Frankfurt University of Applied Sciences

Semester Project
(Cloud Computing)
WS 2020/21

Edge-Computing Framework (EdgeX)

Referent: Prof. Dr. Christian Baun

Submitted by: Jan Wagner (1290974)
jan.wagner2@stud.fra-uas.de

Daniel Helmer (1101678)
helmer@stud.fra-uas.de

Dominic Gibietz (1100239)
gibietz@stud.fra-uas.de

Date of submission: 22nd Jan, 2021

Contents

1 Introduction

EdgeX Foundry is an open source software framework at the edge of the network that connects all kind of *Internet of Things* (IoT) devices and interacts with them. It is serving between physical "things" and applications or clouds. With that, EdgeX makes it easy to monitor IoT devices, collect data from devices or send instructions to them, move the data to a cloud to store or analyze them and more. EdgeX provides multiple protocols to connect IoT devices, like *MQTT*, *REST* or *BLE*. It allows to encrypt, transform, filter or format the data before forwarding it to an external source over different protocols, like MQTT. Data is usually not stored in the *EdgeX Gateway* itself for a long time. EdgeX consists of several microservices, some of which can be optionally switched on or off. For example, rules can be created that automatically execute an action when the rule is fulfilled (if-then). [3, 4]

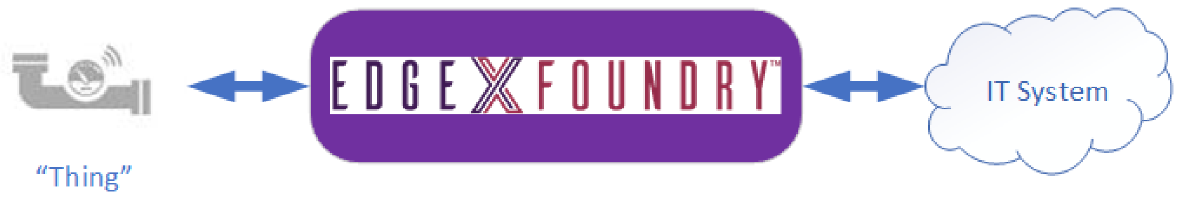


Figure 1: The EdgeX gateway between the "things" and the IT-System. Source: [3]

2 Foundation

2.1 EdgeX and the Internet of Things

The purpose of the so called *Internet of Things* (IoT) is to connect sensor (actor) networks with the internet. Doing so makes them easily accessible from anywhere and anytime. The "things" in the term Internet of things can be virtual or physical things. Many specialized versions of the IoT exist, for instance the *Industrial Internet of Things* (IIoT), the *Artificial Intelligence Internet of Things* (AIIoT) or the *Internet of Robotic Things* (IoRT). In all variations, it remains a common concept to connect the local (sensor)network via a gateway with the internet. The general structure of the IoT is shown in figure 2. Often the edge devices are not more then smart sensors. Since these have limited resources, remote resources need to be used. [5]

As mentioned, EdgeX Foundry is meant to connect a network of physical edge devices

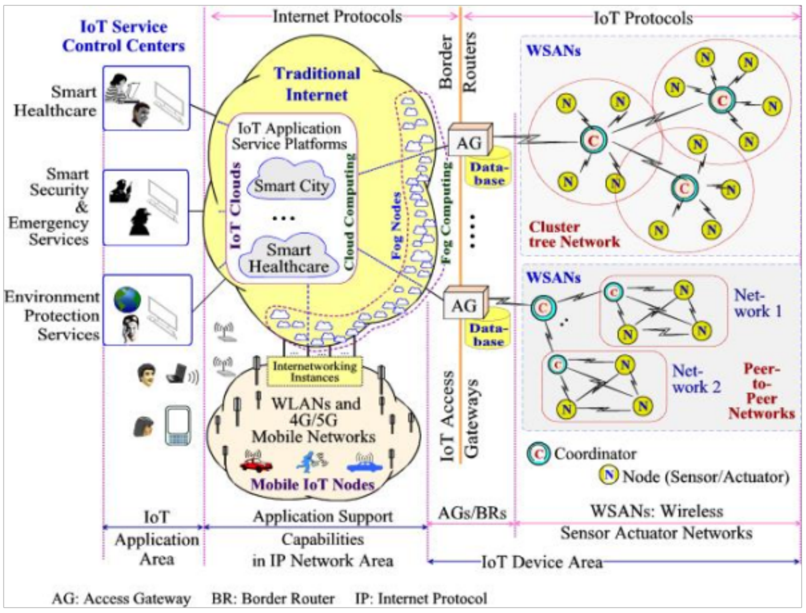


Figure 2: "General technical concept of the IoT" [5]

with the internet. Therefore it basically acts as the gateway in the IoT infrastructure. In the EdgeX Foundry documentation the communication with the physical nodes is called "south end" or "south side" communication and the communication with the web services "north end" or "north side" communication. Furthermore this nomenclature will be used in this document. [3]

2.2 EdgeX Foundation Services

EdgeX Foundry is a collection of open source micro services, which are organized into 4 layers: [3]

- Core Services Layer
- Supporting Services Layer
- Application Services Layer
- Device Services Layer

All layers are "passed through" from the south (*device layer*), through the *core* and *support layer*, to the north (*application layer*). Also there are two underlying services for *Security* and *System Management*. To start specific services with docker, the *docker compose service names* are needed, which can be found in the official documentation. [6] The most important services are described below.

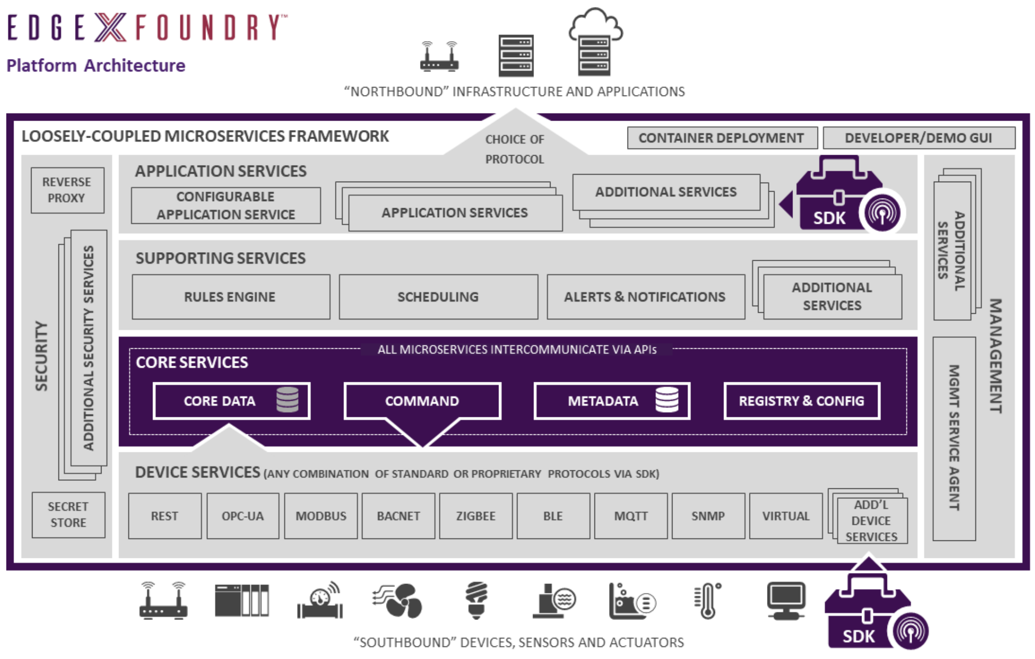


Figure 3: Visualisation of the EdgeX platform architecture. Source: [7]

2.2.1 Core Services Layer

These services are, as the name implies, the "core" to the EdgeX functionality. Here resides the EdgeX configuration, the collected sensor data and the knowledge of the connected "things". [8]

- **Data:** Stores the sensor data on the edge system until data gets moved "north". The open source in-memory structure store "redis" (see section 2.2.5) is used for data storage. Other services (within and outside of EdgeX) access the sensor data only through the core data service. [8]
- **Command:** Enables the communication of commands or actions to the devices. Requests can be sent from north to south. Allows GET (request data) and PUT (take action or actuate the device) commands. [8]
- **Metadata:** Has the knowledge of the devices and sensors and how to communicate with them (used by services like Data and Command). Also uses "redis" (section 2.2.5) to store its knowledge about the devices and values. [8]

2.2.2 Supporting Services Layer

These services perform tasks such as logging, scheduling or sending notifications/alarms. The edge analysis also takes place here. These services can be considered optional and are not required. [9]

- **Scheduler:** Provides a "clock" that fires operations at specified times (interval). For example to clean up old sensed events that have been successfully exported out of EdgeX. [9]
- **Notifications / Alarm:** Send out an alert or notification (to another system or to a person). For example, when sensor data is detected outside of certain parameters. [9]
- **Rules engine / Kuiper:** Implementation of *data analytics* with if-then conditional actuation at the edge based on collected sensor data. *Kuiper* allows fast data processing on the edge and write rules in *SQL*. The rules engine is based on three components: Source, SQL and Sink. [9] More detailed information about the Kuiper rules engine can be found in the documentation [10] or under section 3.5.

2.2.3 Application Services Layer

This layer and its service is used to filter or transform collected sensor data and then send it to a recipient of your choice, e.g. cloud providers like Amazon IoT Hub, Google IoT Core, Azure IoT Hub. Data can be prepared (transform, filter, ...) and groomed (format, compress, encrypt, ...) before being sent to an endpoint. [11]

2.2.4 Device Services Layer

This layer interacts with the devices and sensors. Various protocols, such as *REST*, *MQTT* or even *SNMP*, are available for the south-side connection. The services not only collect data from the devices, but also get status updates, transform the data before sending them to the next layer or discover devices. After installation and running of EdgeX the REST and virtual services are starting by default. Other protocols must be added additionally. Protocols like MQTT or SNMP are already implemented and just need to be "re-enabled". More information about the available protocols and which ones are under development can be found in the EdgeX-Wiki ([12]). [13]

2.2.5 Additional useful services

- **Redis:** Is an open source in-memory data strucutre store, which is used as a database, cache and message broker. EdgeX uses Redis (Geneva release) as the default database for sensor data as well as for metadata about the devices or sensors that are connected. [14]

- **Consul:** Is an open source project used as registry service in EdgeX. It provides native features for service registration, service discovery, and health checking. Therefore it also provides a web user interface dashboard, which enables to see what services are running or if there are any errors. All other services are expected to register with Consul when they start. [15, 16]

3 Implementation

3.1 Use case and system architecture

Since there are multiple variations of IoT, the possible use cases are manifold. For instance an IIoT solution could be developed. For IIoT extensive monitoring of sensor values from industrial machines is often required. In a simple threshold applications machines could be shut down, if certain temperature values become critical for the system. The requirements for this use case would be to collect temperature values from multiple devices and act on a threshold. Storing of the collected data for future data analysis could also be of interest. Therefore a cloud service is most likely to be used, since resources are limited on the edge. Generally speaking such a system is required to get data from multiple nodes, reacts on a threshold and have a connection to a cloud service. Such a system could also be the base of a AI recognition system, that collects data on the edge and performs the classification in the cloud. For security purposes often only the IoT gateway gets connected with the internet, while the sensor nodes are connected via a local network. Therefore the gateway should be located close to the edge devices, possibly be one by its own. For that reason, this installation guide addresses the deployment of EdgeX on an edge device. It also demonstrates the often needed infrastructure of multiple edge devices sending data to the gateway which are the forwarded to a cloud service.
As physical nodes multiple Raspberry pi 3+ were used, since they were available. Any other edge device could also be used as long as it supports one of the communication protocols that EdgeX works with. Initially the Raspberry pi 3+ was considered as gateway as well. EdgeX Foundry states the following minimal requirements: [17]

- Minimum 1GB Memory
- 64bit CPU
- Minimum 3GB of storage (at least 32GB recommended)
- Operating systems:
  - Windows 7-10

- Ubuntu Desktop (14-20)
- Ubuntu Server (14-20)
- Ubuntu Core (16-18)
- Mac OS X 10

The Raspberry pi 3+ just fulfills the minimum requirement of memory. In trials multiple services of EdgeX started only partially or not at all. Therefore the Raspberry pi 4 with 4 GB memory was chosen for the EdgeX implementation.
For then cloud service Amazon Web Services (AWS) was picked due to its good documentation and free trials.
The final system architecture is visualized in figure 4

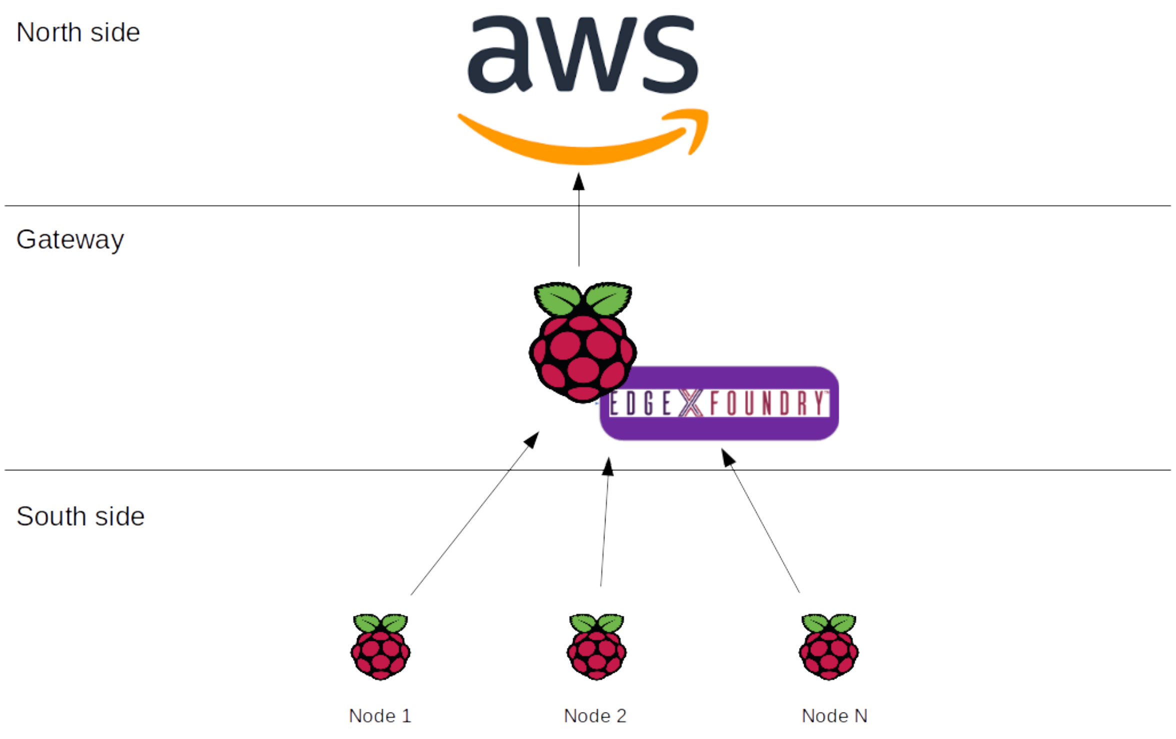


Figure 4: Deployed system architecture

3.2 Implementation on a Virtual Machine

EdgeX can run on any platform supporting Docker and docker-compose. See section 3.1 for the official platform requirements. For this implementation example Ubuntu 20.04 was used.

Note: To assign the virtual machine an IP address in the local network, the VM network should be bridged. So it is possible to communicate directly with the VM. In VirtualBox this can be set under the VM settings → *"Network"* → *"Attached to:"* → *"Bridged Adapter"*. Otherwise ports must be forwarded. [4]

3.2.1 Installation of Docker and docker-compose

To install Docker and Docker Compose the following steps need to be performed: [4]

1. Get the newest system updates
   - `sudo apt update`
   - `sudo apt upgrade`
2. Install Docker-CE [4]
   - `sudo apt install apt-transport-https ca-certificates curl software-properties-common`
   - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
3. Install docker-compose
   - `sudo apt install docker-compose`

3.2.2 Installation of EdgeX

The EdgeX microservices are controlled by a docker-compose file (YAML format). Ports and dependencies can be edited in this file. Also microservices can be activated and deactivated here. To start the EdgeX microservices the "docker-compose" command has to be executed in the same folder as the file. [4]

1. Create a directory for the docker-compose file [optional]
   - `mkdir edgex`

- `cd edgex`

2. Download the docker-compose.yml file (last checked on 06.12.2020)

   - `wget https://raw.githubusercontent.com/edgexfoundry/developer-scripts/master/releases/geneva/compose-files/docker-compose-geneva-redis-no-secty.yml -O docker-compose.yml` [4]

   Older releases can be found in the official Github repository [18].

3. Pull the newest docker container and list them

   - `sudo docker-compose pull`
   - `sudo docker image ls`

3.2.3 Running EdgeX

The following commands can be used to start EdgeX, as well as check that all services are running.

1. Start EdgeX with docker-compose
   docker-compose commands must be executed in the same folder as the docker-compose.yml file.

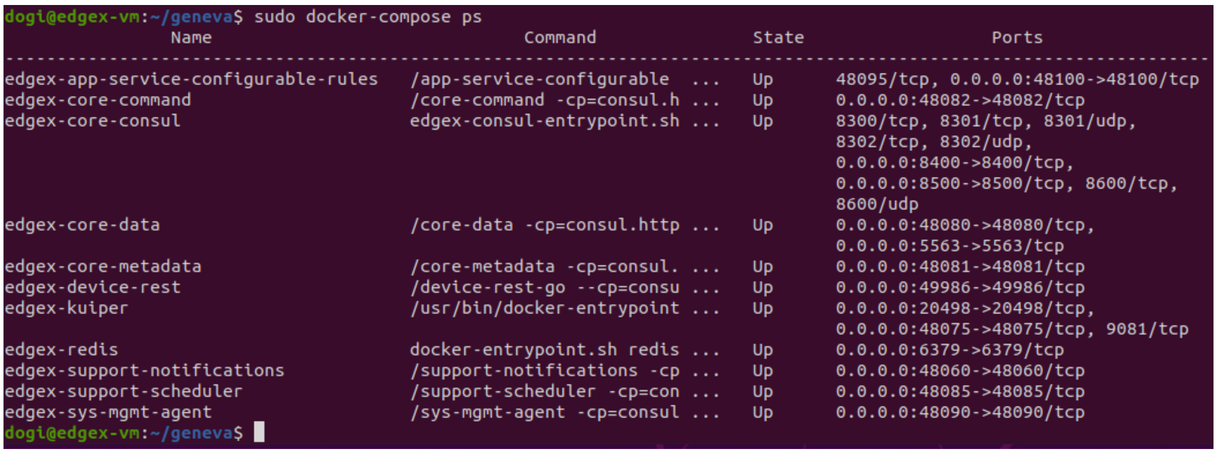
   - `sudo docker-compose up -d`

   "-d" lets EdgeX run in the background and thus prevents the console output of the services.

2. View running mincroservices

   - `sudo docker-compose ps`

   With the 'docker-compose ps' command the used ports of each microservice are listed. The ports are defined in the docker-compose.yml. Alternatively, the URL 'http://localhost:8500/ui/dc1/services' can be called in the browser (replace localhost with EdgeX gateway ip address for external use). This is available when the service 'consul' has been started. All running services and possible errors can be viewed there. [4]



```
dogi@edgex-vm:~/geneva$ sudo docker-compose ps
                Name                                 Command               State                  Ports
-------------------------------------------------------------------------------------------------------------------------
edgex-app-service-configurable-rules   /app-service-configurable  ...   Up      48095/tcp, 0.0.0.0:48100->48100/tcp
edgex-core-command                     /core-command -cp=consul.h ...   Up      0.0.0.0:48082->48082/tcp
edgex-core-consul                      edgex-consul-entrypoint.sh ...   Up      8300/tcp, 8301/tcp, 8301/udp,
                                                                                8302/tcp, 8302/udp,
                                                                                0.0.0.0:8400->8400/tcp,
                                                                                0.0.0.0:8500->8500/tcp, 8600/tcp,
                                                                                8600/udp
edgex-core-data                        /core-data -cp=consul.http ...   Up      0.0.0.0:48080->48080/tcp,
                                                                                0.0.0.0:5563->5563/tcp
edgex-core-metadata                    /core-metadata -cp=consul. ...   Up      0.0.0.0:48081->48081/tcp
edgex-device-rest                      /device-rest-go --cp=consu ...   Up      0.0.0.0:49986->49986/tcp
edgex-kuiper                           /usr/bin/docker-entrypoint ...   Up      0.0.0.0:20498->20498/tcp,
                                                                                0.0.0.0:48075->48075/tcp, 9081/tcp
edgex-redis                            docker-entrypoint.sh redis ...   Up      0.0.0.0:6379->6379/tcp
edgex-support-notifications            /support-notifications -cp ...   Up      0.0.0.0:48060->48060/tcp
edgex-support-scheduler                /support-scheduler -cp=con ...   Up      0.0.0.0:48085->48085/tcp
edgex-sys-mgmt-agent                   /sys-mgmt-agent -cp=consul ...   Up      0.0.0.0:48090->48090/tcp
dogi@edgex-vm:~/geneva$
```

Figure 5: View of all running microservices in the console

Important: To be able to communicate with EdgeX from another device, the IP addresses in the docker-compose.yml file must be changed from "127.0.0.1" to "0.0.0.0" (to allow all IP addresses) or a specific ip address that should have access, otherwise it is only possible to communicate with EdgeX locally, on the VM itself.

To stop EdgeX, the following commands has to be executed in the same folder as the docker-compose.yml file: [4]

1. Stop the services:
   - `sudo docker-compose stop`
2. Stop and remove the services:
   - `sudo docker-compose down`
3. To stop services, which were removed from the docker-compose.yml file:
   - `sudo docker-compose down --remove-orphans`

3.3 Implementation on Raspberry Pi

The following shows the implementation on the Raspberry Pi. First the node is set up on which the data is generated. To generate a real use case, system data are gathered and send to the EdgeX device. Then the implementation of the EdgeX environment on a Raspberry is described. For the use case that two Raspberry devices are to be used to transfer the data, one will act as Node and one as EdgeX device.

Note: The script to read out the data is attached in appendix A.1. It can be used for multiple nodes, but small changes are necessary. The script can be used on any Raspberry Pi with a Linux operating system installed. The EdgeX environment is only executable on a Raspberry Pi 4 as mentioned in section 3.1.

3.3.1 Installing Raspberry Pi Operating System(OS)

The following section describes the installation of the Raspberry Pi operating system. If a basic understanding with the procedure of a Raspberry Pi is available, this chapter can be skipped. It can be continued with the chapter Implementaion of Raspberry-Node.

Note: Installing EdgeX on a Raspberry Pi is similar to installing it on a virtual machine. However, EdgeX requires a 64 bit operating system. It must be downloaded from the following URL:
`https://downloads.raspberrypi.org/raspios_arm64/images/raspios_arm64-2020-05-28/2020-05-27-raspios-buster-arm64.zip`

The 64 bit operating system is only executable on the Raspberry Pi 4 and the Raspberry Pi 3+. If an older Raspberry Pi is used to send and generate data, the 32 bit operating system must be installed. You can choose between an operating system with and without a graphical user interface. In general, an operating system without graphical user interface is sufficient for the implementation of the existing script.

The operating system with user interface can be downloaded from the following URL:
`https://downloads.raspberrypi.org/raspios_armhf/images/raspios_armhf-2020-12-04/2020-12-02-raspios-buster-armhf.zip`
The operating system without user interface can be downloaded from the following URL:
`https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2020-12-04/2020-12-02-raspios-buster-armhf-lite.zip`

The following steps are identical for both operating systems. Only points 8 and 9 are omitted for the operating system without user interface.

1. Download of Raspberry Pi image.
2. Flash Operating System on Raspberry Pi card with balenaEtcher or Win32DiskImager. Win32DiskImager can be downloaded from following URL:
   `https://www.chip.de/downloads/Win32-Disk-Imager_46121030.html`
3. Open flashed card and insert into the boot partition a textfile named:
   - ssh.txt

   this file must be empty.
4. Plug the card into your Raspberry Pi.
5. Now the Raspberry must be connected to the power and the internet. Then Putty must be downloaded on a computer in the network. With this you can access the Raspberry Pi via SSH on port 22 without mouse and keyboard. Putty can be downloaded from following URL:
   `https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`
6. In Putty the IP address of the Raspberry Pi must be entered. This can be found via the graphical user interface of the router.
7. After the connection via Putty is established, you can log in with Username and Password.

   Login informations:
   - Username: pi
   - Password: raspberry

   Now the system updates can be installed first.
   - `sudo apt update`
   - `sudo apt upgrade`
8. Now the graphical user interface can be activated via the Software Configuration Tool. The following command takes you to the mentioned menu:

- `sudo raspi-config`

9. Under the point 5 Interfacing Options the graphical remote access to the Raspberry Pi can be enabled in the menu under point P3 VNC. After that, access via VNC viewer is possible. This allows unrestricted access to the Raspberry via graphical user interface. VNC Viewer is available for Windows, macOS and Linux. It can be downloaded from the following URl:
   `https://www.realvnc.com/de/connect/download/viewer/linux/`

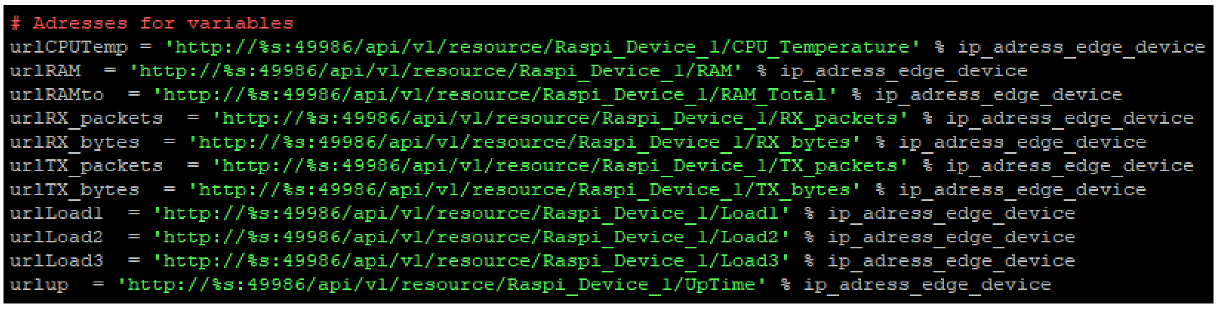
3.3.2 Implementation of Raspberry-Node

Numerous protocols are available for communication. Two well-known ones are REST and Bluetotth Low Energy (BLE). However, BLE is still in the development phase. For this reason, REST communication was chosen.

1. Get the newest system updates
   - `sudo apt update`
   - `sudo apt upgrade`
2. Choose a location for the script. Write your own script to send the data (Or random data) or use the predefined script in appendix A.1. Open a text file with any editor. In the following as an example the editor Nano.
   - `sudo nano Rpi_send.py`

   Then copy the predefined script into the textfile.
3. Now you need to change the ip adress in line 16. Write into this variable the ip address of the EdgeX device.
   - `ip_adress_edge_device = ”*.*.*.*”`
4. If multiple nodes are to be used, the program code in the additional devices must be adapted. The following graphic shows the section that must be changed. The section starts at line 72 and ends at line 82.

   In the addresses of the additional nodes their specific names must be entered. The following nomenclature is recommended:
   - `Raspi_Device_1` → `Raspi_Device_2` → `Raspi_Device_3` → `Etc....`

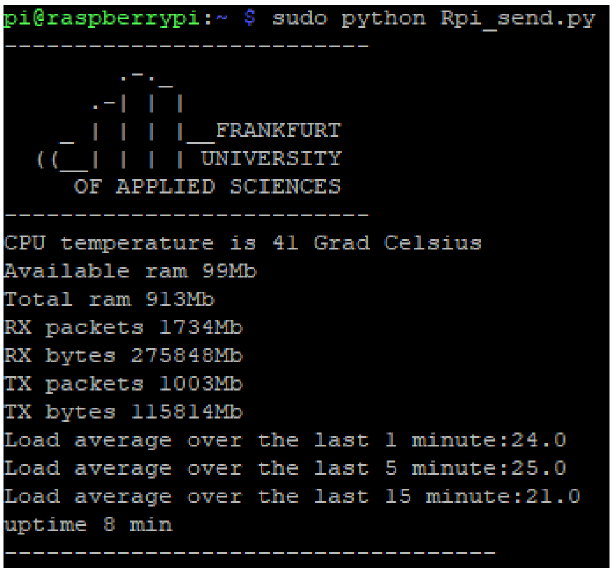


```
# Adresses for variables
urlCPUTemp = 'http://%s:49986/api/v1/resource/Raspi_Device_1/CPU_Temperature' % ip_adress_edge_device
urlRAM  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RAM' % ip_adress_edge_device
urlRAMto  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RAM_Total' % ip_adress_edge_device
urlRX_packets  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RX_packets' % ip_adress_edge_device
urlRX_bytes  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RX_bytes' % ip_adress_edge_device
urlTX_packets  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/TX_packets' % ip_adress_edge_device
urlTX_bytes  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/TX_bytes' % ip_adress_edge_device
urlLoad1  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load1' % ip_adress_edge_device
urlLoad2  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load2' % ip_adress_edge_device
urlLoad3  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load3' % ip_adress_edge_device
urlup  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/UpTime' % ip_adress_edge_device
```

Figure 6: Addresses of the variables

This must also be changed in the EdgeX device. But more about that later in chapter 3.4.2.

5. After the script has been modified it can be exited and started with the following command. The script is only executable after the EdgeX Gateway has been started.
   - `sudo python Rpi_send.py`
6. After the script is started, the following values are output to the console. The



```
pi@raspberrypi:~ $ sudo python Rpi_send.py
--------------------------
        .-._
      .-| | |
    _ | | | |__FRANKFURT
  ((__| | | | UNIVERSITY
     OF APPLIED SCIENCES
--------------------------
CPU temperature is 41 Grad Celsius
Available ram 99Mb
Total ram 913Mb
RX packets 1734Mb
RX bytes 275848Mb
TX packets 1003Mb
TX bytes 115814Mb
Load average over the last 1 minute:24.0
Load average over the last 5 minute:25.0
Load average over the last 15 minute:21.0
uptime 8 min
-----------------------------------
```

Figure 7: The sent data of the Raspberry Node

continuous loop causes the system data to be sent to the EdgeX device again every 5 seconds.

The CPU temperature is sent in degrees Celsius. The total RAM of the Raspberry Pi and its available RAM are transmitted. In addition, the previous Internet traffic via the LAN interface.

Note: If the Raspberry Pi is connected via *wifi*, the program code must be modified accordingly.

The load of the Raspberry Pi is formed by means of an average value. The load is determined over the last 1, 5 and 15 minutes. The load average is given as a percentage. The uptime shows the operating time in minutes.

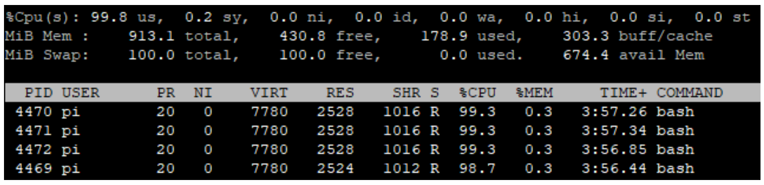
7. To change the system values of the Raspberry Pi, a continuous loop can be executed in the background which increases the system load and the processor temperature. To execute it, the following command must be entered into the console.

   - `for i in 1 2 3 4; do while : ; do : ; done & done`

8. The following command can be used to monitor the system load.

   - `top`

   The following graphic appears.



```
%Cpu(s): 99.8 us,  0.2 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    913.1 total,    430.8 free,    178.9 used,    303.3 buff/cache
MiB Swap:    100.0 total,    100.0 free,      0.0 used.    674.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4470 pi        20   0    7780   2528   1016 R  99.3   0.3   3:57.26 bash
 4471 pi        20   0    7780   2528   1016 R  99.3   0.3   3:57.34 bash
 4472 pi        20   0    7780   2528   1016 R  99.3   0.3   3:56.85 bash
 4469 pi        20   0    7780   2524   1012 R  98.7   0.3   3:56.44 bash
```

Figure 8: Utilization of the Raspberry

   To terminate the processes they must be terminated by the following command. After the kill command the respective process must be named. As an example, here is the command to kill the first process.

   - `sudo kill 4470`

3.3.3 Implementation of Raspberry-EdgeX gateway

This section describes the installation of Docker. This is similar to the installation of Docker on the virtual machine. [19]

Note: The Raspberry Pi 4 has a quad-core Cortex-A72 processor from ARM with 64 bit and 1.5 GHz. The installed operating system is based on Debian. This has to be taken care of during the installation.

1. You have to uninstall the possibly old versions. [19]

    - `sudo apt-get remove docker docker-engine docker.io containerd runc`

    Note: For installing on Raspberry Pi using the repository is not yet supported. We need to use convenience scripts.

2. Install Docker using the convenience script: [19]

    - `curl -fsSL https://get.docker.com -o get-docker.sh`

    - `sudo sh get-docker.sh`

    After Docker has been successfully installed, the function can be tested with a hello world image. If everything works, Docker should show a welcome message. With the following command the check can take place: [19]

    - `sudo docker run hello-world`

Installing Docker-Compose

After installing Docker it is necessary to install Docker Compose. Because this is done on the Raspberry Pi in a different way than to the virtual machine.

1. Installing the required components to make Docker Compose runnable. Execute these three commands one after the other. [20]

    - `sudo apt-get install -y libffi-dev libssl-dev`
    - `sudo apt-get install -y python3 python3-pip`
    - `sudo apt-get remove python-configparser`

2. Docker Compose can then be installed. [20]

    - `sudo pip3 -v install docker-compose`

3. Now the installation of Docker and Docker Compose is complete. Skip to 3.2.2 Installation of EdgeX . From this point on, the procedure is again the same as for the virtual machine. The procedure described there can be used with the Raspberry Pi.

    Note: Before data can be received, the variables must be created. This is done in the next chapter 3.4 Creating of devices and variables.

3.4 Creating of devices and variables

To be able to send data to the created EdgeX gateway, a digital representation of devices and variables must first be created. Devices are described with a device profile, which can then be used to create device instances. A real device then sends data to its digital counterpart, using the ID or name of the digital device. The following descriptions and commands were applied on a Raspberry Pi 4.

3.4.1 Creating the Device Profile

To use devices with EdgeX, a profile must be created. This must be in the same directory as everything else before. Again, an empty document must be created to create the profile. For this purpose there is again a predefined script in the appendix.

1. If the correct directory is not specified, you must navigate to it.
   - `cd edgex`
2. Create an empty file with the name Raspi_Device_Profile.yaml.
   - `sudo nano Raspi_Device_Profile.yaml`
3. Copy the content of the attached script in the appendix A.3 into the file.
4. Save and close the file.

3.4.2 Creating the variables

When all services are started and running, abstract descriptions of the variables, which are sent in the data from the devices, can be created. This can only be done after the services have been started.

Bash file
Attached in appendix A.2 is a bash file. The Bash file automatically creates the variables implemented in this report. It also starts Docker and creates the devices. Multiple devices can be created by modifying the file. The modification is done in analogy to the modification in chapter 3.3.2 Implementation of Raspberry-Node. The content of the bash file can be inserted into a text file of the Raspberry. The bash file must be in the same directory as the docker-compose file. The procedure is as follows:

1. If the correct directory is not specified, you must navigate to it.
   - `cd edgex`
2. Create an empty file with the name start.sh.
   - `sudo nano start.sh`
3. Copy the content of the attached script in appendix A.2 into the file.
4. Save and close the file.
5. To execute the file, the bash file must now be called.
   - `bash start.sh`
6. After the bash file has been successfully executed, you should see a prompt to start the work.

Enter manually
If the individual variables are to be entered manually, the following procedure should be followed:

1. Each information that is transferred from the Raspberry Node must be defined at the EdgeX gateway. As an example the declaration of the CPU temperature is shown here. The variable must be created according to the following scheme. [4]

   ```
   curl --location
   --request POST 'http://localhost:48080/api/v1/valuedescriptor'
   --header 'Content-Type: application/json'
   --data-raw'
   "name": "CPU_Temperature",
   "description": "CPU Temperature in Celsius",
   "min": "", "max": "",
   "type": "Int64",
   "uomLabel": "CPU_Temperature",
   "defaultValue": "0",
   "formatting": "%s",
   "labels": [ "environment", "CPU_Temperature"]'
   ```
2. For each information the variable must be adapted accordingly. Therefore also the file of the node and the file Raspberry_Device_Profile must be adapted.

3. The following command loads the device profile for the Raspberry: [4]

   ```
   curl --location --request POST 'http://localhost:48081/api/v1/deviceprofile/ uploadfile'
   --form 'file=@"./Raspi_Device_Profile.yaml"'
   ```

4. The last step missing is the creation of the devices. With this command the Raspberries are configured. For each node that is used, a configuration must be made. With the following command the devices can be created. For the use of multiple nodes, the devices must be adapted accordingly. As described in the previous chapter 3.3.2 Implementation of Raspberry-Node. [4]

   ```
   curl --location
   --request POST 'http://localhost:48081/api/v1/device'
   --header 'Content-Type: application/json'
   --data-raw '"name": "Raspi_Device_1",
   "description": "Raspberry Pi 1",
   "adminState": "unlocked",
   "operatingState": "enabled",
   "protocols": "example": "host": "dummy", "port": "1234", "unitID": "1",
   "labels": ["UpTime", "Load1", "Load2", "Load3", "RX_packets", "RX_bytes",
   "TX_packets", "TX_bytes", "RAM_Total", "RAM", "CPU_Temperature sensor",
   "DHT11"],
   "location": "Berlin",
   "service": "name": "edgex-device-rest",
   "profile": "name": "Raspi_Device"'
   ```

Now the Raspberry Pi is configured and can be used as a gateway.

3.4.3 Look up stored devices and other data

All created devices, variables, as well as received data can be viewed. For this purpose, the console or alternatively the program "Postman" can be used. However, Postman is not available for Raspberry devices. But the commands can also be sent from other devices, if the ip addresses in the docker-compose.yml are <u>not</u> set to local (127.0.0.1). The following commands are only a few available commands. The following commands can be executed locally. Alternatively, the URLs can be entered into the browser. For an external call, the "localhost" must be replaced with the IP address of the EdgeX gateway. Helpful commands are:

- **List of created devices:** [4]
  `curl -X GET 'http://localhost:48082/api/v1/device' | jq`
- **List of created variables:** [4]
  `curl -X GET 'http://localhost:48080/api/v1/valuedescriptor' | jq`
- **List of all received data:** [4]
  `curl -X GET 'http://localhost:48080/api/v1/reading' | jq`
- **List of the last 10 received data of a specific device:** [21]
  `curl -X GET 'http://localhost:48080/api/v1/event/device/{DEVICE_NAME}/10' | jq`
- **Display number of received data:** [4]
  `curl -X GET 'http://http://localhost:48080/api/v1/event/count'`
- **Remove device with ID or name**
  `curl -X DELETE 'http://localhost:48081/api/v1/device/id/{(device_id_here)}'` or
  `curl -X DELETE 'http://localhost:48081/api/v1/device/name/{(device_name_here)}'`

3.5 Export EdgeX Foundry data

For exporting data to the north side, one can choose between using the Application service and the rules engine. By using the latter, data can be filtered before they are exported. Since the Geneva release, EdgeX Foundry uses EMQ's Kuiper as rules engine. The service consists of the three components: Source, SQL and Sink.[22]
For debugging purposes it might become necessary to alter the default settings of EdgeX services. Their configuration files are located within their corresponding docker containers. For navigating through the directory within a container, a shell can be opened in the running container. This can be achieved by using this command:

```
sudo docker exec -it < containername > /bin/sh
```

Kuipers configuration for instance can be changed in the file */etc/kuiper.yaml*. This way the logging level of EdgeX services can be changed as well. The log files can be shown by the command: [4]

```
sudo docker logs < servicename >
```

Rules, streams and also devices can be managed in a visual way as well. For that purpose the Golang UI interface needs to be included in the docker-compose file by adding these lines: [4]

```
ui:
```

```
container_name: edgex-ui-go
hostname: edgex-ui-go
image: nexus3.edgexfoundry.org:10004/docker-edgex-ui-go:master
networks:
edgex-network: null
ports:
- "0.0.0.0:4000:4000/tcp"
read_only: true
```

It is vital to keep the indentation schema, when making these entries. Best practice is to compare the entries with the once below and above.

3.5.1 Creation of a Kuiper data source

The Source is the data stream, from where the data are fed into the rules engine. In the EdgeX implementation, Kuiper listens per default to the message bus at port 5566, which is used by the Application service for publishing its messages. This default setting can be altered in the configuration file located at *etc/sources/edgex.yaml*. [23]
A stream can be created by sending the following POST message body to Kuiper's REST-API at $< EdgeX\_IP >$: 48075/*streams*: [23]

```
{
  "sql": "create stream raspberries() WITH (FORMAT="JSON",
  TYPE="edgex")"
}
```

The id of this stream is "raspberries" and it contains data in json format with types that are specified by EdgeX value descriptors, which is specified by the FORMAT and TYPE parameter. Custom value types are translated in types that can be interpreted by Kuiper either during runtime or compilation. All created streams can be listed by sending a GET request to the same resource.[24][23]

3.5.2 Creation of Kuiper rules

The two remaining components of Kuiper (SQL and Sink) are defined in the rules themselves. Same as for the stream, Kuiper's REST-API can be used for handling the rules. In case of rules, the target resource for the POST request is $< EdgeXIP >$: 48075/*rules*.

The following rule has the ID "criticalTemperature": [25]

```
{
"id": "criticalTemperature",
"sql": "SELECT temperature, meta(Device) AS device FROM
raspberries WHERE (meta(device)=\"RaspiDevice1\"
OR meta(device)=\"RaspiDevice2\") AND temperature > 80",
"actions": [
    {
        "rest": {
            "dataTemplate": "{\"content\":\"json .\",
                            \"Temperature\":{{.temperature}},
                            \"Device\":\"{{.device}}\"}",
            "method": "post",
            "sendSingle": true,
            "url": "http://<IP>:8000",
            "retryInterval": 0
        }
    },
    {
        "log": {}
    }
],
"options": null
}
```

Apart from the data that are transported through the message bus, additional meta data can be extracted [26]. The presented rule will select an entry of the type "temperature" from the stream along with its meta data "device", which is the device id. The rule filters for the device ids "RaspiDevice1" and "RaspiDevice2" when either of them has been receiving a temperature value above 80 °C. In either case the rule executes what is listed in the "actions" section. So far it is not possible to filter for the data type of the device. This field is not included in the meta data.[25] [26]
The keyword "rest" determines that Kuiper shall use the REST-Sink. The action creates a POST request that contains the data defined in "dataTemplate". The option "sendSingle" specifies that all data are send in a single message. The "retryInterval" is set to 0 and therefore this POST request will only be executed once, even if it fails. For debugging

purposes it proved useful to use an http server as target of the rules. The presented rule targets an http server that listens to port 8000. The source code for the used server is attached in appendix A.9. The URL needs to be exchanged with the address of the AWS Gateway API for the final deployment.[25]
When created, the rule directly runs. To stop the rule, a POST request has to be send to $< EdgeXIP >$: $48075/rules/ < ruleID > /stop$ and for making it run again: $< EdgeXIP >$: $48075/rules/ < ruleID > /start$. To delete either a stream or a rule a DELETE request has to be send to the resource of the rule/stream. [27].

3.6 Amazon Web Services

For the north end implementation, AWS was chosen. This service from Amazon contains a collection of web services, each for a specific task. The four services that were used are called *Gateway API*, *Lambda*, *IAM* and *DynamoDB*. For this project they interact as visualized in figure 9. The *IAM* service is used to control access to service resources and was used for the implementation of the lambda rules. Therefore it is explained in subsection 3.6.2 and is not displayed in the overall service topology.

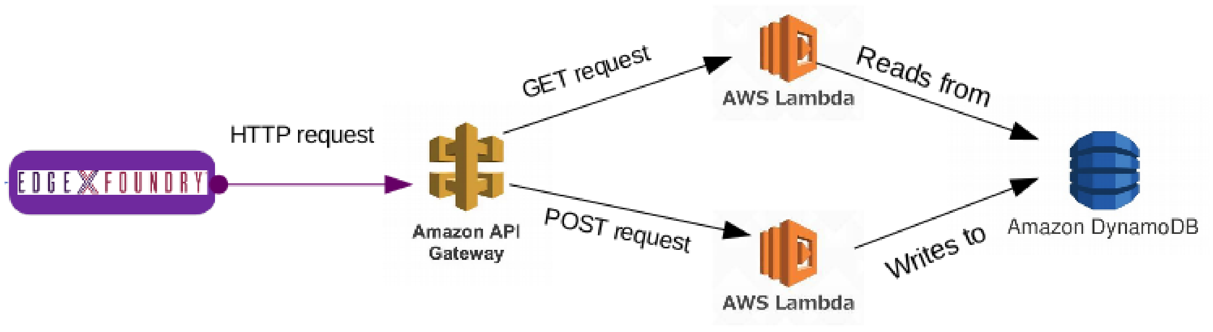


Figure 9: AWS service topology [28][29]

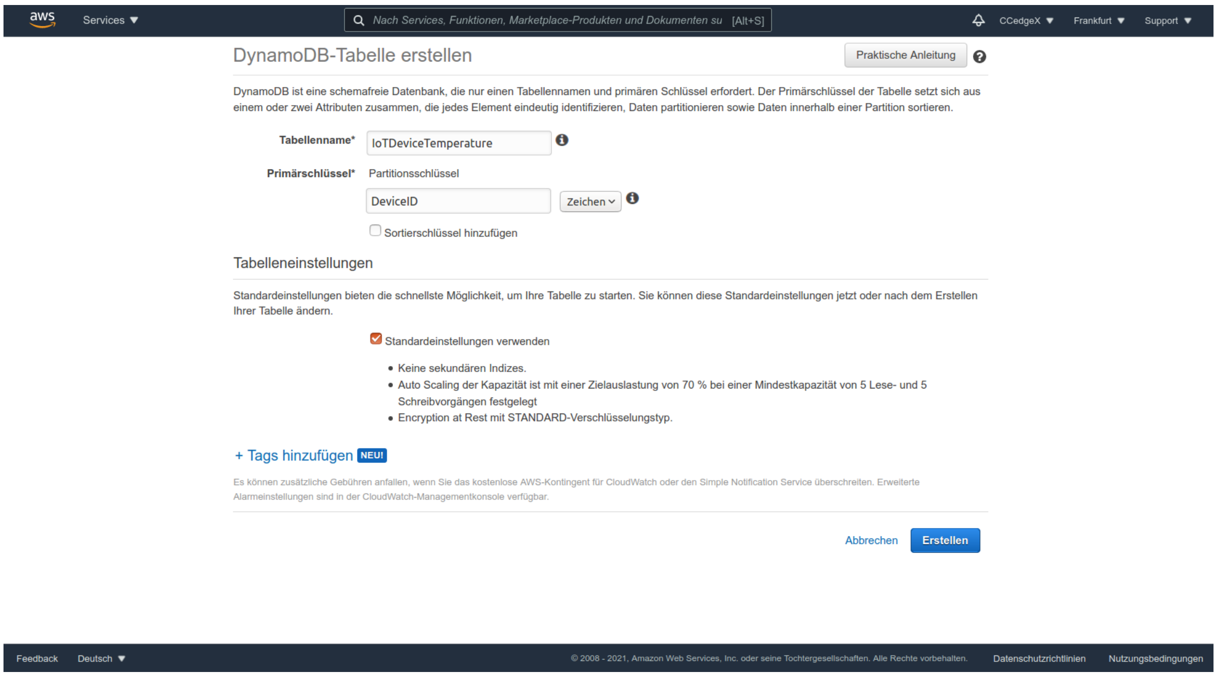
With the *Gateway API* service, one can create an Gateway, such that the implementation on AWS can be accessed by a program on another machine. Since EdgeX supports a REST-API for north side communication, a REST-API is created with *Gateway API*.[30]
The *Lambda* service provides an environment in which custom code can be executed. Its role in this architecture is to extract the entries of messages that came through the Gateway API and to store them in a NoSQL data bank which is implemented via the *DynamoDB* service.[31][32][33]
AWS has several physical server locations to choose from. In this project all services are

using the location *eu-central-1*, since this is the nearest. Regardless which location is chosen it is important to keep it consistent over all used services. Otherwise additional settings might be necessary.
When this document was created, all used services were covered by the free trier of amazon. The API Gateway service had 1 million API calls per month free for the first 12 month after the registration. The Lambda service had 1 million executions free per month and DynamoDB had 25 GB of free storage. The latter two had no time limitations. Generated costs and usage of the services were monitored through the cost explorer service [34]. [35]
This project focused on a proof of concept for the presented use case in section 3.1 and AWS was chosen exclusively due to its numerous documentations and tutorials. Therefore it cannot be said that AWS is a good choice in general for storing IoT data. It highly depends on the specific use case and its scope.

3.6.1 AWS DynamoDB

Firstly the database needs to be created. Type "DynamoDB" In the *AWS Managment console* and select the very first option of the service: "Create table". Choose "IoTDeviceTemperatures" as name of the table and "DeviceID" as primary key as shown in figure 10. Next activate the option "Autoscaling" such that the read-write capacity is managed dynamically. Select the "Use default settings" option, so that DynamoDB creates a role for that purpose automatically. At the very bottom of the page the "create" button can be found.[31]

So far the created table has only the property "DeviceID". Further columns are created automatically when an entry is generated.



DynamoDB-Tabelle erstellen

DynamoDB ist eine schemafreie Datenbank, die nur einen Tabellennamen und primären Schlüssel erfordert. Der Primärschlüssel der Tabelle setzt sich aus einem oder zwei Attributen zusammen, die jedes Element eindeutig identifizieren, Daten partitionieren sowie Daten innerhalb einer Partition sortieren.

Tabellenname*: IoTDeviceTemperature

Primärschlüssel*: Partitionsschlüssel

DeviceID — Zeichen

☐ Sortierschlüssel hinzufügen

Tabelleneinstellungen

Standardeinstellungen bieten die schnellste Möglichkeit, um Ihre Tabelle zu starten. Sie können diese Standardeinstellungen jetzt oder nach dem Erstellen Ihrer Tabelle ändern.

☑ Standardeinstellungen verwenden

- Keine sekundären Indizes.
- Auto Scaling der Kapazität ist mit einer Zielauslastung von 70 % bei einer Mindestkapazität von 5 Lese- und 5 Schreibvorgängen festgelegt
- Encryption at Rest mit STANDARD-Verschlüsselungstyp.

+ Tags hinzufügen NEU!

Es können zusätzliche Gebühren anfallen, wenn Sie das kostenlose AWS-Kontingent für CloudWatch oder den Simple Notification Service überschreiten. Erweiterte Alarmeinstellungen sind in der CloudWatch-Managementkonsole verfügbar.

© 2008 - 2021, Amazon Web Services, Inc. oder seine Tochtergesellschaften. Alle Rechte vorbehalten.

Figure 10: Set values for new AWS DynamoDB table

3.6.2 AWS Lambda implementation

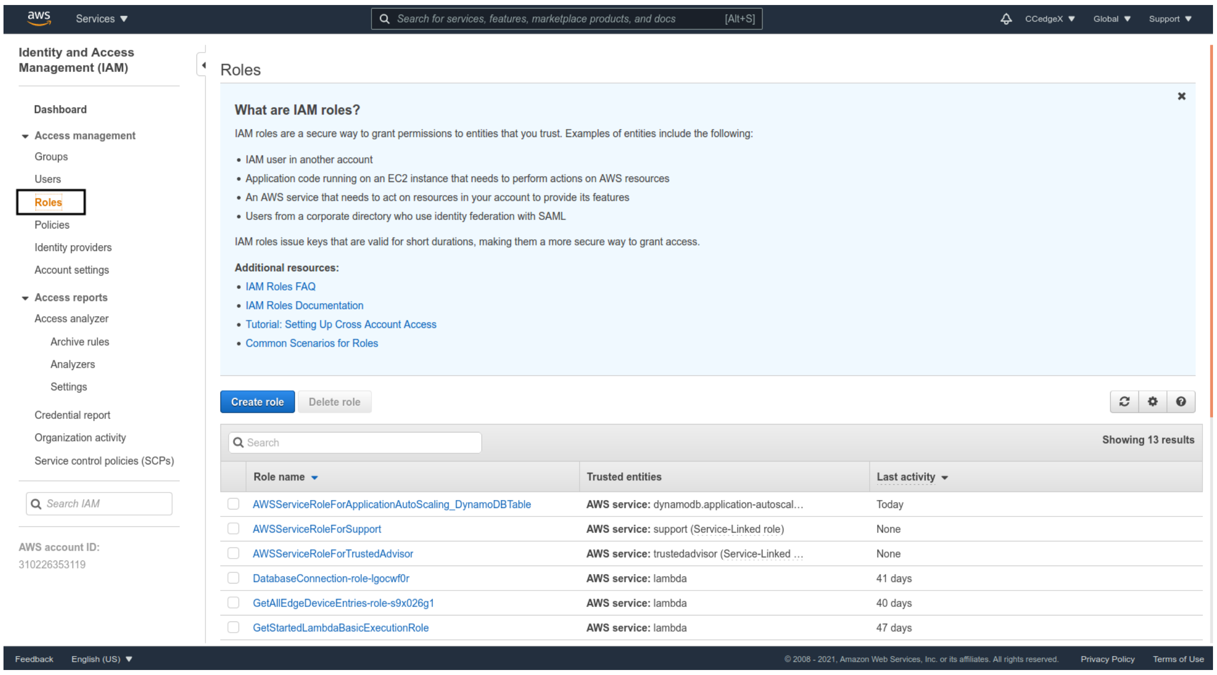
For accessing the created DynamoDB table, lambda functions are needed. However lambda functions cannot access the created table initially. For that purpose a corresponding lambda execution role has to be created. Access the *IAM* service over the management console, select "Roles" on the left hand menu (as marked in figure 11) and click the button "create role". [31]

Keep "AWS service" selected and choose Lambda (see figure 12) as the service this role is designated for. Then click on "Next: Permissions" and enter "AWSLambdaBasicExecutionRole" in the search bar as seen in figure 13. Select the listed role with that name and go on "Next". [31]

For this use case, Tags are not used. Therefore the next side does not need any entries. Now give the role the name "IoTDeviceTemperatureAccess" and click "Create role". For giving this role more specific rights, select the newly created role and click on "Add inline policy" as marked in figure 14.

In the upcoming view enter DynamoDB in the service search bar and add "GetItem" and "PutItem" in the Action search bar as seen in figure 15. [31]

Then go to the resource section and click on add ARN (Amazon Resource Name). This ARN is a unique identifier of the DynamoDB table we want to give access to. It can be



AWS account ID:
310226353119

Roles

What are IAM roles?

IAM roles are a secure way to grant permissions to entities that you trust. Examples of entities include the following:

- IAM user in another account
- Application code running on an EC2 instance that needs to perform actions on AWS resources
- An AWS service that needs to act on resources in your account to provide its features
- Users from a corporate directory who use identity federation with SAML

IAM roles issue keys that are valid for short durations, making them a more secure way to grant access.

Additional resources:

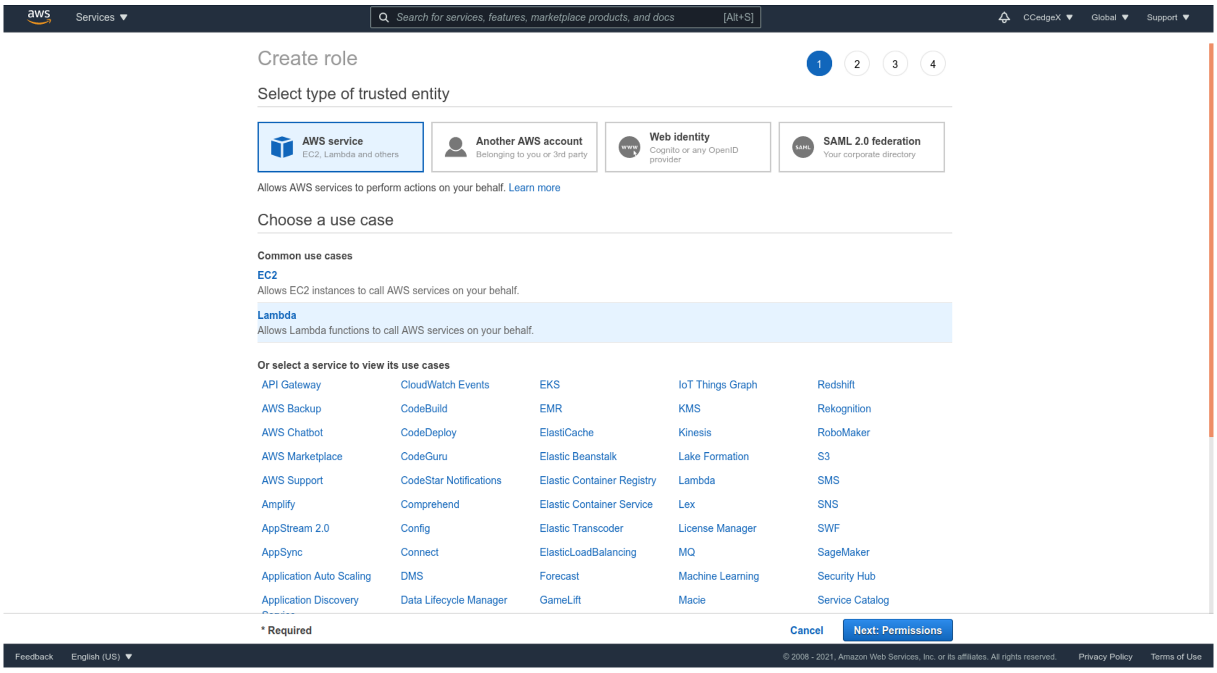
- IAM Roles FAQ
- IAM Roles Documentation
- Tutorial: Setting Up Cross Account Access
- Common Scenarios for Roles

Showing 13 results

Role name	Trusted entities	Last activity
AWSServiceRoleForApplicationAutoScaling_DynamoDBTable	**AWS service:** dynamodb.application-autoscal...	Today
AWSServiceRoleForSupport	**AWS service:** support (Service-Linked role)	None
AWSServiceRoleForTrustedAdvisor	**AWS service:** trustedadvisor (Service-Linked ...	None
DatabaseConnection-role-lgocwf0r	**AWS service:** lambda	41 days
GetAllEdgeDeviceEntries-role-s9x026g1	**AWS service:** lambda	40 days
GetStartedLambdaBasicExecutionRole	**AWS service:** lambda	47 days

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 11: Create a new IAM role



Create role

Select type of trusted entity

AWS service EC2, Lambda and others	**Another AWS account** Belonging to you or 3rd party	**Web identity** Cognito or any OpenID provider	**SAML 2.0 federation** Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

API Gateway	CloudWatch Events	EKS	IoT Things Graph	Redshift
AWS Backup	CodeBuild	EMR	KMS	Rekognition
AWS Chatbot	CodeDeploy	ElastiCache	Kinesis	RoboMaker
AWS Marketplace	CodeGuru	Elastic Beanstalk	Lake Formation	S3
AWS Support	CodeStar Notifications	Elastic Container Registry	Lambda	SMS
Amplify	Comprehend	Elastic Container Service	Lex	SNS
AppStream 2.0	Config	Elastic Transcoder	License Manager	SWF
AppSync	Connect	ElasticLoadBalancing	MQ	SageMaker
Application Auto Scaling	DMS	Forecast	Machine Learning	Security Hub
Application Discovery	Data Lifecycle Manager	GameLift	Macie	Service Catalog

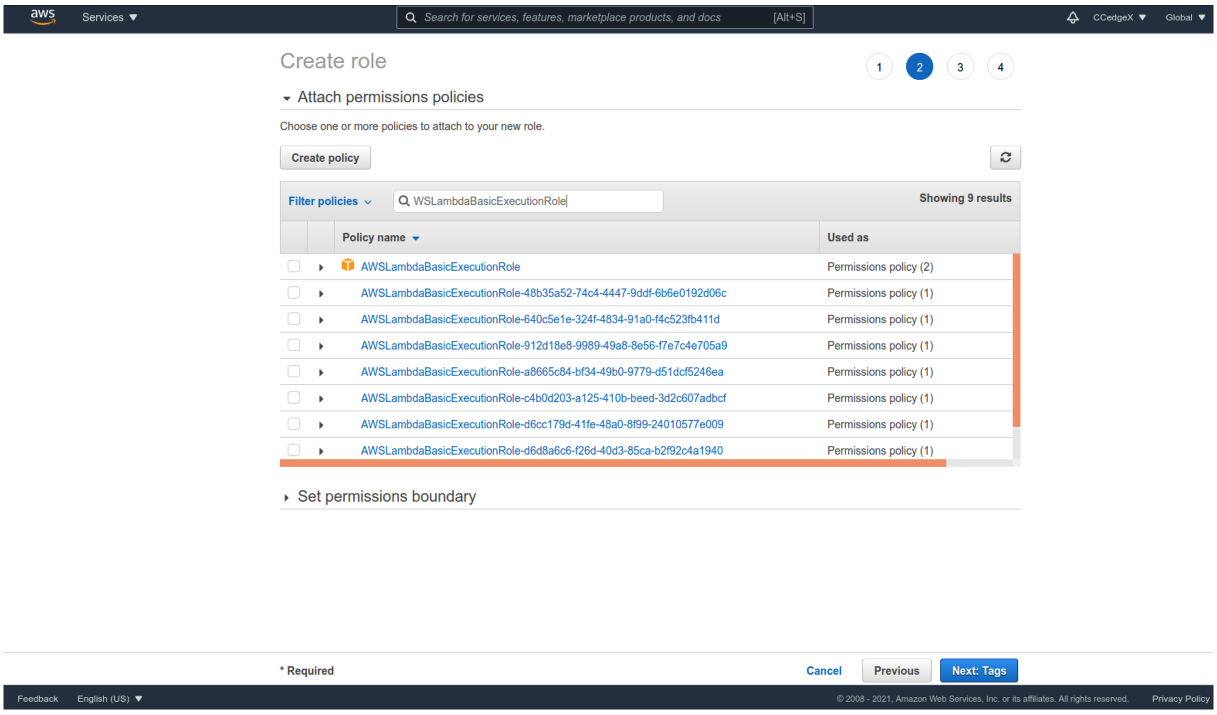
* Required

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 12: create new Lambda service role

found in the overview tab in the description of the IoTDeviceTemperatures table (see figure 16).[31]

Once added click on "Review policy", give it the name "DynamoReadWriteAccess" and click on "Create policy".[31]



Create role

Attach permissions policies

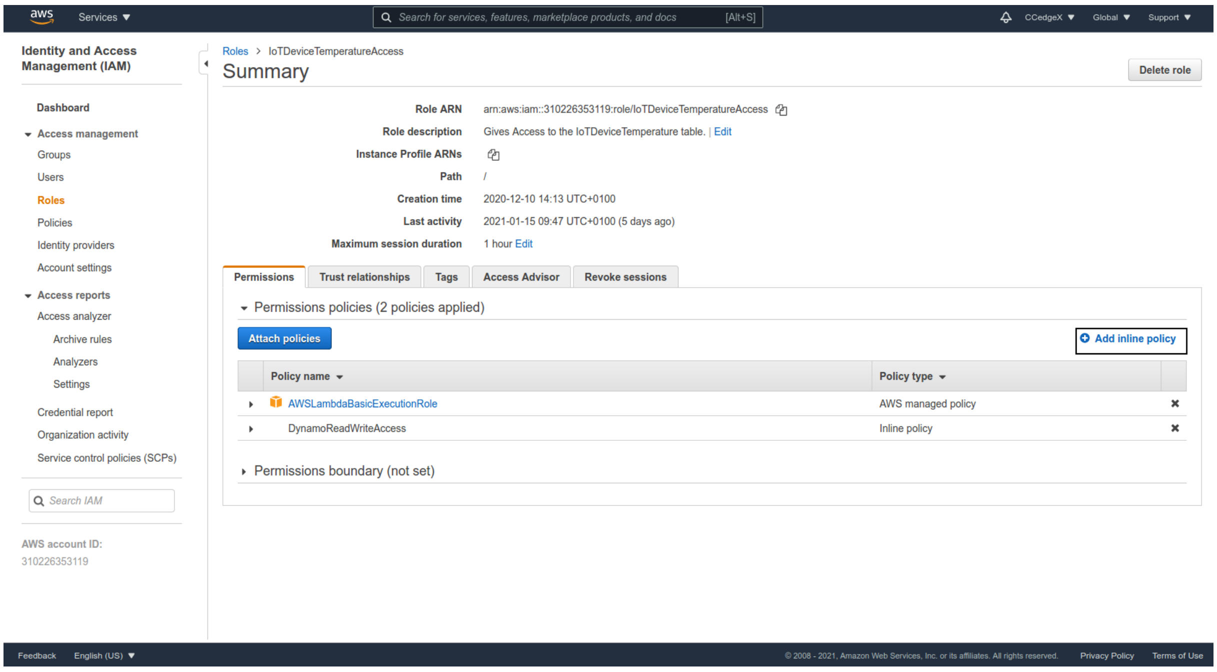
Choose one or more policies to attach to your new role.

		Policy name	Used as
		AWSLambdaBasicExecutionRole	Permissions policy (2)
		AWSLambdaBasicExecutionRole-48b35a52-74c4-4447-9ddf-6b6e0192d06c	Permissions policy (1)
		AWSLambdaBasicExecutionRole-640c5e1e-324f-4834-91a0-f4c523fb411d	Permissions policy (1)
		AWSLambdaBasicExecutionRole-912d18e8-9989-49a8-8e56-f7e7c4e705a9	Permissions policy (1)
		AWSLambdaBasicExecutionRole-a8665c84-bf34-49b0-9779-d51dcf5246ea	Permissions policy (1)
		AWSLambdaBasicExecutionRole-c4b0d203-a125-410b-beed-3d2c607adbcf	Permissions policy (1)
		AWSLambdaBasicExecutionRole-d6cc179d-41fe-48a0-8f99-24010577e009	Permissions policy (1)
		AWSLambdaBasicExecutionRole-d6d8a6c6-f26d-40d3-85ca-b2f92c4a1940	Permissions policy (1)

Set permissions boundary

aws
Services
Search for services, features, marketplace products, and docs [Alt+S]
CCedgeX
Global
1
2
3
4
Create policy
Filter policies
WSLambdaBasicExecutionRole
Showing 9 results
* Required
Cancel
Previous
Next: Tags
Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy

Figure 13: Create new Lambda basic execution role



Roles > IoTDeviceTemperatureAccess

Summary

Role ARN	arn:aws:iam::310226353119:role/IoTDeviceTemperatureAccess
Role description	Gives Access to the IoTDeviceTemperature table. | Edit
Instance Profile ARNs	
Path	/
Creation time	2020-12-10 14:13 UTC+0100
Last activity	2021-01-15 09:47 UTC+0100 (5 days ago)
Maximum session duration	1 hour Edit

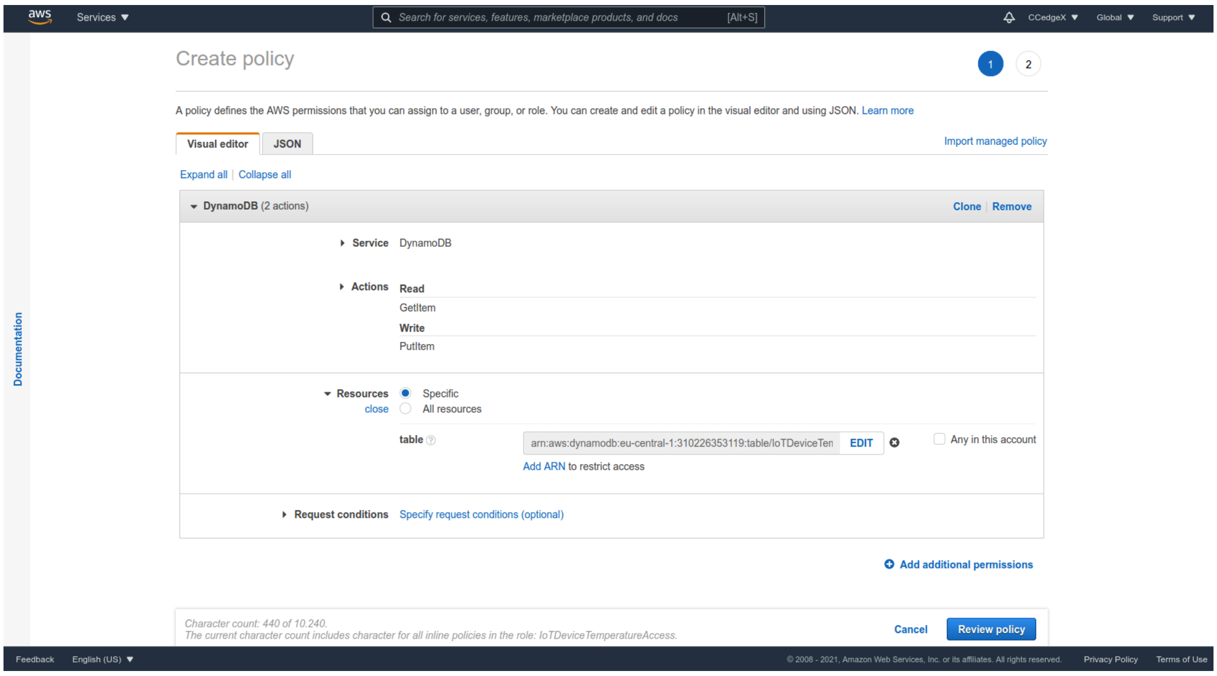
Permissions policies (2 policies applied)

	Policy name	Policy type	
	AWSLambdaBasicExecutionRole	AWS managed policy	✖
	DynamoReadWriteAccess	Inline policy	✖

Permissions boundary (not set)

Figure 14: Add inline policies to the IoTDeviceTemperatureAccess role

Now the actual lambda functions can be created. Enter "Lambda" in the management console and select the AWS Lambda service. Now click on the "Create function" button



Create policy

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more

DynamoDB (2 actions)

Service DynamoDB

Actions Read
GetItem
Write
PutItem

Resources close ● Specific ○ All resources

table arn:aws:dynamodb:eu-central-1:310226353119:table/IoTDeviceTen ☐ Any in this account

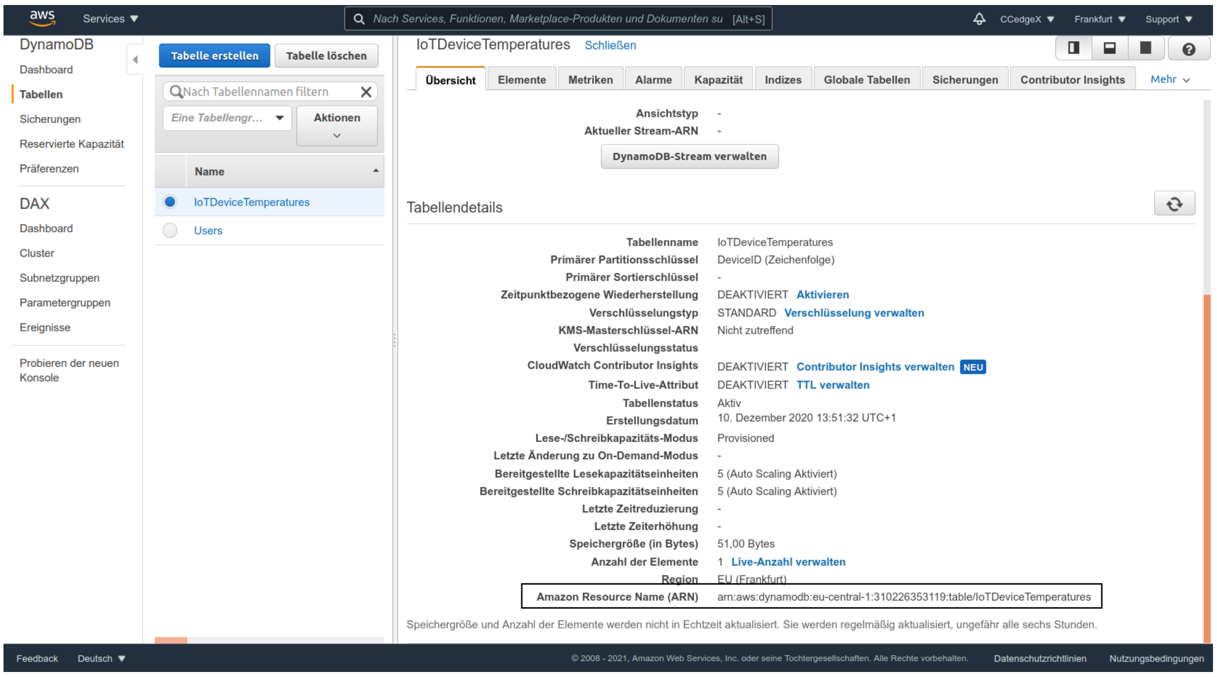
Add ARN to restrict access

Request conditions Specify request conditions (optional)

Character count: 440 of 10.240.
The current character count includes character for all inline policies in the role: IoTDeviceTemperatureAccess.

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 15: Add read and write access to a defined DynamoDB table



Name
● IoTDeviceTemperatures
○ Users

IoTDeviceTemperatures

Ansichtstyp	-
Aktueller Stream-ARN	-

Tabellendetails

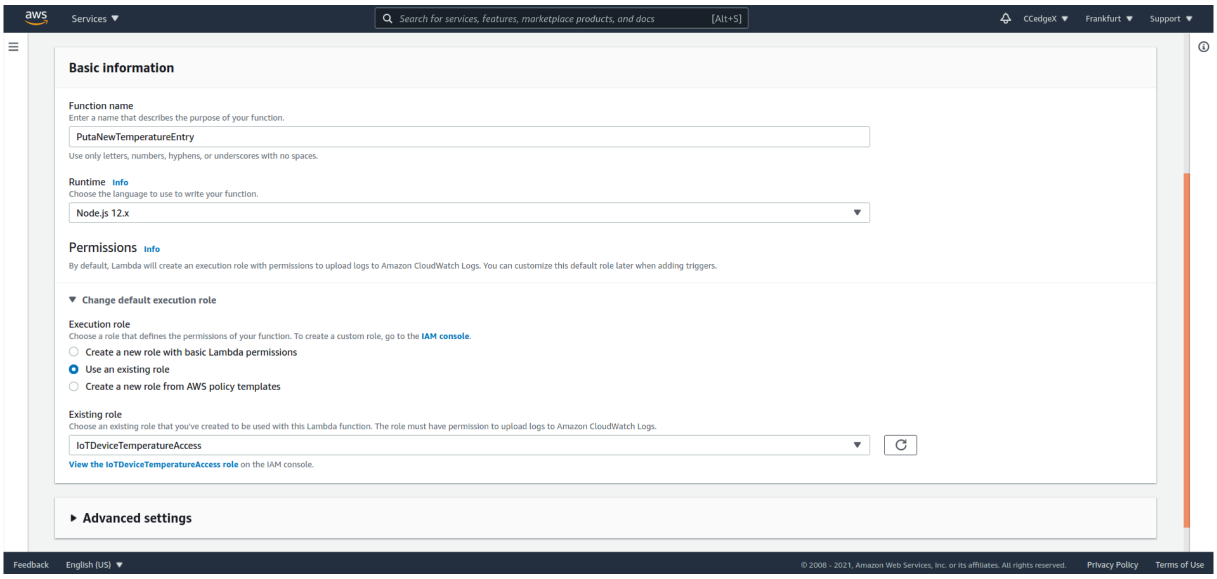
Tabellenname	IoTDeviceTemperatures
Primärer Partitionsschlüssel	DeviceID (Zeichenfolge)
Primärer Sortierschlüssel	-
Zeitpunktbezogene Wiederherstellung	DEAKTIVIERT Aktivieren
Verschlüsselungstyp	STANDARD Verschlüsselung verwalten
KMS-Masterschlüssel-ARN	Nicht zutreffend
Verschlüsselungsstatus	
CloudWatch Contributor Insights	DEAKTIVIERT Contributor Insights verwalten NEU
Time-To-Live-Attribut	DEAKTIVIERT TTL verwalten
Tabellenstatus	Aktiv
Erstellungsdatum	10. Dezember 2020 13:51:32 UTC+1
Lese-/Schreibkapazitäts-Modus	Provisioned
Letzte Änderung zu On-Demand-Modus	-
Bereitgestellte Lesekapazitätseinheiten	5 (Auto Scaling Aktiviert)
Bereitgestellte Schreibkapazitätseinheiten	5 (Auto Scaling Aktiviert)
Letzte Zeitreduzierung	-
Letzte Zeiterhöhung	-
Speichergröße (in Bytes)	51,00 Bytes
Anzahl der Elemente	1 Live-Anzahl verwalten
Region	EU (Frankfurt)
Amazon Resource Name (ARN)	arn:aws:dynamodb:eu-central-1:310226353119:table/IoTDeviceTemperatures

Speichergröße und Anzahl der Elemente werden nicht in Echtzeit aktualisiert. Sie werden regelmäßig aktualisiert, ungefähr alle sechs Stunden.

© 2008 - 2021, Amazon Web Services, Inc. oder seine Tochtergesellschaften. Alle Rechte vorbehalten.

Figure 16: "Amazon Resource Name of the IoTDeviceTemperatures table"

and enter "PutNewTemperatureEntry" as name. Lambda functions support multiple different programming languages. In this case JavaScript was used, therefore the selected runtime must be Node .js. In the role-dropdown, select "Choose an existing role" and select the newly created "IoTDeviceTemperatureAccess" role (as seen in figure 17).


Services
Search for services, features, marketplace products, and docs [Alt+S]
CCedgeX
Frankfurt
Support

Basic information

Function name
Enter a name that describes the purpose of your function.
PutaNewTemperatureEntry
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.
Node.js 12.x

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
- Create a new role with basic Lambda permissions
- [✓ Use an existing role]
- Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
IoTDeviceTemperatureAccess
View the IoTDeviceTemperatureAccess role on the IAM console.

Advanced settings

Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Figure 17: Create a lambda function using the DynamoReadWriteAccess policy

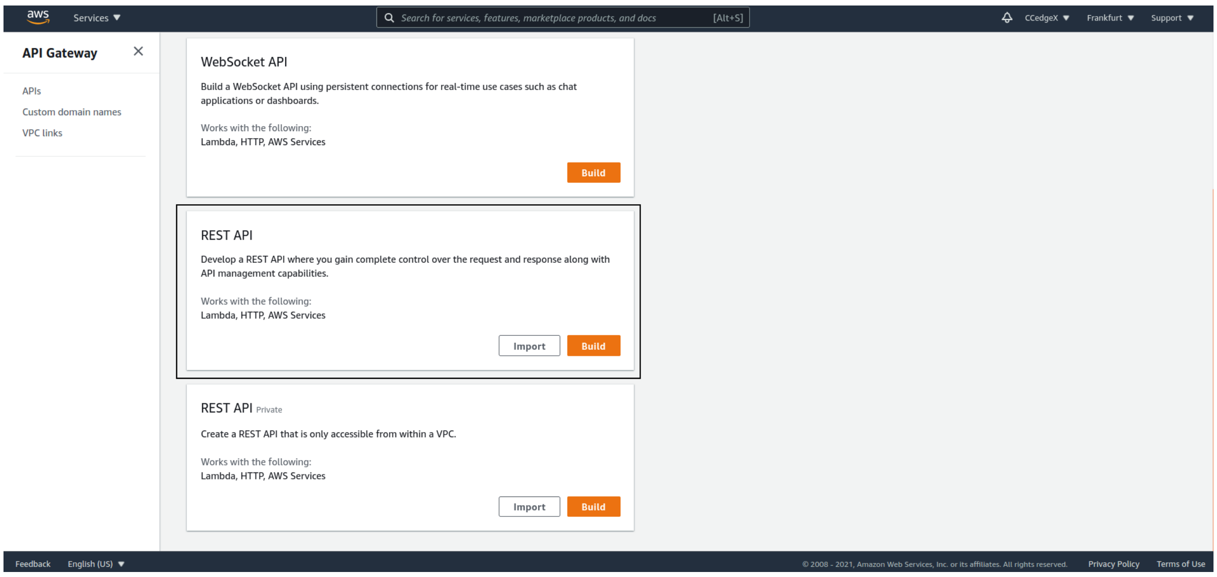
Paste the code from the appendix A.5 into the index.js file. When the script is triggered, it will receive an event object that contains a json body with the entries "DeviceID" and "Temperature". After these two values are extracted, they are inserted in the DynamoDB table "IoTDeviceTemperatures". It also gives back a response object, that contains the http response status.[31]
Perform the same steps for creating the lambda function "GetAllEdgeDeviceEntries" but use the script in appendix A.4. The only difference in this script is, that it scans the "IoTDeviceTemperatures" table and returns all entries in the response body.

3.6.3 AWS Gateway API implementation

The Gateway API service supports RESTful-APIs as well as Websocket APIs. A RESTful API can be created by following these steps: [30]

- In the AWS-Managment console select the *API Gateway* service.
- Select the button "Create API" and select REST API (see figure 18).
- Choose the API-Name as "EdgeXSensorData" and keep the endpoint setting on regional. Select create API.
- Create the wanted resources and add all methods that shall be supported by them.
- Implement some kind of access protection.

- Deploy the API.



WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

REST API Private

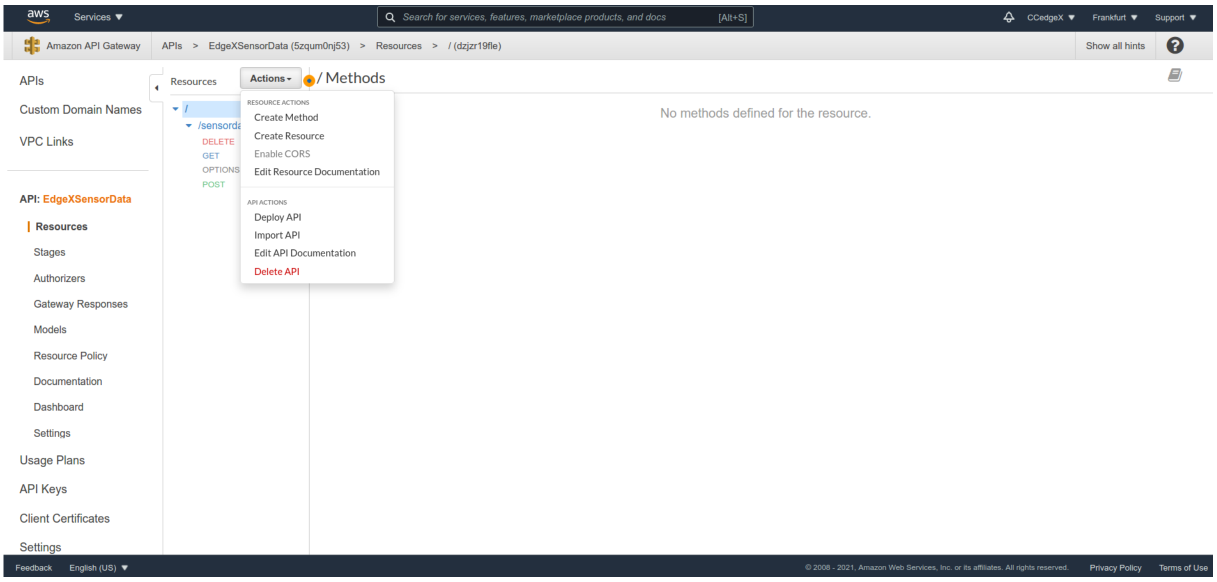
Create a REST API that is only accessible from within a VPC.

Works with the following:
Lambda, HTTP, AWS Services

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 18: Create a RESTfull API with Gateway API

In this tutorial the root-resource "sensordata" which supports the methods "GET" and "POST" needs to be created. For creating the resource select the "Action" button and choose the option "Create resource" (see figure 19). Keep the "configure as proxy resource" as well as the "Enable API Gateway CORS" unchecked and select the button "Create resource". [30]

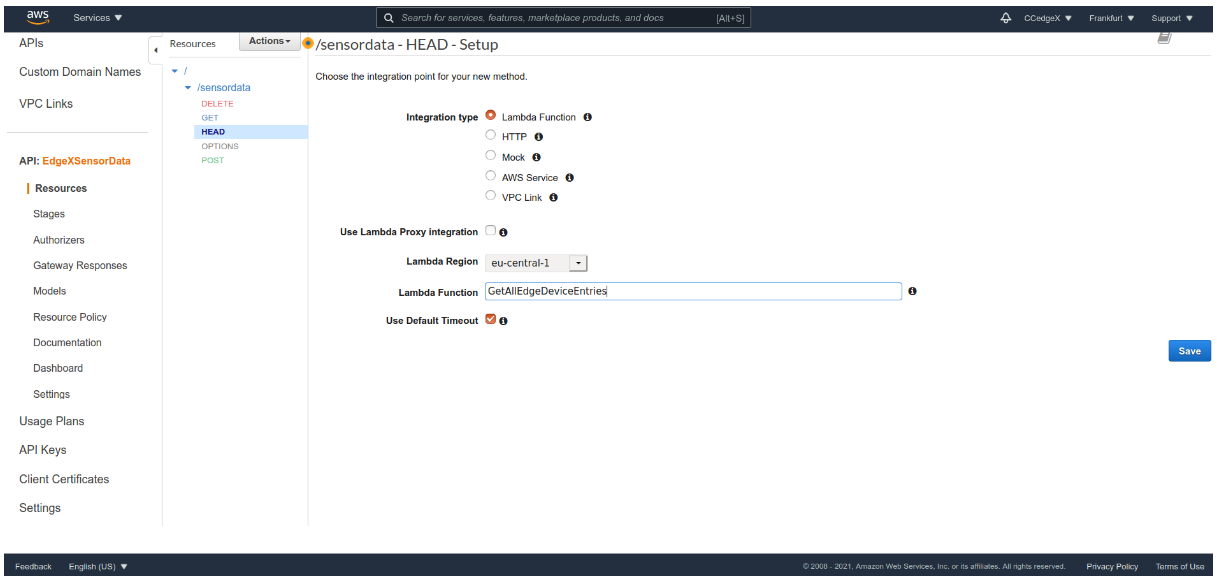


No methods defined for the resource.

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 19: Create a Resource, a Method or Deploy API

Subsequently each of the possible HTTP methods can be defined. First select the resource "sensordata", then press the "Action" button and choose "create method". A dropdown selection will appear below the resource, where the HTTP method can be selected. For this use case the GET and POST methods needs to be implemented. Select either of the methods and define it in the upcoming view. Keep the integrationstype on "Lambda-function" and select "eu-central-1" as location as shown in figure 20. When any character is typed in the Lambda-function field, the lambda functions from that region are listed. For the GET method choose the "GetAllEdgeDeviceEntries" and for the POST method the "PutNewTemperatureEntry" lambda function.[30]



/sensordata - HEAD - Setup

Choose the integration point for your new method.

Integration type: [✓ Lambda Function] HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration: ☐

Lambda Region: eu-central-1

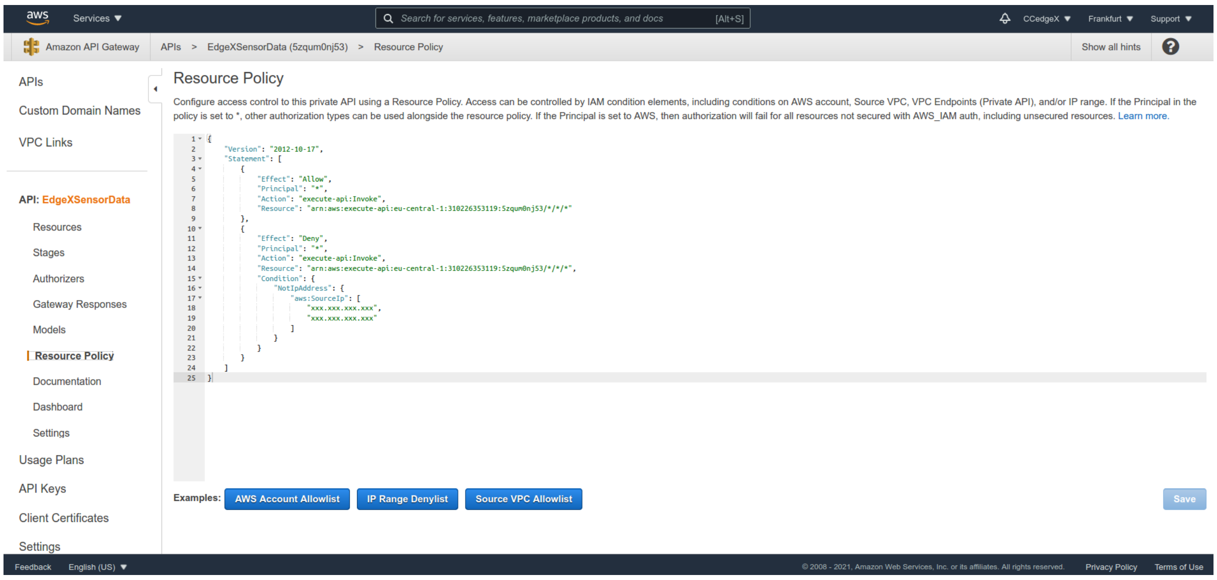
Lambda Function: GetAllEdgeDeviceEntries

Use Default Timeout: ✓

Figure 20: Create the HTTP Method HEAD

Once deployed, anyone could access the resource provided by the API Gateway service. Since only a certain amount of accesses is free it is highly advisable to use some kind of access regulation. Amazon recommends to use the *IAM service* for this purpose. For this project however, resource policies were used. They are more straight forward, since the access can be limited to one or multiple IP addresses. The negative aspect is that IP addresses of a personal router changes from time to time and therefore these guidelines have to be updated frequently. To implement a policy, select the "Resource policy" option on the left hand side (as marked in figure 21), among the options from the EdgeXSensorData API. Paste the code form appendix A.6 in there and enter the ip addresses that shall have access through this gateway.[1]

Finally select the button "Action" and choose "Deploy API" (figure 19). As a stage



Resource Policy

Configure access control to this private API using a Resource Policy. Access can be controlled by IAM condition elements, including conditions on AWS account, Source VPC, VPC Endpoints (Private API), and/or IP range. If the Principal in the policy is set to *, other authorization types can be used alongside the resource policy. If the Principal is set to AWS, then authorization will fail for all resources not secured with AWS_IAM auth, including unsecured resources. Learn more.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-central-1:310226353119:5zqum0nj53/*/*/*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-central-1:310226353119:5zqum0nj53/*/*/*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "xxx.xxx.xxx.xxx",
                        "xxx.xxx.xxx.xxx"
                    ]
                }
            }
        }
    ]
}
```

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 21: Set Gateway API resource policy

name choose "EdgeXCollectedData". After "create stage" was selected a description of the stage is presented, including a URL for accessing this API.
For testing purposes, the python scripts located in appendix A.8 and appendix A.7 can be used for either creating a new entry or getting all existing. Before they are executed, the url in the script has to be set to the Gateway API URL with the "sensordata" resource at the end.

References

[1] Amazon Web Services, Inc. How do i use a resource policy to allow certain ip addresses to access my api gateway rest api?, . URL `https://aws.amazon.com/de/premiumsupport/knowledge-center/api-gateway-resource-policy-access/`. [Accessed on 2021-01-15].

[2] Miel Donkers. Simple python 3 http server for logging all get and post requests. URL `https://gist.github.com/mdonkers/63e115cc0c79b4f6b8b3a6b797e485c7`. [Accessed on 2021-01-19].

[3] EdgeX Foundry. Edgex foundry documentation - introduction, . URL `https://docs.edgexfoundry.org/1.2/`. [Accessed on 2021-01-18].

[4] EdgeX Foundry. Edgex foundry - hands on tutorial, . URL `https://docs.edgexfoundry.org/1.2/examples/LinuxTutorial/LinuxTutorial/`. [Accessed on 2021-01-18].

[5] Anatol Badach. Internet of things – iot. 2014. doi: 10.13140/RG.2.1.5157.5527.

[6] EdgeX Foundry. Edgex foundry documentation - container names, . URL `https://docs.edgexfoundry.org/1.2/general/ContainerNames/`. [Accessed on 2021-01-19].

[7] EdgeX Foundry. Edgex foundry documentation - service layers, . URL `https://docs.edgexfoundry.org/1.2/#edgex-foundry-service-layers`. [Accessed on 2021-01-18].

[8] EdgeX Foundry. Edgex foundry documentation - core services, . URL `https://docs.edgexfoundry.org/1.2/microservices/core/Ch-CoreServices/`. [Accessed on 2021-01-19].

[9] EdgeX Foundry. Edgex foundry documentation - supporting services, . URL `https://docs.edgexfoundry.org/1.2/microservices/support/Ch-SupportingServices/`. [Accessed on 2021-01-18].

[10] EdgeX Foundry. Edgex foundry documentation - kuiper rules engine, . URL `https://docs.edgexfoundry.org/1.2/microservices/support/Kuiper/Ch-Kuiper/`. [Accessed on 2021-01-18].

[11] EdgeX Foundry. Edgex foundry documentation - application service, . URL https://docs.edgexfoundry.org/1.2/microservices/application/ApplicationServices/. [Accessed on 2021-01-19].

[12] EdgeX Foundry. Edgex foundry wiki: Device services - existing and work underway, . URL https://wiki.edgexfoundry.org/display/FA/Device+Services+-+existing+and+work+underway. [Accessed on 2021-01-19].

[13] EdgeX Foundry. Edgex foundry documentation - device services, . URL https://docs.edgexfoundry.org/1.2/microservices/device/Ch-DeviceServices/. [Accessed on 2021-01-19].

[14] Redis Labs. Redis.io homepage. URL https://redis.io/. [Accessed on 2021-01-19].

[15] EdgeX Foundry. Edgex foundry documentation - web user interface, . URL https://docs.edgexfoundry.org/1.2/microservices/configuration/Ch-Configuration/#web-user-interface. [Accessed on 2021-01-21].

[16] HashiCorp. Consul - explore the consul ui. URL https://learn.hashicorp.com/tutorials/consul/get-started-explore-the-ui. [Accessed on 2021-01-21].

[17] EdgeX Foundry. Edgex foundry documentation: Platform requirements, . URL https://docs.edgexfoundry.org/1.2/general/PlatformRequirements/. [Accessed on 2021-01-19].

[18] EdgeX Foundry. Edgex github repository, . URL https://github.com/edgexfoundry/developer-scripts/tree/master/releases. [Accessed on 2021-01-19].

[19] Docker Docs. Install docker engine. URL https://docs.docker.com/engine/install/ubuntu/. [Accessed on 2020-12-12].

[20] Rohan Sawant. Installing docker and docker compose on the raspberry pi in 5 simple steps. URL https://dev.to/rohansawant/installing-docker-and-docker-compose-on-the-raspberry-pi-in-5-simple-steps-3mgl. [Accessed on 2020-12-12].

[21] EdgeX Foundry. Edgex foundry walkthrough - query events / readings, . URL https://docs.edgexfoundry.org/1.2/walk-through/Ch-WalkthroughReading/#walkthrough-query-eventsreadings. [Accessed on 2021-01-20].

[22] EdgeX Foundry. Edgex foundry documentation kuiper rules engine, . URL https://docs.edgexfoundry.org/1.2/microservices/support/Kuiper/Ch-Kuiper/. [Accessed on 2021-01-19].

[23] EMQ Technologies Co. Stream specs. URL https://docs.emqx.io/en/kuiper/latest/sqls/streams.html#data-types. [Accessed on 2021-01-19].

[24] EdgeX Foundry. Edgex source, . URL https://github.com/emqx/kuiper/blob/master/docs/en_US/rules/sources/edgex.md. [Accessed on 2021-01-19].

[25] EMQ Technologies Co. Kuiper has officially become the edgex rule engine, . URL https://www.emqx.io/blog/kuiper-becomes-edgex-rule-engine. [Accessed on 2021-01-19].

[26] EdgeX Foundry. How to use meta function to extract addtional data from edgex message bus?, . URL https://github.com/emqx/kuiper/blob/master/docs/en_US/edgex/edgex_meta.md. [Accessed on 2021-01-19].

[27] EMQ Technologies Co. Rules management, . URL https://docs.emqx.io/en/kuiper/latest/restapi/rules.html#create-a-rule. [Accessed on 2021-01-19].

[28] Amazon Web Services, Inc. How to use the new amazon dynamodb key diagnostics library to visualize and understand your application's traffic patterns, . URL https://aws.amazon.com/de/blogs/compute/introducing-amazon-api-gateway-private-endpoints/. [Accessed on 2021-01-19].

[29] Chan Ryan, Elhemali Mostafa, Malligarjuna Padma. How to use the new amazon dynamodb key diagnostics library to visualize and understand your application's traffic patterns. URL https://aws.amazon.com/de/blogs/database/how-to-use-the-new-amazon-dynamodb-key-diagnostics-library-to-visualize-and-understand-your-applications-traffic-patterns/. [Accessed on 2021-01-19].

[30] Amazon Web Services, Inc. Tutorial: Hello world rest-api mit lambda-proxy-integration erstellen, . URL https://docs.aws.amazon.com/de_de/apigateway/latest/developerguide/api-gateway-create-api-as-simple-proxy-for-lambda.html. [Accessed on 2021-01-15].

[31] Amazon Web Services, Inc. Aws lambda und dynamodb — aws serverless tutorial — part i, . URL https://www.youtube.com/watch?v=VGerk8hrP9U&list=PLD_RqipWO-9uDz_KkexA5eGwd3cm3maq7. [Accessed on 2021-01-19].

[32] Amazon Web Services, Inc. Amazon dynamodb, . URL `https://aws.amazon.com/de/dynamodb/`. [Accessed on 2021-01-19].

[33] Amazon Web Services, Inc. Using aws lambda with amazon api gateway, . URL `https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html`. [Accessed on 2021-01-19].

[34] Amazon Web Services, Inc. Fakturierungs- und kostenverwaltungs-dashboard, . URL `https://console.aws.amazon.com/billing/home?#/`. [Accessed on 2021-01-15].

[35] Amazon Web Services, Inc. Kostenloses kontingent für aws, . URL `https://aws.amazon.com/de/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc`. [Accessed on 2021-01-15].

A Appendix

A.1 Code for Raspberry-Node

```
# ------------------------------
#           .-._
#         .-| | |
#       _ | | | |__FRANKFURT
#    ((__| | | |  UNIVERSITY
#        OF APPLIED SCIENCES
# ------------------------------
# Version: 1.1
# Authors: Dominic Gibietz, Jan Wagner, Daniel Helmer
# Student research project in the master's program in Allgemeine Informatik
# of the Department of Computer Science and Engineering

import sys, time, requests, json, subprocess, re, os
from gpiozero import CPUTemperature

ip_adress_edge_device = "*.*.*.*"

print("------------------------------")
print("          .-._                ")
print("        .-| | |               ")
print("      _ | | | |__FRANKFURT    ")
print("   ((__| | | |  UNIVERSITY    ")
print("       OF APPLIED SCIENCES    ")
print("------------------------------")


while True:

        # CPU temperature
        cpu = int(CPUTemperature().temperature)
        print("CPU temperature is "+ str(cpu)+ " Grad Celsius")

        # Memory use
        s = subprocess.check_output(["free","-m"])
        lines = s.split('\n')
        RAMav = lines[2].split()[3]
        RAMto = lines[1].split()[1]
        print("Available ram "+ str(lines[2].split()[3])+ "Mb")
        print("Total ram "+ str(lines[1].split()[1])+ "Mb")

        # Network traffic
        output = subprocess.Popen(['ifconfig', 'eth0'], stdout=subprocess.PIPE).communicate()[0]
        rx_bytes = re.findall('RX packets ([0-9]*) * bytes ([0-9]*)', output)[0]
        tx_bytes = re.findall('TX packets ([0-9]*) * bytes ([0-9]*)', output)[0]
        print("RX packets "+ str(rx_bytes[0])+ "Mb")
        print("RX bytes "+ str(rx_bytes[1])+ "Mb")
        print("TX packets "+ str(tx_bytes[0])+ "Mb")
        print("TX bytes "+ str(tx_bytes[1])+ "Mb")

        # Raspi load
        load=os.getloadavg()
        print("Load average over the last 1 minute:" + str(load[0]*100) )
        print("Load average over the last 5 minute:" + str(load[1]*100) )
        print("Load average over the last 15 minute:"+ str(load[2]*100) )

        # Raspi uptime
        s = subprocess.check_output(["uptime"])
        load_split = s.split('load average: ')
        load_five = float(load_split[1].split(',')[1])
        up = load_split[0]
        up_pos = up.rfind(',',0,len(up)-4)
        up = up[:up_pos].split('up ')[1]
        print("uptime "+str(up))

        # CPU info
        # f = os.popen('cat /proc/cpuinfo')
        # cpu = f.read()

        print("—————————————————————————————————")

        # Adresses for variables
        urlCPUTemp = 'http://%s:49986/api/v1/resource/Raspi_Device_1/CPU_Temperature' %
             ip_adress_edge_device
        urlRAM  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RAM' % ip_adress_edge_device
        urlRAMto  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RAM_Total' % ip_adress_edge_device
        urlRX_packets  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RX_packets' %
             ip_adress_edge_device
        urlRX_bytes  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/RX_bytes' %
             ip_adress_edge_device
        urlTX_packets  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/TX_packets' %
             ip_adress_edge_device
        urlTX_bytes  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/TX_bytes' %
             ip_adress_edge_device
        urlLoad1  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load1' % ip_adress_edge_device
        urlLoad2  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load2' % ip_adress_edge_device
        urlLoad3  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/Load3' % ip_adress_edge_device
        urlup  = 'http://%s:49986/api/v1/resource/Raspi_Device_1/UpTime' % ip_adress_edge_device

        # Edge headers
        headers = {'content-type': 'application/json'}

        # Send data
        response = requests.post(urlCPUTemp, data=json.dumps(cpu), headers=headers, verify=False)

        ## Commented out for presentation ##
        # response = requests.post(urlRAM, data=RAMav, headers=headers, verify=False)
        # response = requests.post(urlRAMto, data=RAMto, headers=headers, verify=False)
        # response = requests.post(urlRX_packets, data=rx_bytes[0], headers=headers, verify=False)
        # response = requests.post(urlRX_bytes, data=rx_bytes[1], headers=headers, verify=False)
        # response = requests.post(urlTX_packets, data=tx_bytes[0], headers=headers, verify=False)
        # response = requests.post(urlTX_bytes, data=tx_bytes[1], headers=headers, verify=False)
        # response = requests.post(urlLoad1, data=json.dumps(int(load[0]*100)), headers=headers, verify=
             False)
        # response = requests.post(urlLoad2, data=json.dumps(int(load[1]*100)), headers=headers, verify=
             False)
        # response = requests.post(urlLoad3, data=json.dumps(int(load[2]*100)), headers=headers, verify=
             False)
        # response = requests.post(urlup, data=json.dumps(up)*100, headers=headers, verify=False)

        time.sleep(5)
```

A.2 Code for Raspberry EdgeX Bash file

```

#!/bin/bash

docker-compose up -d

docker-compose ps

docker-compose up -d

echo "Docker-Compose running"

sleep 1s

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "CPU_Temperature",    "description": "CPU Temperature in Celsius",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "CPU_Temperature",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "CPU_Temperature"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "RAM",    "description": "RAM in GB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "RAM",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "RAM"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "RAM_Total",    "description": "RAM total in GB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "RAM_Total",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "RAM_Total"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "RX_packets",    "description": "RX_packets in MB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "RX_packets",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "RX_packets"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "RX_bytes",    "description": "RX_bytes in MB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "RX_bytes",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "RX_bytes"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "TX_packets",    "description": "TX_packets in MB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "TX_packets",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "TX_packets"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "TX_bytes",    "description": "TX_bytes in MB",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "TX_bytes",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "TX_bytes"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "Load1",    "description": "Raspi Load 1 min",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "Load1",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "Load1"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "Load2",    "description": "Raspi Load 5 min",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "Load2",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "Load2"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "Load3",    "description": "Raspi Load 15 min",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "Load3",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "Load3"    ]}'

curl --location --request POST 'http://localhost:48080/api/v1/valuedescriptor' \--header 'Content-Type: application/json' \--data-raw '{    "name": "UpTime",    "description": "Raspi UpTime",    "min": "",    "max": "",    "type": "Int64",    "uomLabel": "UpTime",    "defaultValue": "0",    "formatting": "%s",    "labels": [        "environment",        "UpTime"    ]}'

curl --location --request POST 'http://localhost:48081/api/v1/deviceprofile/uploadfile' --form 'file=@"./Raspi_Device_Profile.yaml"'

echo "Create devices"
```

```

sleep 2s

curl --location --request POST 'http://localhost:48081/api/v1/device' \--header 'Content-Type:
     application/json' \--data-raw '{"name": "Raspi_Device_1", "description": "Raspberry Pi 1", "
     adminState": "unlocked", "operatingState": "enabled", "protocols": {"example": {"host": "dummy", "
     port": "1234", "unitID": "1"}}, "labels": ["UpTime", "Load1", "Load2", "Load3", "RX_packets", "
     RX_bytes", "TX_packets", "TX_bytes", "RAM_Total", "RAM", "CPU_Temperature sensor", "DHT11"], "
     location": "Berlin", "service": {"name": "edgex-device-rest"}, "profile": {"name": "Raspi_Device
     "}}'

echo "Created device 1"

curl --location --request POST 'http://localhost:48081/api/v1/device' \--header 'Content-Type:
     application/json' \--data-raw '{"name": "Raspi_Device_2", "description": "Raspberry Pi 2", "
     adminState": "unlocked", "operatingState": "enabled", "protocols": {"example": {"host": "dummy", "
     port": "1234", "unitID": "1"}}, "labels": ["UpTime", "Load1", "Load2", "Load3", "RX_packets", "
     RX_bytes", "TX_packets", "TX_bytes", "RAM_Total", "RAM", "CPU_Temperature sensor", "DHT11"], "
     location": "Berlin", "service": {"name": "edgex-device-rest"}, "profile": {"name": "Raspi_Device
     "}}'

echo "Created device 2"

echo "------------------------------"

echo "                                                  "

echo "          .-._                                    "

echo "        .-| | |                                   "

echo "      _ | | | |__FRANKFURT  "

echo "  ((__| | | | UNIVERSITY  "

echo "       OF APPLIED SCIENCES  "

echo "                                                  "

echo "------------------------------"

echo "Start your work and have fun! :) "

echo "------------------------------"
```

A.3 Code for Raspberry Profile

```

name: "Raspi_Device"
manufacturer: "FRA-UAS"
model: "Raspberry Pi 4"
labels:
        - "rpi"
description: "Raspberry Pi Profile"

deviceResources:
        -
        name: CPU_Temperature
        description: "CPU temperature values"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}
        -
        name: RAM
        description: "RAM available values"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: RAM_Total
        description: "RAM total values"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: RX_packets
        description: "RX packets"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: RX_bytes
        description: "RX bytes"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: TX_packets
        description: "TX packets"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: TX_bytes
        description: "TX bytes"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: Load1
        description: "TX bytes"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: Load2
        description: "TX bytes"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: Load3
        description: "TX bytes"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}

        -
        name: UpTime
        description: "Raspi UpTime"
        properties:
        value:
        { type: "Int64", readWrite: "RW", minimum: "", maximum: "", size: "4", LSB: "true",
            defaultValue: "9"}
```

A.4 Code for AWS Lambda GetAllEdgeDeviceEntries

```
'use strict';
const AWS = require('aws-sdk');

AWS.config.update({region: "eu-central-1"});
exports.handler = async (event, context) => {
    console.log('PutNewTemperatureEntry invoked');
    const documentClient = new AWS.DynamoDB.DocumentClient({region: "eu-central-1"});

    let responseBody ="";
    let statusCode = 0;

    const params = {
        TableName: "IoTDeviceTemperatures"
    };

    try
    {
     const data = await documentClient.scan(params).promise();
     responseBody = JSON.stringify(data.Items);
     console.log("Entry successfully created: " + responseBody);
     statusCode = 200;
    } catch (e)
    {
        responseBody = 'Unable to get entries:' + e;
        statusCode = 403;
        console.log("Execution failed " + e);
    }

    const response =
    {
    statusCode: statusCode,
    headers: {
      "Content-Type": "application/json",
      "access-control-allow-origin": "*"
    },
    body: responseBody
  };

    return response;
};
```

A.5 Code for AWS Lambda PutNewTemperatureEntry

```
'use strict';
const AWS = require('aws-sdk');

AWS.config.update({region: "eu-central-1"});
exports.handler = async (event, context) => {
    console.log('PutNewTemperatureEntry invoked');
    const ddb = new AWS.DynamoDB({apiVersion: "2012-10-08"});
    const documentClient = new AWS.DynamoDB.DocumentClient({region: "eu-central-1"});

    let responseBody ="";
    let statusCode = 0;

    const {DeviceID, Temperature} = JSON.parse(event.body);

    const params = {
        TableName: "IoTDeviceTemperatures",
        Item: {
            DeviceID: DeviceID,
            Temperature: Temperature
        }
    };
    try
    {
     const data = await documentClient.put(params).promise();
     responseBody = JSON.stringify(data);
     console.log("Entry successfully created: " + responseBody);
     statusCode = 201;
    } catch (e)
    {
        responseBody = 'Unable to create entry: ${e}'
        statusCode = 403;
        console.log("Execution failed " + e);
    }

    const response =
    {
    statusCode: statusCode,
    headers: {
      "Content-Type": "application/json"
    },
    body: responseBody
  };

    return response;
};
```

A.6 Code for AWS Gateway API resource policy [1]

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-central-1:310226353119:5zqum0nj53/*/*/*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-central-1:310226353119:5zqum0nj53/*/*/*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "xxx.xxx.xxx.xxx",
                        "xxx.xxx.xxx.xxx"
                    ]
                    }
                }
            }
        ]
}
```

A.7 Script for getting all DynamoDB entries

```
#!/usr/bin/env python3
import requests, json

awsIP ="<AmazonGWAPI_URL>/EdgeXCollectedData/sensordata"
response = requests.get(awsIP)
print(response.text)
```

A.8 Script for creating new DynamoDB entry

```
#!/usr/bin/env python3
import requests, json, sys

awsIP ="<AmazonGWAPI_URL>/EdgeXCollectedData/sensordata"
if len(sys.argv) != 4:
        print ("The function needs 3 parameter")
else:
        headers = {'content-type': 'application/json'}
        body = {"DeviceID": sys.argv[1], "Temperature": sys.argv[2]}
        response = requests.post(awsIP, data=json.dumps(body), headers=headers)
        print(response.text)
```

A.9 Python 3 HTTP server [2]

```
#!/usr/bin/env python
"""
Very simple HTTP server in python (Updated for Python 3.7)
Usage:
    ./dummy-web-server.py -h
    ./dummy-web-server.py -l localhost -p 8000
Send a GET request:
    curl http://localhost:8000
Send a HEAD request:
    curl -I http://localhost:8000
Send a POST request:
    curl -d "foo=bar&bin=baz" http://localhost:8000
"""
import argparse
from http.server import HTTPServer, BaseHTTPRequestHandler


class S(BaseHTTPRequestHandler):
    def _set_headers(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

    def _html(self, message):
        """This just generates an HTML document that includes `message`
        in the body. Override, or re-write this do do more interesting stuff.
        """
        content = "<html><body><h1>%s</h1></body></html>" % message
        return content.encode("utf8")  # NOTE: must return a bytes object!

    def do_GET(self):
```

```python
        self._set_headers()
        self.wfile.write(self._html("hi!"))

    def do_HEAD(self):
        self._set_headers()

    def do_POST(self):
        # Doesn't do anything with posted data
        content_length = int(self.headers['Content-Length']) # <--- Gets the size of data
        post_data = self.rfile.read(content_length) # <--- Gets the data itself
        print("POST request,\nPath: {0}\nHeaders:\n{1}\n\nBody: \n{2}\n".format(str(self.path), str(self.headers), post_data.decode('utf-8')))
        self._set_headers()
        self.wfile.write(self._html("POST!"))


def run(server_class=HTTPServer, handler_class=S, addr="localhost", port=8000):
    server_address = (addr, port)
    httpd = server_class(server_address, handler_class)

    print("Starting httpd server on %s:%s" % (addr,port))
    httpd.serve_forever()


if __name__ == "__main__":

    parser = argparse.ArgumentParser(description="Run a simple HTTP server")
    parser.add_argument(
        "-l",
        "--listen",
        default="0.0.0.0",
        help="Specify the IP address on which the server listens",
    )
    parser.add_argument(
        "-p",
        "--port",
        type=int,
        default=8000,
        help="Specify the port on which the server listens",
    )
    args = parser.parse_args()
    run(addr=args.listen, port=args.port)
```

Eigenständigkeitserklärung

Ich versichere, dass die vorstehende Arbeit von mir selbstständig ohne unerlaubte fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde, und dass ich alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, als solche gekennzeichnet habe.

Frankfurt, den 22nd January, 2021