MedTech
MEDITERRANEAN
INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

CS 453: Cloud and pervasive computing

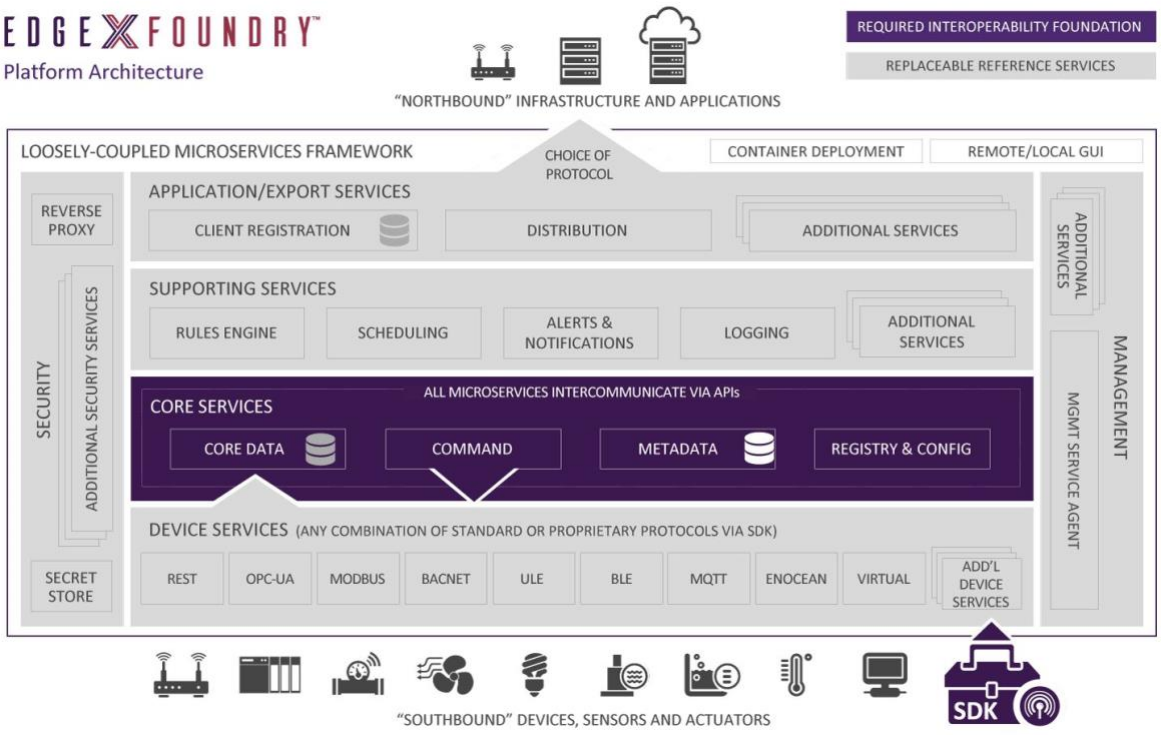
Session: Fall 2021

Professor: Amina Mseddi

Lab 6: EdgeX

EdgeX presentation

EdgeX Foundry is an open source IoT solution ideally used for ingesting data from multiple sources and then forwarding that data to a central system. It natively speaks multiple protocols used by IoT devices, like BACNET, OPC-UA, MQTT and REST. It can also be configured to match individual data formats used by devices from different vendors by using device profiles. EdgeX Foundry is made up of a collection of micro services, each of which runs in a container. Microservices communicate with each other over REST API interfaces.



EdgeX Foundry can convert source data from proprietary data formats into XML or JSON, encrypt, compress, and finally forward that data to an external source over MQTT or other protocol. Data is normally not retained long-term by EdgeX Foundry itself.

Depending on protocol, sending commands is also supported. Therefore, it's possible to use EdgeX as an intermediary when wanting to communicate with a device. By using the REST API provided by the command service, commands can be automatically translated by EdgeX from REST into the correct protocol and format expected by the end device.

Rules can be used to create logic for triggering actions based on input. For example, if value A goes above X, execute a pre-set command.

Installation of EdgeX Foundry would generally be done close to the sensors / data being generated. For example, on an edge gateway appliance. There could be thousands of these, each with its own EdgeX installation, ingesting, converting, and forwarding data to a central location.

EdgeX installation

1. Verify that docker and docker-compose are installed

```
$ which docker
$ which docker-compose
```

Which command shows the executable's location. If the command returns nothing then, you must install the packages.

Installing EdgeX Foundry

The microservices making up EdgeX Foundry are controlled by a docker-compose file in YAML format. It specifies how each microservice should run, its ports, volumes and dependencies.

Docker-compose manages the containers in the YAML file as a group. Commands starting with "docker-compose" are context sensitive and need to be executed in the same folder as the docker-compose.yml file.

a. Create a directory for the EdgeX Foundry docker-compose.yml file (Geneva release):

```
$ mkdir geneva
$ cd geneva
```

b. Use wget to download the docker-compose.yml file

```
$ wget  https://raw.githubusercontent.com/jonas-
werner/EdgeX_Tutorial/master/docker-compose_files/docker-
compose_step1.yml
$ cp docker-compose_step1.yml docker-compose.yml
```

c. Pull the containers and list the newly downloaded images

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

Starting EdgeX Foundry

a. Start EdgeX Foundry using docker-compose
   Make sure the commands below are executed in the same folder as the YAML file. Don't forget the "-d" at the end or the terminal will be flooded by log output from ALL the microservices. If this happens, press "CTRL+C" to shut down EdgeX Foundry.

```
$ sudo docker-compose up -d
```

b. View the running containers

```
$ sudo docker-compose ps
```

Note that the ports used by EdgeX are listed for each container. These ports are defined in the docker-compose.yml file along with many other settings.

Basic interaction

We will use curl to interact with EdgeX

a. We list the devices registered:

```
$ curl http://<edgex ip>:48082/api/v1/device
```

It may take a few seconds to complete and will result in some rather difficult to read output. Let's improve that.

b. Install jq to do pretty formatting of JSON output

```
$ sudo apt install jq
```

c. Issue the same curl command as before but add a pipe and the jq command:

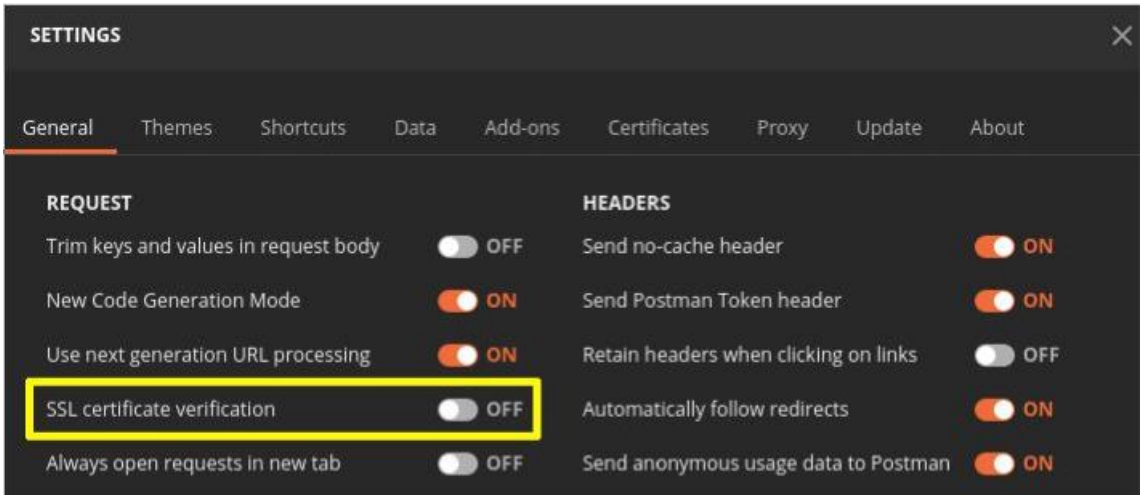
```
$ curl http://<edgex ip>:48082/api/v1/device | jq
```

   This will result in much prettier formatting, making it easy to see the sample devices which have already been created

Postman

Postman is an excellent tool for interacting with REST APIs in a graphical manner. It's also great since it saves the history for later reference and it also comes with the ability to save frequently used queries into collections. Install the stand-alone version of Postman rather than the Google Chrome plug-in, since the plug-in is outdated.

1. After installation, disable certificate verification:

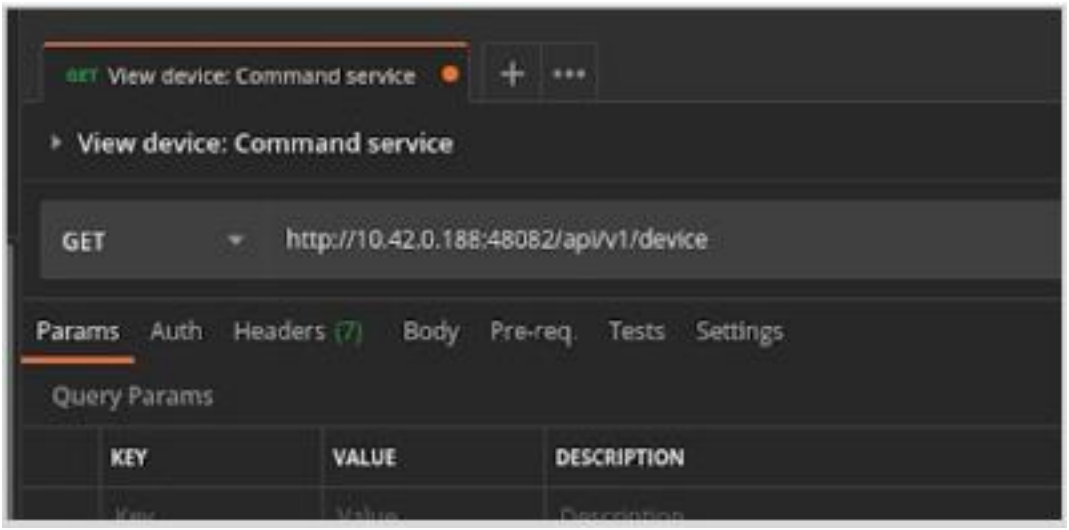


It's also possible to change the theme under the "Themes" tab

2. Set the method to "GET"
3. Enter the URI to be used:

   ```
   http://<edgex ip>:48082/api/v1/device
   ```

4. Push "Send"



5. The devices registered with EdgeX Foundry will be listed in the results pane. This is the exact same output as was received back when using cURL

Creating a device

A device could be any type of edge appliance which is generating or forwarding data. It could be an edge gateway in a factory with sensors of its own or an industrial PC hooked up to a PLC or any other device.

1. In this tutorial two types of devices will be created:
    1. Sensor cluster generating temperature and humidity data
    2. Generic device with a REST interface, supporting commands

    Two methods of device creation will be covered:

    3. Manual: The Sensor cluster will be created by issuing individual REST commands
    4. Scripted: The Generic device will be created instantly using a Python script

Introduction to Device Profiles

EdgeX incorporates device profiles as a way of easily adding new devices. A device profile is essentially a template which describes the device, its data formats, and supported commands. It is a text file written in YAML format which is uploaded to EdgeX and later referenced whenever a new device is created. Only one profile is needed per device type. Some vendors provide pre-written device profiles for their devices. In this tutorial custom device templates will be used.

Sensor cluster

This device will be created manually to showcase how to use the EdgeX Foundry REST APIs. The sensor cluster, which will be generating temperature and humidity data, will be created using Postman with the following steps:

- Create value descriptors
- Upload the device profile
- Create the device

Each step will include the same IP address - that of the host, and a port number. The port number determines which microservice is targeted with each command. For example:

- 48080: edgex-core-data
- 48081: edgex-core-metadata
- 48082: edgex-core-command

Create value descriptors

Value descriptors are what they sound like. They describe a value. They tell EdgeX what format the data comes in and what to label the data with. In this case value descriptors are created for temperature and humidity values respectively

Open Postman and use the following values:

```
Method: POST
URI: http://<edgex ip>:48080/api/v1/valuedescriptor
Payload settings: Set Body to “raw” and “JSON”
Payload data:
{ "name": "humidity",
  "description": "Ambient humidity in percent",
  "min": "0",
  "max": "100",
  "type": "Int64",
  "uomLabel": "humidity",
  "defaultValue": "0",
  "formatting": "%s",
  "labels": [
    "environment",
    "humidity"
    ]
}
```

Watch for the return code of “**200 OK**” and the ID of the newly created value descriptor. There is no need to make note of the ID.

Update the body and issue the command again for temperature:

```
Method: POST
URI: http://<edgex ip>:48080/api/v1/valuedescriptor
Payload settings: Set Body to “raw” and “JSON”
Payload data:
{ "name": "temperature",
  "description": "Ambient temperature in Celsius",
  "min": "-50",
  "max": "100",
  "type": "Int64",
  "uomLabel": "temperature",
  "defaultValue": "0",
  "formatting": "%s",
  "labels": [
  "environment",
  "temperature"
  ]
}
```

Upload a device profile

Get a copy of the device profile from here:

https://raw.githubusercontent.com/jonas-werner/EdgeX_Tutorial/master/deviceCreation/sensorClusterDeviceProfile.yaml

In Postman use the following settings:

Method: POST
URI: http://<edgex ip>:48081/api/v1/deviceprofile/uploadfile
Payload settings: This part is a bit tricky:

- Set Body to “form-data”
- Hover over KEY and select “File”
- Select the yaml file: sensorClusterDeviceProfile.yaml
- In the KEY field, enter “file” as key

Create the device

Now EdgeX is finally ready to receive the device creation command in Postman as follows:

Two items are particularly important in this JSON body:

- The device service “edgex-device-rest” is used since this is a REST device.
- The profile name “SensorCluster” must match the name in the device profile yaml file uploaded in the previous step.

In Postman use the following settings:

```
Method: POST
URI: http://<edgex ip>:48081/api/v1/device
Payload settings: Set Body to “raw” and “JSON”
Payload data:
{
  "name": "Temp_and_Humidity_sensor_cluster_01",
  "description": "Raspberry Pi sensor cluster",
  "adminState": "unlocked",
  "operatingState": "enabled",
  "protocols": {
    "example": {
```

```
      "host": "dummy",
      "port": "1234",
      "unitID": "1"
      }
    },
    "labels": [
      "Humidity sensor",
      "Temperature sensor",
      "DHT11"
      ],
    "location": "Tokyo",
    "service": {
      "name": "edgex-device-rest"},
    "profile": {"name": "SensorCluster" }
}
```

Sending data to EdgeX Foundry

EdgeX is now ready to receive temperature and humidity data. To begin with, the functionality can be tested by posting individual data values using Postman. The next step after that is to use a Python script to simulate data values continuously.

In Postman use the following settings to send a temperature value:

```
Method: POST
URI: http://<edgexip>:49986/api/v1/resource/Temp_and_Humidity_sensor_cluster_01/temperature
Payload settings: Set Body to “raw” and “text”
Payload data: 23 (any integer value will do)
```

View the data

Use Postman to view the data stored in the EdgeX Foundry Redis DB as follows

In Postman use the following settings to view the temperature value:

```
Method: GET
URI: http://<edgex ip>:48080/api/v1/reading
```

Generate sensor data with Python

It’s possible to use a simple Python script to generate simulated sensor data continuously.

1. On the Linux VM, clone the Git repository for this tutorial:

   ```
   git clone https://github.com/jonas-werner/EdgeX_Tutorial.git
   ```

2. Enter the directory containing the data simulation script

   ```
   cd EdgeX_Tutorial/sensorDataGeneration/
   ```

   The Python module “requests” need to be installed to run the script. It is advisable to install modules in a separate virtual Python environment. Fortunately it’s very quick to create one:

3. Install python3-venv

   ```
   sudo apt install python3-venv -y
   ```

4. Create a new virtual environment called simply “venv”

   ```
   python3 -m venv venv
   ```

5. Enter the virtual environment

   ```
   . ./venv/bin/activate
   ```

   or

   ```
   source ./venv/bin/activate
   ```

   Note that the terminal is now prefixed with the name of the virtual environment

6. Verify that the Python executable used is the one located in the virtual environment

   ```
   which python3
   ```

7. Install the requests module using pip

   ```
   pip install requests
   ```

8. If executing the script on any other host than the EdgeX Foundry VM, edit the file and change 127.0.0.1 to the IP address of the VM where EdgeX Foundry is installed
9. Run the script

   ```
   python3 ./genSensorData.py
   ```

Stopping EdgeX Foundry

The commands in this section need to be issued from the folder containing the docker-compose.yml file (“geneva” in this example).

1. To just stop the containers:

   ```
   docker-compose stop
   ```

2. To stop and remove the containers:

```
docker-compose down
```

3. To stop and remove containers + volumes (the original images will remain):

```
docker-compose down -v
```

Optional Adding graphical user interfaces

There are multiple UI's that can be added to EdgeX Foundry. To do this, simply add entries for them in the docker-compose.yml file. In this case Portainer will be added as a way to view and interact with the containers in a graphical manner.

Portainer

1. Open the docker-compose.yml file. Under the volumes section at the beginning of the file, add an entry for portainer as per the below:

```
volumes:

  db-data:

  log-data:

  consul-config:

  consul-data:

  portainer_data:
```

2. Under the "services" section, add the following entry for Portainer:

```
portainer:
  image: portainer/portainer
  ports:
    - "0.0.0.0:9000:9000"
  container_name: portainer
  command: -H unix:///var/run/docker.sock
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock:z
    - ${HOME}/docker/portainer_data:/data
```

Note: Be careful to match the indentation with the other services entries. If in doubt, compare to entries above or below to make sure the indentation matches.

3. Save and exit the editor
4. Start (or restart) EdgeX Foundry (with docker-compose up -d)
5. Portainer can now be accessed in a browser at: http://<edgex ip>:9000

Reference

EdgeX Foundry: A hands-on tutorial, A practical guide to getting started with open source IoT Author: Jonas Werner
Date: 2020-08-26
Version: 1.0
EdgeX Foundry version:Geneva