr (
  ts datetime NOT NULL,
  ldr SMALLINT NOT NULL,
  br DOUBLE PRECISION NOT NULL
);
	Zeitstempel
	Light Dependent Resistor (0-15000[Ohm])
	Helligkeit [%] (wird aus ldr ermittelt)
	[ldr: 15000 = 0%; 7500 = 50%; 0 = 100%]

Time stamp
Brightness is determined from ldr value

ts	ldr	br
29.09.2021 09:56:04	4611	69,3000030517578
29.09.2021 09:56:16	4707	68,5999984741211
29.09.2021 09:56:28	4694	68,6999969482422
29.09.2021 09:56:34	4702	68,6999969482422
29.09.2021 09:56:49	4710	68,5999984741211
29.09.2021 09:56:58	4710	68,5999984741211
29.09.2021 09:57:07	4707	68,5999984741211

For the station on Hive the following table is created:

Contains relevant data for the stations

Enthält stationsrelevante Daten	

CREATE TABLE
lernfabrik_stationen(
  ts string,
  station string,
  code int,
  description string,
  target string,
  active boolean,
primary key(station) disable novalidate
);
	Zeitstempel
	Station (s. Tabelle unten)
	Acknowledgment-Code (s. Tabelle unten)
	Beschreibung
	Zielstation (s. Tabelle unten)
	Active-Flag [Aktiv=True; Inaktiv=False]

Time stamp
Station
Ack-Code (see below)
Description
Target station
Active flag

lernfabrik_stationen

ts	station	code	description	target	active
2021-10-21T09:39:59.088Z	dso	1			False
2021-10-21T09:40:01.094Z	dsi	1			False
2021-10-21T09:40:01.680Z	mpo	1			False
2021-10-21T09:40:02.111Z	hbw	1			False
2021-10-21T09:40:02.352Z	sld	0			True

Acknowledgment-Codes

Station	Beschreibung	Acknowledgment-Codes
dsi	Delivery and Pickup Station (Input)	-
dso	Delivery and Pickup Station (Output)	-
hbw	High-Bay Warehouse (Hochregallager)	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
mpo	Multi Processing Station (Bearbeitung)	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
sld	Sorting Line (Sortierstation)	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_SORTED
vgr	Vacuum Gripper Robot (Sauggreifer)	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

You can find these data in the HDFS directory /user/lernfabrik.

6.2 6.2 Node-Red

In Node-RED, the OPC data is written to a Microsoft SQL server cluster of the university cloud. Here, the metadata of the respective tables are described.
The following MSSQL tables were created for Node-RED:

APPENDIX A

Contains information to the storage level of the 9 containers in the high-bay warehouse

Enthält Infos zum Lagerbestand der 9 Container im Hochregallager

```
CREATE TABLE lernfabrik_hbw (
  ts datetime NOT NULL,
  a1_id VARCHAR(14),
  a1_state VARCHAR(9),
  a1_type VARCHAR(5),
  a2_id VARCHAR(14),
  a2_state VARCHAR(9),
  a2_type VARCHAR(5),
  a3_id VARCHAR(14),
  a3_state VARCHAR(9),
  a3_type VARCHAR(5),
  b1_id VARCHAR(14),
  b1_state VARCHAR(9),
  b1_type VARCHAR(5),
  b2_id VARCHAR(14),
  b2_state VARCHAR(9),
  b2_type VARCHAR(5),
  b3_id VARCHAR(14),
  b3_state VARCHAR(9),
  b3_type VARCHAR(5),
  c1_id VARCHAR(14),
  c1_state VARCHAR(9),
  c1_type VARCHAR(5),
  c2_id VARCHAR(14),
  c2_state VARCHAR(9),
  c2_type VARCHAR(5),
  c3_id VARCHAR(14),
  c3_state VARCHAR(9),
  c3_type VARCHAR(5)
);
```

```
Zeitstempel
RFID ['123456789ABCDE']
Status ['RAW'/'PROCESSED']
Werkstückfarbe ['BLUE'/'RED'/'WHITE']
…
```

Time stamp

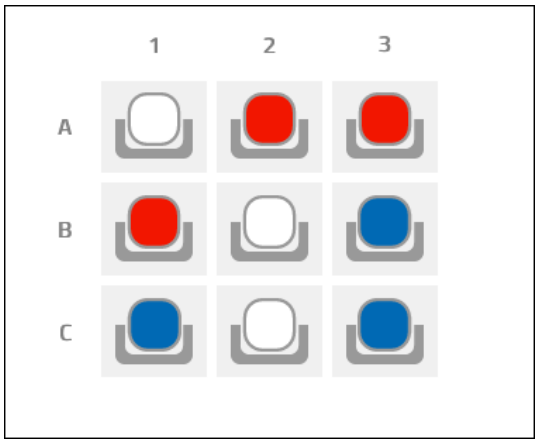
RFID

State

Work piece color

lernfabrik_hbw (high-bay warehouse)

#	ts	a1_id	a1_state	a1_type	a2_id	a2_state	a2_type	a3_id	a3_state	a3_type	b1_id	b1_state	b1_type	b2_id	b2_state	b2_type	b3_id	b3_state	b3_type	c1_id	c1_state	c1_type	c2_id	c2_state	c2_type	c3_id	c3_state	c3_type
1	2022-07-05 16:26:30.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
2	2022-07-05 16:26:30.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
3	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
4	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
5	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
6	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
7	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
8	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
9	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
10	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
11	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
12	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
13	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
14	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
15	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
16	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
17	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
18	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
19	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
20	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
21	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
22	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
23	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
24	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
25	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
26	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
27	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
28	2022-07-05 16:36:35.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
29	2022-07-05 16:36:35.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE



lernfabrik_stateorder (ordering state)

APPENDIX A

Contains ordering state

Enthält den Bestellstatus

```
CREATE TABLE lernfabrik_stateorder (
  ts datetime NOT NULL,
  type VARCHAR(5),
  state VARCHAR(20)
);
```

```
Zeitstempel
Bestellte Farbe ['BLUE'/'RED'/'WHITE']
Status ['WAITING_FOR_ORDER'/'ORDERED'/
        'IN_PROCESS'/'SHIPPED']
```

Time stamp
Ordered color
State

#	ts	type	state
1	2022-07-05 16:26:30.000		WAITING_FOR_ORDER
2	2022-07-05 16:31:56.000	RED	ORDERED
3	2022-07-05 16:33:02.000	RED	IN_PROCESS
4	2022-07-05 16:34:42.000	RED	SHIPPED
5	2022-07-05 16:36:35.000		WAITING_FOR_ORDER

Contains relevant data for the stations

Enthält stationsrelevante Daten

```
CREATE TABLE lernfabrik_stationen (
  ts datetime NOT NULL,
  station VARCHAR(3),
  code smallint,
  description VARCHAR(20),
  target VARCHAR(3),
  active BIT DEFAULT 'false'
);
```

```
Zeitstempel
Station (s. Tabelle unten)
Acknowledgment-Code (s. Tabelle unten)
Beschreibung
Zielstation (s. Tabelle unten)
Active-Flag [Aktiv=True; Inaktiv=False]
```

Time stamp
Station
Ack-Code
(see below)
Description
Target station
Active flag

lernfabrik_stationen (state of stations)

#	ts	station	code	description	target	active
1	2022-07-05 16:33:55.000	dsi	1			☐
2	2022-07-05 16:33:55.000	dso	1			☐
3	2022-07-05 16:33:55.000	hbw	1			☐
4	2022-07-05 16:33:55.000	mpo	1			☐
5	2022-07-05 16:33:55.000	sld	1			☑
6	2022-07-05 16:33:55.000	vgr	2			☐
7	2022-07-05 16:34:16.000	dsi	1			☐
8	2022-07-05 16:34:16.000	dso	1			☐
9	2022-07-05 16:34:16.000	hbw	1			☐
10	2022-07-05 16:34:16.000	mpo	1			☐
11	2022-07-05 16:34:16.000	sld	1			☐
12	2022-07-05 16:34:16.000	vgr	2		dso	☑

Acknowldgment-Codes:

Beschreibung	Station	Acknowledment-Codes
Delivery **S**tation **I**nput	**dsi**	0 = *Lichtschranke unterbrochen*
Delivery **S**tation **O**utput	**dso**	1 = *Lichtschranke nicht unterbrochen*
High-**B**ay **W**arehouse	**hbw**	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
Multi **P**rocessing Station with **O**ven	**mpo**	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
Sorting **L**ine with Color **D**etection	**sld**	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_PRODUCED
Vacuum **G**ripper **R**obot	**vgr**	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

6.3 OPC-Router

The OPC router writes the data to a Microsoft SQL server cluster in the university cloud. Here the metadata of the respective tables are described.

The following MSSQL tables are created for the OPC router:

Contains all values of the environmental sensor

Enthält alle Werte des Umweltsensors

```
CREATE TABLE
lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Sensorwert (gas resistance [Ohm])
Luftqualität Genauigkeit [0-3]
Luftqualität [index air quality 0-500]
Luftdruck [hPa]
Luftfeuchtigkeit [raw-value]
Luftfeuchtigkeit [%]
Temperatur [raw-value]
Temperatur [°C]
```

Time stamp
Sensor value
Air quality precision
Air quality index
Air pressure
Raw humidity
Humidity
Raw temperature
Temperature

lernfabrik_bme680 (environmental sensor)

ts	gr	aq	iaq	p	rh	h	rt	t
29.09.2021 09:56:07	185450	3	144	966,799987792969	49,5200004577637	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:16	186598	3	142	966,900024414063	49,5	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:28	185762	3	144	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:34	185867	3	142	966,900024414063	49,5099983215332	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:49	185762	3	143	966,799987792969	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:58	187554	3	141	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:57:07	187013	3	142	966,900024414063	49,5	60	25,5900001525879	22,3999996185303

APPENDIX A

Contains all values of the brightness sensor

Enthält alle Werte des Helligkeitssensors

```
CREATE TABLE
lernfabrik.lernfabrik_ldr (
  ts datetime NOT NULL,
  ldr SMALLINT NOT NULL,
  br DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Light Dependent Resistor (0-15000[Ohm])
Helligkeit [%] (wird aus ldr ermittelt)
[ldr: 15000 = 0%; 7500 = 50%; 0 = 100%]
```

Time stamp
Brightness is determined from ldr value

lernfabrik_ldr (brightness sensor)

ts	ldr	br
29.09.2021 09:56:04	4611	69,3000030517578
29.09.2021 09:56:16	4707	68,5999984741211
29.09.2021 09:56:28	4694	68,6999969482422
29.09.2021 09:56:34	4702	68,6999969482422
29.09.2021 09:56:49	4710	68,5999984741211
29.09.2021 09:56:58	4710	68,5999984741211
29.09.2021 09:57:07	4707	68,5999984741211

Contains information to the storage level of the 9 containers in the high-bay warehouse

Enthält Infos zum Lagerbestand der 9 Container im Hochregallager

```
CREATE TABLE lernfabrik_hbw (
  ts datetime NOT NULL,
  a1_id VARCHAR(14),
  a1_state VARCHAR(9),
  a1_type VARCHAR(5),
  a2_id VARCHAR(14),
  a2_state VARCHAR(9),
  a2_type VARCHAR(5),
  a3_id VARCHAR(14),
  a3_state VARCHAR(9),
  a3_type VARCHAR(5),
  b1_id VARCHAR(14),
  b1_state VARCHAR(9),
  b1_type VARCHAR(5),
  b2_id VARCHAR(14),
  b2_state VARCHAR(9),
  b2_type VARCHAR(5),
  b3_id VARCHAR(14),
  b3_state VARCHAR(9),
  b3_type VARCHAR(5),
  c1_id VARCHAR(14),
  c1_state VARCHAR(9),
  c1_type VARCHAR(5),
  c2_id VARCHAR(14),
  c2_state VARCHAR(9),
  c2_type VARCHAR(5),
  c3_id VARCHAR(14),
  c3_state VARCHAR(9),
  c3_type VARCHAR(5)
);
```

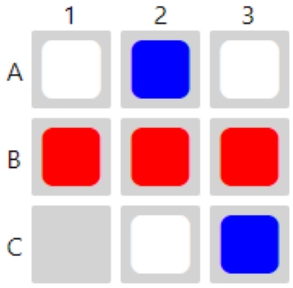
```
Zeitstempel
RFID ['123456789ABCDE']
Status ['RAW'/'PROCESSED']
Werkstückfarbe ['BLUE'/'RED'/'WHITE']
…
```

Time stamp
RFID
State
Work piece color

lernfabrik_hbw (high-bay warehouse)

ts	a1_id	a1_state	a1_type	a2_id	a2_state	a2_type	a3_id	a3_state	a3_type	b1_id	b1_state	b1_type	b2_id	b2_state	b2_type	b3_id	b3_state	b3_type	c1_id	c1_state	c1_type	c2_id	c2_state	c2_type	c3_id	c3_state	c3_type
18.02.2022 14:36:35	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	0			046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:25	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	0			046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:17	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:07	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:35:57	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:35:47	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	046c58a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE

APPENDIX A



Contains ordering state

Enthält den Bestellstatus

```
CREATE TABLE lernfabrik_stateorder (
  ts datetime NOT NULL,
  type VARCHAR(5),
  state VARCHAR(20)
);
```

```
Zeitstempel
Bestellte Farbe ['BLUE'/'RED'/'WHITE']
Status ['WAITING_FOR_ORDER'/'ORDERED'/
        'IN_PROCESS'/'SHIPPED']
```

Time stamp
Ordered color
State

lernfabrik_stateorder (ordering state)

ts	type	state
18.02.2022 14:36:25	BLUE	IN_PROCESS
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:33:00		WAITING_FOR_ORDER

Contains relevant data for the stations

Enthält stationsrelevante Daten

```
CREATE TABLE lernfabrik_stationen (
  ts datetime NOT NULL,
  station VARCHAR(3),
  code smallint,
  description VARCHAR(20),
  target VARCHAR(3),
  active BIT DEFAULT 'false'
);
```

```
Zeitstempel
Station (s. Tabelle unten)
Acknowledgment-Code (s. Tabelle unten)
Beschreibung
Zielstation (s. Tabelle unten)
Active-Flag [Aktiv=True; Inaktiv=False]
```

Time stamp
Station
Ack-Code
(see below)
Description
Target
station
Active flag

lernfabrik_stationen (State of the stations)

ts	station	code	description	target	active
18.02.2022 14:36:25	dso	1			False
18.02.2022 14:36:25	hbw	2			False
18.02.2022 14:36:25	mpo	2			True
18.02.2022 14:36:25	vgr	2		mpo	True
18.02.2022 14:36:24	dsi	1			False
18.02.2022 14:36:23	sld	1			False

Acknowledgment-Codes:

APPENDIX A

Station	Beschreibung	Acknowledgment-Codes
dsi	Delivery and Pickup Station (Input)	-
dso	Delivery and Pickup Station (Output)	-
hbw	High-Bay Warehouse (Hochregallager)	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
mpo	Multi Processing Station (Bearbeitung)	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
sld	Sorting Line (Sortierstation)	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_SORTED
vgr	Vacuum Gripper Robot (Sauggreifer)	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

APPENDIX A

CHAPTER 7 Module 7

Welcome to Module 7.

In Module 7 we will learn how to create a neural network using a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) and how to prepare the data for this neural network model.

Learning Objectives

- Preparation of the data so that TensorFlow can handle it.
- Creating and training a neural network model.
- Application of a trained model in a live environment.

Materials

- various Python scripts for data preparation
- TensorFlow CNN model
- TensorFlow RNN model

7.1 Neural Network Model as CNN and RNN with Tensorflow and Keras

We try to use the data produced by the learning factory by means of a neural network model to automatically determine the current state of the learning factory. We take the MQTT messages that are sent every two seconds for each topic as our data.

Unfortunately, it's not like we take a bunch of big data, in our case the messages, put it into the model and say we want to get a certain result. It's not that simple. The data has to be processed and cleaned beforehand so that the model can handle it at all.

7.2 System requirement

A Jupyter Notebook is required for training. In our case, it is an Ubuntu 20 machine with Jupyter Notebook already set up with all the necessary modules.

The Tensorflow module is easily installed with the pip command. Keras is already included.

```
pip install --upgrade tensorflow
```

7.3 Data collection

We first write a simple Python script with which we record all MQTT messages of the learning factory during an order and write them to a file. Then we don't have to permanently run an order when developing the model afterwards, but can always refer back to the recorded raw data.

The Python script is located on the kubernetes-master machine in the folder:

/home/lernfabrik/ki-campus/modul_7/mqtt_client_hs_threads_stati_raw.py

Below is the function that writes the messages to a file. The messages are already in JSON format. We just add the topic in front of it and save the whole thing. Because of the JSON format, we can then quickly and easily go through the data.

```
# prueft den Inhalt der Message, nur wenn neuer inhalt update
def check_message_payload(message):
  try:
      data = { 'topic': message.topic,
               'message': json.loads(message.payload) }

      if print_flag: print(message.topic)
      f_out.write(json.dumps(data) + "\n")

   except Exception as e:
      if print_flag: print("Exception check_message_payload()")
      raise e
### end check_message_payload()
```

We start the script and run an order in parallel. After the order has been completed and the workpiece has been put back into storage, we end the script.

The file that is created is called:

```
data_file="/home/lernfabrik/ki-
campus/modul_7/data_file_big_raw.json"
```

Example:

```
{"topic": "f/i/state/hbw",
"message": {"ts": "2022-04-06T18:43:06.453Z", "station": "hbw",
"code": 1, "description": "", "active": false, "target": ""}}
```

7.4 Data preparation first part

If we go into the raw data, we see that the messages are not sorted by timestamp. Below is an example:

```
{"topic": "f/i/state/vgr",
"message": {"ts": "2022-04-06T18:43:08.792Z", "station": "vgr",
"code": 1, "description": "", "active": false, "target": ""}}
{"topic": "f/i/state/sld",
"message": {"ts": "2022-04-06T18:43:07.542Z", "station": "sld",
"code": 1, "description": "", "active": false, "target": ""}}
```

This is probably due to the fact that the intervals between the topics are sometimes only a few milliseconds and the data therefore gets a bit mixed up when writing. But this is not a problem. We write a simple script which sorts the data correctly according to the timestamps.

The script is located on the kubernetes-master machine under Jupyterhub in the following folder:

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CrearteFileOrderedByTime.ipynb

The script reads all data into a list:

```
import numpy as np
import json
import pandas as pd

# Datei einlesen
data_file = "data_file_big_raw_blue_without_cam_new.json"
f_in = open(data_file, "r")
file = []

for line in f_in:
    line = line.strip()
    file.append(line)

f_in.close()

print(type(file))
print(len(file))
file
```

```
<class 'list'>
1107
['{"topic": "f/i/state/vgr", "message":
```

```
{"ts": "2022-04-06T13:30:17.130Z", "station": "vgr", "code": 1,
"description": "", "active": false, "target": ""}}',
'{"topic": "f/i/state/sld", "message":
{"ts": "2022-04-06T13:30:16.829Z", "station": "sld", "code": 1,
"description": "", "active": false, "target": ""}}',
'{"topic": "f/i/state/mpo", "message":
{"ts": "2022-04-06T13:30:17.121Z", "station": "mpo", "code": 1,
"description": "", "active": false, "target": ""}}' …]
```

Then we create a DataFrame with two columns and insert the data into the DataFrame and write the timestamp in the first column.

```
# DataFrame bilden
df = pd.DataFrame(columns=['Zeitstempel', 'Message'])

for line in file:
    dict_value = json.loads(line)
    df = df.append({'Zeitstempel': dict_value['message']['ts'],
'Message': dict_value}, ignore_index=True)
df
```

	Time stamp	**Message**
0	2022-02-06 15:43:42.470	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
1	2022-02-06 15:43:39.454	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
...	...	...

Now we can easily sort the date after the time stamp

```
# Nach Zeitstempel sortieren
df_sort = df.sort_values(by=['Zeitstempel'])
df_sort
```

	Time stamp	**Message**
23	2022-02-06 15:43:42.470	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
4	2022-02-06 15:43:39.454	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
...	...	...

The data is now sorted and can be written into a file. To do so we have to change the DataFrame into a simple list.

```
# Aus DataFrame wieder Array umwandeln in neu Datei schreiben
np_array = df_sort['Message'].to_numpy()

data_file_2 = "data_file_big_raw_blue_without_cam_new_ordered.json"
f_out = open(data_file_2, "w")

for line in np_array:
    f_out.write(json.dumps(line) + "\n")
f_out.close()
```

7.5 Data analysis

First we have to consider which messages are suitable to feed the model. The most suitable ones are the statuses that the stations themselves already send.

These are the following topics:

- f/i/state/vgr -> vacuum gripper robot
- f/i/state/sld -> sorting station
- f/i/state/mpo -> multi processing station
- f/i/state/hbw -> high-bay warehouse
- f/i/state/dsi -> input station
- f/i/state/dso -> output station

Let's first take a closer look at a message:

```
{"ts":"2022-03-18T13:26:42.391Z", "station":"dsi", "code":1,
"description":"", "target":"", "active":false}
```

We see the message has a timestamp, the name of the station, a code column, a description column, a target column and an active column. The columns ts, description and station are not interesting for us. The columns code, target and active are more interesting.

The model does not handle different data types very well. What also doesn't work well are too big distances between the input numbers e.g. 1 and 1000 or 0.001 and 1. Therefore we would have to think about how to normalize the values. The Code column can only take the states 0, 1, 2. The Active column can only accept "false" or "true", which we can convert to 0 and 1. Thus we see already at the first two columns, that small integer values would fit here best to our model. They already exist or can easily be converted to integer values.

Remains the column Target. We go into the raw data and see what values this column can take. The values are "" (empty), mpo, hbw, dso. These can also be easily converted to integer values to 0, 1, 2, 3. When looking through the raw data, we notice that the Target column is only set by the station vgr. Thus this column is omitted for all other stations.

Let's look at an intermediate state of the normalized input data.

Topic	Station	active (state)	code	target
f/i/state/vgr	Vacuum gripper robot	0,1	1,2	0,1,2,3
f/i/state/sld	Sorting line	0,1	1,2	
f/i/state/mpo	Multi processing station	0,1	1,2	
f/i/state/hbw	High-bay warehouse	0,1	1,2	
f/i/state/dsi	Input station	0,1	1,0	
f/i/state/dso	Output station	0,1	1,0	

From this you will get the following input data:

```
hbwstate, hbwcode, vgrstate, vgrcode, vgrtarget, mpostate, mpocode,
sldstate, sldcode, dsistate, dsicode, dsistate, dsocode
```

We have a total of 13 different states that we can take as input data. These are actually sufficient to create a first model. If they are not enough, we can add more topics to make our model more precise.

7.6 Data preparation second part

In the previous chapter we thought about what we can take as input data and determined these states. Now we have to create these pure states from the received MQTT messages and save them in a new file. With this we can finally create our model.

We accomplish this again with a Python script. This is located on kubernetes-master under the following path:
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CreateFileForNNTensorflow.ipynb
The function getIntToTarget() returns an integer value.

```
import json

# Multiple Topics Subscribe muessen zwingend eine Liste mit Tuples
sein
#topics = [("f/i/state/vgr", 0), ("f/i/state/sld", 0),
```

```
("f/i/state/mpo", 0), ("f/i/state/hbw", 0), ("f/i/state/dsi", 0),
("f/i/state/dso", 0)]
#targets = ["null", "mpo", "hbw", "dso"]
#          0       1     2       3

def getIntToTarget(target):
    if(len(target) == 0):
        return 0
    elif(target == "mpo"):
       return 1
    elif(target == "hbw"):
       return 2
    elif(target == "dso"):
        return 3
# end def
```

In the next section we define 13 variables. These are our input values. Then we define the cleaned and sorted file to be read and the new file to which the data will be written. First we write the column names into the new file.

```
try:
    hbwstate = -1
    hbwcode = -1

    vgrstate = -1
    vgrcode = -1
    vgrtarget = -1

    mpostate = -1
    mpocode = -1

    sldstate = -1
    sldcode = -1

    dsistate = -1
    dsicode = -1

    dsostate = -1
    dsocode = -1

    path = "/home/lernfabrik/ki-campus/modul_7/"
    data_file_in = path +
"data_file_big_raw_white_without_cam_new_ordered.json"
    f_in = open(data_file_in, "r")

    data_file_out = path + "data_file_big_raw_white_nn.csv"
    f_out = open(data_file_out, 'w+')

f_out.write('hbwstate,hbwcode,vgrstate,vgrcode,vgrtarget,mpostate,m
pocode,sldstate,sldcode,dsistate,dsicode,dsostate,dsocode\n')
```

We go through the individual topics of the file and let us always output all states at once in one line in the new file. However, only when one of the six topics comes up.

```
for line in f_in:
        json_value = json.loads(line)
        topic = json_value['topic']
        message = json_value['message']
        write_flag = False

        if (topic == "f/i/state/vgr"):
            vgrstate = 0 if message['active'] == Falseelse 1
            vgrcode = message['code']
            vgrtarget = getIntToTarget(str(message['target']))
            write_flag = True
        elif (topic == "f/i/state/sld"):
            sldstate = 0 if message['active'] == Falseelse 1
            sldcode = message['code']
            write_flag = True
        elif (topic == "f/i/state/mpo"):
            mpostate = 0 if message['active'] == False else 1
            mpocode = message['code']
            write_flag = True
        elif (topic == "f/i/state/hbw"):
            hbwstate = 0 if message['active'] == Falseelse 1
            hbwcode = message['code']
            write_flag = True
        elif (topic == "f/i/state/dsi"):
            dsistate = 0 if message['active'] == Falseelse 1
            dsicode = message['code']
            write_flag = True
        if (topic == "f/i/state/dso"):
            dsostate = 0 if message['active'] == Falseelse 1
            dsocode = message['code']
            write_flag = True

        if write_flag:
            line = str(hbwstate) + "," + str(hbwcode) + "," + \
                     str(vgrstate) + "," + str(vgrcode) + "," +
str(vgrtarget) + "," + \
                       str(mpostate) + "," + str(mpocode) + "," + \
                         str(sldstate) + "," + str(sldcode) + "," +
\
                            str(dsistate) + "," + str(dsicode) + ","
+ \
                              str(dsostate) + "," + str(dsocode) +
",\n"
            f_out.write(line)
            #print(message)
            #print(line)
    f_in.close()
    f_out.close()
except Exception as e:
```

```
    f_in.close()
    f_out.close()
    raise e
```

The result looks something like this. We see how the topics run in and gradually change from "-1" to the correct state. What should be immediately noticeable is that the entry "0,1,0,1,0,0,1,0,1,0,1," occurs very frequently. This seems to be the default state of the learning factory.

hbwstate,hbwcode,vgrstate,vgrcode,vgrtarget,mpostate,mpocode,sldstate,sldcode,dsi state,dsicode,dsostate,dsocode

-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,1,

-1,-1,-1,-1,-1,-1,0,1,-1,-1,0,1,

-1,-1,-1,-1,-1,-1,-1,0,1,0,1,0,1,

-1,-1,-1,-1,-1,0,1,0,1,0,1,0,1,

-1,-1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1, ...

With this data, we can build our model. We first want to create a simple CNN (Convolutional Neural Network) model. This will be computed from the current states only.

7.7 Data preparation third part

That means we first try to find all possible states and if possible reduce them to a very short list so that we have a better overview of the data. For this we can take out all duplicate entries that are between the states themselves. We can do this manually, but again to avoid errors we write a simple Python script that does the work for us.

The script is located on kubernetes-master under the following path:

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CreateFileForCNNTensorflow.ipynb

So we read the data back in and remove duplicate entries in the state transitions. The script is self-explanatory. It omits the first line with the columns. Then it always compares the current row with the previous one. If they differ it is written to a file. Finally the last status is also saved.

```
path = "/home/lernfabrik/ki-campus/modul_7/"
data_file = path + "data_file_big_raw_white_nn.csv"
f_in = open(data_file, "r")

data_file = path + "data_file_white_for_cnn.csv"
f_out = open(data_file, "w")

i=0
line_bevore = ""

for line in f_in:
    if(i == 0):
        i = i + 1
        continue
    if(i == 1):
        line_bevore = line

    if(line != line_bevore):
        print(i)
        print(line_bevore)
        f_out.write(line_bevore)
        line_bevore = line

    i = i + 1

print(line)
f_out.write(line)

f_in.close()
f_out.close()
```

Now, out of the 653 states, only 40 remain. Let's take a closer look at these numbers. We can delete the lines with "-1" manually. This is the initialization of the state variables, which is completed after the sixth station.

```
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,1,
-1,-1,-1,-1,-1,-1,-1,0,1,-1,-1,0,1,
-1,-1,-1,-1,-1,-1,-1,0,1,0,1,0,1,
-1,-1,-1,-1,-1,0,1,0,1,0,1,0,1,
-1,-1,0,1,0,0,1,0,1,0,1,0,1,
```

Now we have to consider which states the learning factory can assume and assign them to the respective line. We have to do this classification once otherwise the

model does not know which changes lead to which states. The first and the last one is simple. This is our basic state, we call it "Ruhend" (“Resting”). What is also easy to see is the output station in the middle where the last state changes from 1 to 0 in two places. The second line must be picking. This is because when an order is placed, the component must first be removed from storage. Using the targets, we can also define some states quite precisely. So, based on the code changes, we try to determine the states little by little.

The following is the finished data:

```
0,1,0,1,0,0,1,0,1,0,1,0,1,Ruhend
0,1,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
0,2,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,2,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,1,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,1,0,2,0,0,2,0,1,0,1,0,1,Bearbeitung
0,2,0,2,0,0,2,0,1,0,1,0,1,Bearbeitung
0,2,1,2,1,0,2,0,1,0,1,0,1,TransportToMPO
0,2,1,2,1,1,2,0,1,0,1,0,1,TransportToMPO
0,2,0,2,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,2,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,0,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,0,2,1,2,0,1,0,1,Sortierung
0,1,0,1,0,0,1,1,2,0,1,0,1,Sortierung
0,1,0,2,0,0,1,1,2,0,1,0,1,Sortierung
0,1,0,2,0,0,1,1,1,0,1,0,1,Transport
0,1,0,2,0,0,1,0,1,0,1,0,1,Transport
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,0,1,0,0,AusgabeStation
0,1,1,2,3,0,1,0,1,0,1,1,0,AusgabeStation
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,1,0,0,1,TransportToDSO
0,1,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,1,2,2,0,1,0,1,1,0,0,1,TransportToHBW
0,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
0,1,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,0,2,0,0,1,0,1,0,1,0,1,Einlagerung
1,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
1,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,1,0,1,0,0,1,0,1,0,1,0,1,Ruhend
```

7.8 Data Preparation fourth part

Now we have to check if there are states that could belong to more than one category. This is done again with a script. All states are read in. During the reading we check if this state is already assigned to a category. If not then it is written into the DataFrame. If the key already exists, an output is made. The script is located under jupyterhub-gpu at.

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/create_file_for_rnn_all.ipynb

```
import numpy as np
import json
import pandas as pd

# Datei einlesen
data_file = "data_file_white_for_cnn.csv"
f_in = open(data_file, "r")

# DataFrame bilden
df_cat = pd.DataFrame(columns=['Kategorie', 'Key'])

# Kategorien durchgehen, doppelte auslassen
for line in f_in:
    cat = line[26:].strip()
    key = line[:26]

    flag = True
    for df in df_cat.values:
        if key == df[1]:
            print("Kategorie: " + cat + " vorhanden in " + df[0] +
" bei Zustand: " + key)
            flag = False
            break

    if flag:
        df_cat = df_cat.append({'Kategorie': cat, 'Key': key},
ignore_index=True)

f_in.close()
```

We see in the output that the storage in (Einlagerung) and storage out (Auslagerung) use identical states. Furthermore, there is a hit on storage out and transport.

```
Kategorie: Transport vorhanden in Auslagerung bei Zustand:
0,1,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: TransportToDSO vorhanden in TransportToDSO bei Zustand:
0,1,1,2,3,0,1,0,1,0,1,0,1,
```

```
Kategorie: Einlagerung vorhanden in Auslagerung bei Zustand:
1,2,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: Einlagerung vorhanden in Einlagerung bei Zustand:
1,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Einlagerung vorhanden in Einlagerung bei Zustand:
0,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ruhend vorhanden in Ruhend bei Zustand:
0,1,0,1,0,0,1,0,1,0,1,0,1,
```

We can combine the categories " Storage in" and "Storage out" into one category "Storage in/out". In case of storage out and transport, the combination does not make sense. Furthermore, we see that transportation occurs approximately in the middle of the order. It would not make sense to define the " Storage out " state in the middle of the order. So we change " Storage out" at the beginning to "Transport". We do this manually and save it in a new file.

/home/learning_factory/ki-campus/modul_7/data_file_white_for_cnn_new_kategory.csv

We run the script over the new file. The output shows that there are no more states assigned to more than one category.

```
Kategorie: Transport vorhanden in Transport bei Zustand:
0,1,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: TransportToDSO vorhanden in TransportToDSO bei Zustand:
0,1,1,2,3,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 1,2,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 1,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 0,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ruhend vorhanden in Ruhend bei Zustand:
0,1,0,1,0,0,1,0,1,0,1,0,1,
```

7.9 Data preparation the fifth part

As the last step in the data preparation we only have to insert the just defined categories into the original input file. For this we can take the previous script create_file_for_rnn_all.ipynb and extend it only by the part of the category writing.

```
ef getCatToKey(key):
    for df in df_cat.values:
        if key == df[1]:
            return df[0]

# Datei einlesen
data_file = "data_file_big_raw_white_nn.csv"
f_in = open(data_file, "r")

# Datei schreiben
data_file = "data_file_big_raw_white_nn_kategory.csv"
f_out = open(data_file, "w")

for line in f_in:
    new_line = line.strip() + str(getCatToKey(line.strip()))
    f_out.write(new_line + "\n")

f_in.close()
f_out.close()
```

The new data is now in:
/home/lernfabrik/ki-campus/modul_7/data_file_big_raw_white_nn_kategory.csv
Finally we can start to create the actual model.

7.10 CNN - Introduction

After the long data acquisition and data preparation, we finally come to the creation of the model. As a basis for this, we are guided by the example keras_iris_klassification from the book Deep Learning with TensorFlow, Keras and TensorFlow.js.
We will not go into the basic concepts of AI or neural networks here.
The completed model can be viewed on the kubernetes-master engine.
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/lernfabrik_model_mqtt_cnn_new_kategory.ipynb

7.11 CNN - Load and provide data

We load the data into a Numpy array. We see each of the individual values is automatically recognized.

```
# Laden der aufbereiteten Daten
stati_data = np.loadtxt("data_file_white_for_nn_kategory.csv", \
    delimiter=",", dtype="str", skiprows=0)
stati_data
```

```
array([['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ...,
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend']], dtype='<U16')
```

As we already know, Tensorflow doesn't handle different data types very well. The stats are strings so we can read them more easily. For our model, however, we need to convert them to integers.

For this we simply define an array with all our stats and replace the string values in the stati_data with integer values starting at 0. Finally we define that the whole array consists of integer values. The integers that are still stored as strings will be converted to integers.

```
# Wir erstellen die Kategorien:
stati_label_array = ["Auslagerung","TransportToMPO","Bearbeitung",
\
    "Sortierung", "AusgabeStation","Transport","TransportToDSO", \
    "TransportToHBW", "Einlagerung", "Ruhend"]
label_index = 0
for label in stati_label_array :
stati_data[np.where(stati_data[:,13]==label),13] = label_index
    label_index = label_index + 1

stati_data = stati_data.astype("int")
stati_data
```

```
array([[0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       ...,
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8]])
```

Columns 1-13 are our states and therefore the input data. We create a new list in which we only extract these columns.

```
# Aufteilen der Daten in Input...
input_data = stati_data[:,0:13] # Spalten 0 bis 13 werden extrahier
input_data
```

```
array([[0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       ...,
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1]])
```

The last column with the categories that we just turned into integer values is our output data. We create a new list and extract only the last column with the categories into it.

```
# ... und Output
output_data = stati_data[:,13]
output_data
```

```
array([8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
5, 5,5, 5, 5, 0, 0, 0...
```

We have nine categories defined, but only a simple list. Tensorflow cannot handle this. Tensorflow works with matrices. Therefore we have to transform the list. For this there is a function to_categorical(), this turns the simple list into a 1D array. We see it consists of 10 columns that are the categories and depending on which category it is it is set to 1 and all other columns are set to 0.

```
output_data = to_categorical(output_data)
output_data
```

```
array([[0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Now we have our input data as well as the output data. We now need to divide this into training data and evaluation data. In AI, a ratio of 4 to 1 has become common. That means 80% training data and 20% evaluation data. Keras already has a function for this. We give it our data as a transfer parameter, as well as the size of the training data and receive this divided back.

```
# Splitting des Datasets CNN
# Wir splitten in train und evaluation Daten
# 80% train und 20% Evaluation
stati_train_input, stati_test_input, \
stati_train_output, stati_test_output =
train_test_split(input_data, output_data, test_size=0.20)
```

7.12 CNN - Model Definition

Let's move on to the definition of our model. It is a sequential model. We will use "Dense" as the layer. The model has 13 input neurons, because 13 states. However, it is defined with 14. The last neuron is a slide neuron. This means that it always fires a random value. This is necessary so that a model does not over-specify. We add a hidden layer with 10 neurons in between and as output we also add 10 neurons, these are our output categories. But here we take "softmax" instead of "relu2" as activation function. Finally we output a summary of the model with the function summary(). With a total of 456 parameters, the model is one of the smaller ones.

```
# Aufbau des Modells mit Keras Convolutional CNN
stati_model = Sequential()
# Eingabeschicht 13 Neuronen + 1 Dias Neuron
stati_model.add(Dense(14,input_shape=(13,),activation="relu"))
stati_model.add(Dense(9,activation="relu"))
# Ausgabeschicht: 9 Zustaende
stati_model.add(Dense(9,activation="softmax"))
stati_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 14)                196

 dense_1 (Dense)             (None, 9)                 135
```

```
 dense_2 (Dense)             (None, 9)                 90

=================================================================
Total params: 421
Trainable params: 421
Non-trainable params: 0
_________________________________________________________________
```

The model is defined and we can now train it. With the compile function we configure our model. As "loss" function we take "categorical_crossentropy". We change the optimizer from SGD to the standard optimizer, the Adam, because we achieve better values with our model. Under "metrics" we indicate that we expect "categorical" as output, i.e. a unique result for one of the ten categories. In this case, "mae" means Mean Absolute Error.

```
# CNN
stati_model.compile(loss="categorical_crossentropy",
optimizer='adam', \
    metrics=["accuracy",tf.keras.metrics.mae]) # categorical
```

7.13 CNN - Model Training

With the function fit() we start the training of the model. We give the input data and the output data. As "batch-size" we take the common default1 value of 32. Under Epochs we choose 1000. Since we have a good GPU this should not be a problem. Also we want to have a closer look at the status output by doing this. With the parameter "verbose" we can output the status output during training.

We can re-train the model a few times, and will always get similar results. From the 100th epoch the accuracy is > 99%. The mean_absolute_error is <2%. From the 200th epoch the accuracy is 100% and the mean_absolute_error is 0%. From the 200th epoch on, no improvement of the model is possible. Better values cannot be achieved. This is probably due to the very precisely defined states that do not allow any room for multiple interpretations.

```
stati_model.fit(x=stati_train_input, y=stati_train_output,
batch_size=32, epochs=1000, verbose=1)
```

```
Epoch 1/1000
```

```
17/17 [==============================] - 1s 2ms/step - loss: 2.1947
- accuracy: 0.0494 - mean_absolute_error: 0.1961
Epoch 2/1000
17/17 [==============================] - 0s 2ms/step - loss: 2.0551
- accuracy: 0.1863 - mean_absolute_error: 0.1926
Epoch 3/1000
17/17 [==============================] - 0s 2ms/step - loss: 1.9491
- accuracy: 0.3365 - mean_absolute_error: 0.1891
. . .
Epoch 100/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0894
- accuracy: 0.9924 - mean_absolute_error: 0.0150
Epoch 101/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0872
- accuracy: 0.9924 - mean_absolute_error: 0.0146
Epoch 102/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0853
- accuracy: 0.9924 - mean_absolute_error: 0.0144
. . .
17/17 [==============================] - 0s 2ms/step - loss: 0.0108
- accuracy: 0.9981 - mean_absolute_error: 0.0022
Epoch 212/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 1.0000 - mean_absolute_error: 0.0022
Epoch 213/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 1.0000 - mean_absolute_error: 0.0022
...
Epoch 998/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.3315e-06 - accuracy: 1.0000 - mean_absolute_error: 2.9760e-07
Epoch 999/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.3045e-06 - accuracy: 1.0000 - mean_absolute_error: 2.9208e-07
Epoch 1000/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.2961e-06 - accuracy: 1.0000 - mean_absolute_error: 2.8876e-07
```

7.14 CNN - Model tests

We test our model with the test data and get what we have already seen in the intermediate output. We get an accuracy of 100% with a mean_absolute_error of 0%.

```
# Evaluation auf Test Daten
evaluation_results = stati_model.evaluate(stati_test_input,
stati_test_output)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 1.2138e-
06 - accuracy: 1.0000 - mean_absolute_error: 2.7068e-07
```

We want to run our own test and define some custom states for the test. For this we simply take some states from our input file and write them into a numpy array. With the function predict() we can output the predictions. And with the function argma()x from Numpy the highest value of these.

```
# Eigener Test
test_data = np.array([[0,1,0,1,0,0,1,1,2,0,1,0,1], # Sortierung
                      [0,1,1,2,3,0,1,0,1,0,1,0,1], # TransportToDSO
                      [1,2,0,2,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [0,2,0,1,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [0,1,1,2,3,0,1,0,1,0,1,0,0], # AusgabeStation
                      [0,1,0,1,0,0,1,0,1,0,1,0,1], # Ruhend
                      [0,2,0,2,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [1,2,0,2,0,0,1,0,1,0,1,0,1] ]) # Ein-
/Auslagerung
predictions = stati_model.predict(test_data)
index_max_predictions = np.argmax(predictions,axis=1)
```

We iterate through the predictions and output the highest calculated category. Our own test confirms the very good result.

```
# Eigener Test Ausgabe
for i, e in enumerate(index_max_predictions):
    print("Datenreihe {} ergibt den Zustand: {}".format(
    test_data[i],
    stati_label_array[e]))
```

```
Datenreihe [0 1 0 1 0 0 1 1 2 0 1 0 1] ergibt den Zustand:
Sortierung
Datenreihe [0 1 1 2 3 0 1 0 1 0 1 0 1] ergibt den Zustand:
TransportToDSO
Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [0 2 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [0 1 1 2 3 0 1 0 1 0 1 0 0] ergibt den Zustand:
AusgabeStation
Datenreihe [0 1 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ruhend
Datenreihe [0 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
```

Thus, we have successfully created the model for status detection. We save the model so that we can use it in another script.

```
# Modell speichern
stati_model.save("lernfabrik_model_cnn_saved.h5")
```

With that, we are done with this model. for completeness, we take a look at a RNN Model (Recurrent Neuronal Network)

7.15 RNN - Introduction

With a CNN, only the state matters. With an RNN, the previous states can be included in the prediction. This leads to a more accurate and faster result. However, we need to make a lot of adjustments to our model.
The definition of the categories is identical to that of the CNN model. The output data as well.

7.16 RNN - Data preparation

The model is stored under the following path.
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/lernfabrik_model_mqtt_rnn.ipynb
In RNN, however, we need to further adjust our input data. First, we determine how many states - including the current one - should be considered. That means, if we want to consider two previous states, we take two. The reorganization is done by the function arrange_data(). It goes through all entries of the given data and writes the defined number of previous values to the current state into a separate array. This is stored as a multidimensional array.

```
ZUSTAND_BEFORE = 2

# Reorganisiert die Daten fuer RNN
def arrange_data(data, days):
    days_before_values = [] # T - days
    days_values = []  # T
    for i in range(len(data) - days):
        days_before_values.append(data[i:(i+days)])
```

```
        days_values.append(data[i + days])
    return np.array(days_before_values),np.array(days_values)

days_before_values_input, days_values_input =
arrange_data(input_data,ZUSTAND_BEFORE)
days_before_values_output, days_values_output =
arrange_data(output_data,ZUSTAND_BEFORE)
```

In the sample output we see the array days_values_input which contains the actual states. It is a one-dimensional array.

```
print(len(days_values_input))
print(days_values_input)
```

```
656
[[0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 ...
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]]
```

And the multidimensional array days_before_values_input that always has two states for each entry, because we have chosen two states that should be included in the prediction.

```
print(len(days_before_values_input))
print(days_before_values_input)
```

```
656
[[[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]
...

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]]
```

Splitting of the data works the same.

```
# Splitting des Datasets RNN
X_train, X_test, Y_train, Y_test = \
train_test_split(days_before_values_input, days_values_output,
test_size=0.20)
```

7.17 RNN - Model Definition

In an RNN we have to use a different layer for the input. This is called LSTM (Long short-term Memory). Also, for input_shape we have to keep in mind that we now have a multidimensional array as input. For the probability output of our categories we stay with Dense. We output information about the model. We see that with the LSTM layer alone, our model parameters have almost quadrupled compared to CNN.

```
# Aufbau des Modells mit Keras RNN
stati_model = Sequential()
# Eingabeschicht 13 Neuronen + 1 Dias Neuron | + 2 vorherige
Zustände
stati_model.add(LSTM(14, return_sequences=False,
input_shape=(ZUSTAND_BEFORE, 13)))
# Ausgabeschicht: 9 Wahrscheinlichkeitswerte fuer jeden der 9
Zustaende
stati_model.add(Dense(9))
stati_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 14)                1568

 dense_1 (Dense)             (None, 9)                 135

=================================================================
Total params: 1,703
Trainable params: 1,703
Non-trainable params: 0
_________________________________________________________________
```

7.18 RNN - Model Training

In model training there is no difference to CNN. The result already looks very good after the 50th epoch. We let the model train a few times always with the same result accuracy at >98%. The mean_absolute_error however remains between 1-2%. In fact, for our model the CNN seems to fit better. As we have already noticed with the CNN, we are dealing with well-defined states and a simple CNN already finds the solution very quickly.

```
stati_model.fit(x=stati_train_input, y=stati_train_output,
batch_size=32, epochs=1000, verbose=1)
```

```
17/17 [==============================] - 2s 3ms/step - loss: 0.1484
- accuracy: 0.2691 - mean_absolute_error: 0.2450
Epoch 2/500
17/17 [==============================] - 0s 3ms/step - loss: 0.1115
- accuracy: 0.4580 - mean_absolute_error: 0.1932
Epoch 3/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0968
- accuracy: 0.4714 - mean_absolute_error: 0.1765
. . .
Epoch 50/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0107
- accuracy: 0.9656 - mean_absolute_error: 0.0512
Epoch 51/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 0.9656 - mean_absolute_error: 0.0505
Epoch 52/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0102
- accuracy: 0.9656 - mean_absolute_error: 0.0492
. . .
Epoch 498/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0039
- accuracy: 0.9828 - mean_absolute_error: 0.0169
Epoch 499/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0038
- accuracy: 0.9828 - mean_absolute_error: 0.0169
Epoch 500/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0038
- accuracy: 0.9828 - mean_absolute_error: 0.0158
```

7.19 RNN – Model tests

The test data shows a good result.

```
evaluation_results = stati_model.evaluate(X_test, Y_test)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.0060 -
accuracy: 0.9697 - mean_absolute_error: 0.0168
```

Let's try our own data for testing. Here we have to be careful to pass a multidimensional array as well. We choose two consecutive states from our input file. The result should be the state after the two selected.

```
# Eigener Test
test_data = np.array([ [ [0,1,0,1,0,0,2,1,2,0,1,0,1],      #
(Sortierung) -2
                         [0,1,0,1,0,0,2,1,2,0,1,0,1], ],   #
(Sortierung) -1
                         #0,1,0,1,0,0,2,1,2,0,1,0,1,
Sortierung   0

                       [ [0,1,1,2,3,0,1,0,1,0,1,0,1],    #
(TransportToDSO) -2
                         [0,1,1,2,3,0,1,0,1,0,1,0,1], ], #
(TransportToDSO) -1
                         #0,1,1,2,3,0,1,0,1,0,1,0,1,
TransportToDSO   0

                       [ [0,2,0,2,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                         [0,2,0,2,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                         #0,2,0,2,0,0,1,0,1,0,1,0,1,        Ein-
/Auslagerung   0

                       [ [1,2,0,1,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                         [0,2,0,1,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                         #1,2,0,1,0,0,1,0,1,0,1,0,1,        Ein-
/Auslagerung   0

                       [ [0,1,1,2,3,0,1,0,1,0,1,0,0],    #
(AusgabeStation) -2
                         [0,1,1,2,3,0,1,0,1,0,1,1,0], ], #
(AusgabeStation) -1
                         #0,1,1,2,3,0,1,0,1,0,1,1,0,
AusgabeStation   0
```

```
                        [ [0,1,0,1,0,0,1,0,1,0,1,0,1],    # (Ruhend)
-2
                          [0,1,0,1,0,0,1,0,1,0,1,0,1], ], # (Ruhend)
-1
                          #0,1,0,1,0,0,1,0,1,0,1,0,1,
Ruhend   0

                        [ [0,2,0,1,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                          [0,2,0,1,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                          #0,1,0,1,0,0,1,0,1,0,1,0,1,        Ruhend

                        [ [1,2,1,2,2,0,1,0,1,0,1,0,1],     #
(TransportToHBW) -2
                          [1,2,1,2,2,0,1,0,1,0,1,0,1], ]]) #
(TransportToHBW) -1
                          #1,2,0,2,0,0,1,0,1,0,1,0,1,         Ein-
/Auslagerung 0

predictions = stati_model.predict(test_data)
index_max_predictions = np.argmax(predictions,axis=1)
```

We see, that the RNN is really very good, but it has its Problems with the category transitions.

```
for i, e in enumerate(index_max_predictions):
    print("Datenreihe {} ergibt den Zustand: {}".format(
    test_data[i],
    stati_label_array[e]))
```

```
Datenreihe [[0 1 0 1 0 0 2 1 2 0 1 0 1]
 [0 1 0 1 0 0 2 1 2 0 1 0 1]] ergibt den Zustand: Sortierung
Datenreihe [[0 1 1 2 3 0 1 0 1 0 1 0 1]
 [0 1 1 2 3 0 1 0 1 0 1 0 1]] ergibt den Zustand: TransportToDSO
Datenreihe [[0 2 0 2 0 0 1 0 1 0 1 0 1]
 [0 2 0 2 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[1 2 0 1 0 0 1 0 1 0 1 0 1]
 [0 2 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[0 1 1 2 3 0 1 0 1 0 1 0 0]
 [0 1 1 2 3 0 1 0 1 0 1 1 0]] ergibt den Zustand: AusgabeStation
Datenreihe [[0 1 0 1 0 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ruhend
Datenreihe [[0 2 0 1 0 0 1 0 1 0 1 0 1]
 [0 2 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[1 2 1 2 2 0 1 0 1 0 1 0 1]
 [1 2 1 2 2 0 1 0 1 0 1 0 1]] ergibt den Zustand: TransportToHBW
```

Now that we have looked at both possibilities, we want to apply one of the models during an order process. This is part of the next chapter.

7.20 Model application in live mode

For the model application we need our saved model from chapter 3 or 4. Furthermore we need the source code from chapter 7.1.3. (data acquisition) and the source code from chapter 7.1.4. (data preparation). We need to combine these two scripts. For one, we need to receive the MQTT messages, convert them to the states that the neural network model can work with, and then make a prediction from them.
The finished script is on kubernetes-master at:
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/Lernfabrik_MQTT_Stati_Modell_Load.ipynb
This sounds more complicated than it is. We just need to put the data preparation in the check_message_payload() function, as well as the prediction. The script for data retrieval is already running in an infinite loop. As soon as a new state message comes, the prediction is made and the state is output.

```
def check_message_payload(message):
    try:
        # Zeit tauschen zum vergleichen
        global hbwstate,hbwcode, \
             vgrstate,vgrcode,vgrtarget, \
             mpostate,mpocode, \
             sldstate,sldcode, \
             dsistate,dsicode, \
             dsostate,dsocode #, nfc

        if print_flag: print("\nNeue Message mit Topic: " +
str(message.topic))
        mess_dict = json.loads(message.payload.decode("utf-8"))

        if (message.topic == "f/"+s_i_let+"/state/vgr"):
            vgrstate = 0 if mess_dict['active'] == False else 1
            vgrcode = mess_dict['code']
            vgrtarget = getIntToTarget(str(mess_dict['target']))
        elif (message.topic == "f/"+s_i_let+"/state/sld"):
            sldstate = 0 if mess_dict['active'] == False else 1
            sldcode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/mpo"):
            mpostate = 0 if mess_dict['active'] == False else 1
            mpocode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/hbw"):
            hbwstate = 0 if mess_dict['active'] == False else 1
```

```
            hbwcode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/dsi"):
            dsistate = 0 if mess_dict['active'] == False else 1
            dsicode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/dso"):
            dsostate = 0 if mess_dict['active'] == False else 1
            dsocode = mess_dict['code']
        #elif (message.topic == "f/i/nfc/ds"):
            #nfc = 0 if mess_dict['workpiece'] == None else 1

        # Stati durch Modell und deren Wahrscheinlichkeiten
anzeigen lassen
        data = np.array([[int(hbwstate),int(hbwcode),

int(vgrstate),int(vgrcode),int(vgrtarget),
                         int(mpostate),int(mpocode),
                         int(sldstate),int(sldcode),
                         int(dsistate),int(dsicode),
                         int(dsostate),int(dsocode),]
                         #int(nfc),], # aktuellste Datenreihe
                         ])
        predictions = model.predict(data)
        index_max_predictions = np.argmax(predictions,axis=1)

        for i, e in enumerate(index_max_predictions):
            print("aktualisierte Datenreihe {} ergibt den Zustand:
{}".format(
            data[i],
            stati_label_array[e]))

    except Exception as e:
        if print_flag: print("Exception check_message_payload()")
        raise e
### end check_message_payload()<
```

When we start the script in the idle mode of the learning factory, only the "Idle" status is always displayed. We need to run an order to see a change in the output. It looks something like this:

```
. . .
Neue Message mit Topic: f/s/state/mpo
aktualisierte Datenreihe [1 2 1 2 2 0 1 0 1 0 1 0 1] ergibt den
Zustand: TransportToHBW

Neue Message mit Topic: f/s/state/hbw
aktualisierte Datenreihe [1 2 1 2 2 0 1 0 1 0 1 0 1] ergibt den
Zustand: TransportToHBW

Neue Message mit Topic: f/s/state/vgr
aktualisierte Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den
Zustand: Ein-/Auslagerung
```

```
Neue Message mit Topic: f/s/state/dsi
aktualisierte Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den
Zustand: Ein-/Auslagerung
. . .
```

The model is ready and works in live mode. We can now add more topics to further refine the categories. For example, the topic "f/i/nfc/ds" which manages the readout on the NFC chip.

7.21 Exercise

Apply the neural network model during an order.