




Getting Started With EdgeX Foundry

Servisno-orientisane arhitekture

Agenda

- Installation of EdgeX Foundry
- Starting / stopping microservices
- How to add / enable services
- Interacting with EdgeX using Postman, cURL and Python
- Creating devices (sources of sensor data)
- Sending data to EdgeX using REST
- Exporting a stream of data using MQTT
- How to issue commands from EdgeX to devices
- Creating rules
- Debug flags and container logs





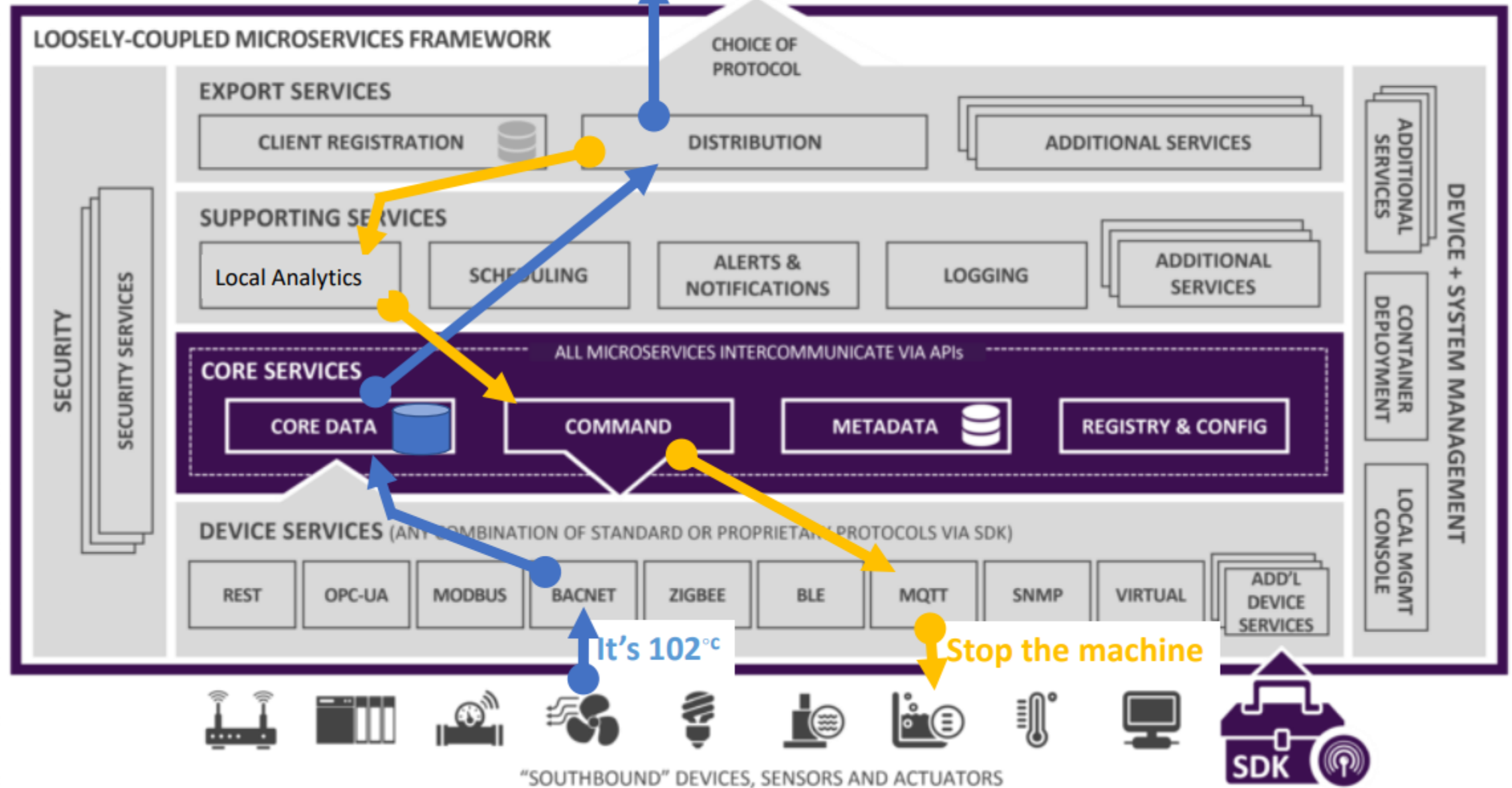
EdgeX Foundry Basics

Introducing EdgeX Foundry

- An open source, vendor neutral project (and ecosystem) 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.
 - A micro service, loosely coupled software framework for IoT edge computing
- Hardware and OS agnostic, Linux Foundation - Apache 2 project
- Goal: enable and encourage the rapidly growing community of IoT solutions providers to create an ecosystem of interoperable plug-and-play components
 - The community builds and maintains common building blocks and APIs
 - Certification of EdgeX components to ensure interoperability and compatibility
 - Collaborate with relevant open source projects, standards groups, and industry alliances to ensure consistency and interoperability across the IoT

EdgeX Primer - How it works

- A collection of a dozen+ micro services
 - Written in multiple languages (Java, Go, C, ...)
 - Each microservice runs in a container (via Docker and Docker Compose)
 - Microservices communicate with each other over REST API interfaces.
- EdgeX data flow:
 - Sensor data is collected by a **Device Service** from a thing
 - Data is passed to the **Core Services** for local persistence
 - Data is then passed to **Export Services** for transformation, formatting, filtering and can then be sent "north" to enterprise/cloud systems
 - Data is then available for edge analysis and can trigger device actuation through Command service
 - Many others services provide the supporting capability that drives this flow



EdgeX Foundry – Data Capabilities

- Can convert source data from proprietary data formats into XML or JSON
- Can 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
- Can be used as an intermediary for communication with a device over for example BACNET without building support for that protocol.
- Commands can be automatically translated by EdgeX from REST into the correct protocol and format expected by the end device
 - using the REST API provided by the command service
- 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.

EdgeX Architectural Tenets

- EdgeX Foundry must be platform agnostic with regard to hardware, OS, distribution/deployment, protocols/sensors
- EdgeX Foundry must be extremely flexible
- Any part of the platform may be upgraded, replaced or augmented by other micro services or software components
- Allow services to scale up and down based on device capability and use case
- EdgeX Foundry should provide “reference implementation” services but encourages best of breed solutions
- EdgeX Foundry must provide for store and forward capability (to support disconnected/remote edge systems)
- EdgeX Foundry must support and facilitate “intelligence” moving closer to the edge in order to address
 - Actuation latency concerns
 - Bandwidth and storage concerns
 - Operating remotely concerns
- EdgeX Foundry must support brown and green device/sensor field deployments
- EdgeX Foundry must be secure and easily managed



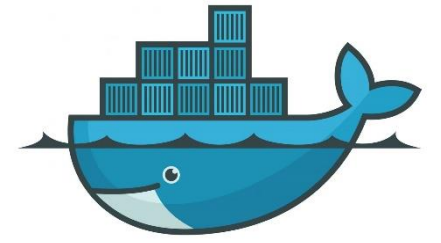
EdgeX Foundry Demo

EdgeX Foundry - Installation

- 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.
- While EdgeX runs as a collection of containerized microservices it doesn't currently support k8s.
 - Note that since the overhead for k8s can be prohibitive for low-powered edge nodes, lack of k8s support isn't necessarily a concern.

Options to Getting & Running EdgeX

- EdgeX micro services can be built and deployed in a number of ways
 - “Contributors Approach”
 - Get the raw code, build it, and deploy the services to the target platform(s) •
 - “Users Approach”
 - Get EdgeX Docker container images and deploy/run to a platform where Docker is installed
 - “Hybrid Approach”
 - Get, build and deploy some of the services on your own
 - Get and use Docker container images for the other services
 - Docker Compose is a tool to help get and run multiple containers
 - Docker Compose can be used with either the User or Hybrid approaches
- EdgeX Foundry Code is available at GitHub (<https://github.com/edgexfoundry/>)
 - See the EdgeX Getting Started guides for more directions on the contributor and hybrid approaches



User Approach to Get & Run

- The EdgeX community provides a Docker container image for each micro service (and underlying infrastructure such as the database)
 - This convenience allows users to quickly get pre-built EdgeX micro services
 - Because the container images have all the necessary environment (OS, configuration, etc.) for the micro services, it makes deploying EdgeX easier
 - The container images can be run on any platform that runs Docker
 - There are different container images for hardware platforms (Intel or Arm)
- The EdgeX Docker container images are available in Docker Hub (hub.docker.com)
 - The most recent code is always built to “developer” container images
 - These are made available from a Linux Foundation Nexus repository
 - These should only be used when you need the latest developer work

Tutorial prerequisites

- Ubuntu 20.04 (preferably a VM)
 - Any Linux OS supporting docker and docker-compose should work but the tutorial uses Ubuntu 20.04 and commands will reflect this
- Internet access
 - For downloading container images and sending data via MQTT
- Familiarity with Linux, general terminal commands and text editing tools
- Optional:
 - Raspberry Pi (for those who want to use a DHT sensor to send data to EdgeX)
 - An IDE like VS Code, Atom or similar to edit code and settings files

Installing Docker and docker-compose

1. SSH to the VM where EdgeX Foundry will be installed

2. Update system

```
sudo apt update
```

```
sudo apt upgrade -y
```

3. Install Docker-CE

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

```
sudo apt update
```

```
sudo apt install docker-ce -y
```

```
sudo usermod -aG docker ${USER}
```

4. Log out and back in again so the new permissions applied by “usermod” can take effect

5. Install docker-compose `sudo apt install docker-compose -y`

Note: Match “focal”
below with your
distribution if different
from 20.04 (check with
“lsb_release -a” if unsure
about the version)

Installing EdgeX Foundry

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

```
mkdir geneva
```

```
cd geneva
```

2. 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
```

3. Pull the containers and list the newly downloaded images

```
docker-compose pull
```

```
docker image ls
```

Starting EdgeX Foundry

1. Start EdgeX Foundry using docker-compose

docker-compose up -d

2. View the running containers

docker-compose ps

3. Compare the previous output with the output from the docker ps command

docker ps

Make sure the commands below are executed in the same folder as the YAML file

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

- Consul
 - View the status of the microservices via the Consul web interface.
 - Use a browser to access: <http://:8500/ui/dc1/services>
 - Consul can also be used to change configuration settings, for example switching on debugging



Basic Interaction

- cURL

1. Interact with EdgeX using curl. In this case we list the devices registered:

```
curl http://: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.

2. Install jq to do pretty formatting of JSON output

```
sudo apt install jq
```

Note the port used in the curl command.

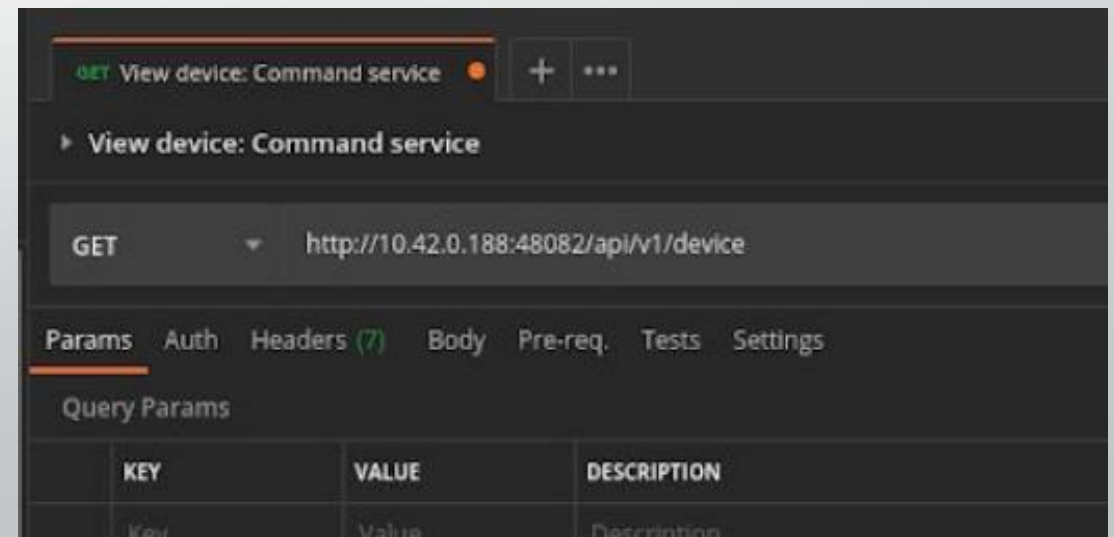
3. Issue the same curl command as before but add a pipe and the jq command:

```
curl http://:48082/api/v1/device | jq
```

```
{
  "id": "f14e3e10-de12-42ec-bff6-c32e48e073f0",
  "name": "sample-image",
  "adminState": "UNLOCKED",
  "operatingState": "ENABLED",
  "labels": [
    "rest",
    "binary",
    "image"
  ]
}
```

Basic Interaction

- Postman
 1. After installation, disable certificate verification
 2. Set the method to "GET"
 3. Enter the URI to be used: <http://: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



Stopping EdgeX Foundry

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`

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

Creating a device

Creating, or registering, a device in EdgeX Foundry is required for EdgeX to:

- Become aware of the existence of the device
- Be able to receive data from the device
- Be able to send commands to the device (if it supports commands)
- Understand what type of data the device will generate
- Understand how and in what format the device sends data / receives commands.
 - For example:
 - What protocol is used?
 - What types of data is supported (temp, humidity, vibrations/sec, etc.)?
 - What format does the data come in (Int64, Str, etc.)?

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

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 generating temperature and humidity data

- 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
- etc.

Sensor cluster: Create value descriptors

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" ]  
}
```

Value descriptors 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

Sensor cluster: Create value descriptors

Update the body and issue the command again for temperature:

Method: POST

URI: `http://<edge ip>: 48080/api/v1/valuedescriptor`

Payload settings: Set Body to "raw" and "JSON"

Payload data:

```
{  
  "name": "humidity",  
  "description": "Ambient temperature in percent",  
  "min": "-50",  
  "max": "100",  
  "type": "Int64",  
  "uomLabel": "temperature",  
  "defaultValue": "0",  
  "formatting": "%s",  
  "labels": [  
    "environment",  
    "temperature" ]  
}
```

Sensor cluster: Upload the device profile

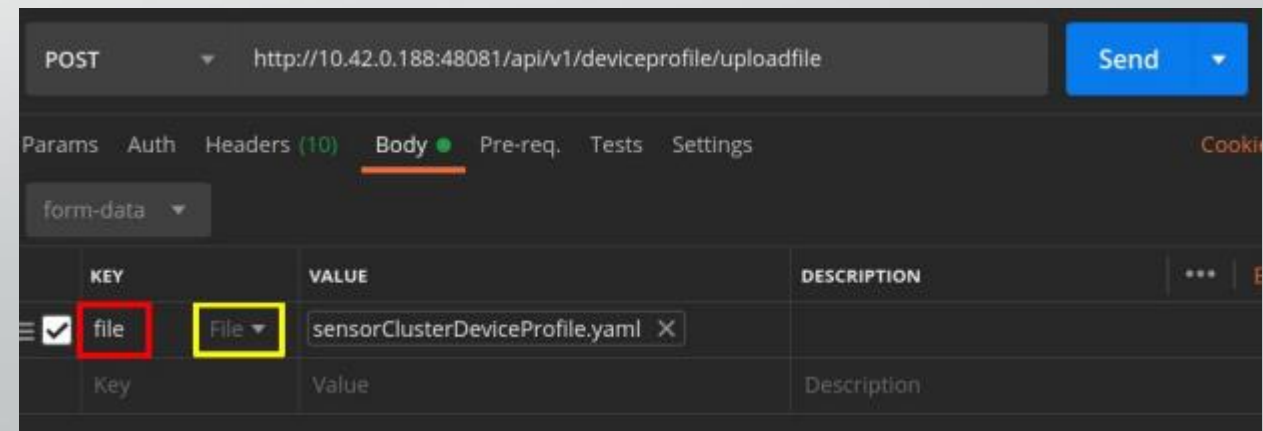
- Get a copy of the device profile from here: https://raw.githubusercontent.com/jonas-werner/EdgeX_Tutorial/master/deviceCreation/sensorClusterDeviceProfile.yaml
- **In Postman use the following settings:**

Method: POST

URI: <http://10.42.0.188:48081/api/v1/deviceprofile/uploadfile>

Payload settings:

- 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



Sensor cluster: Create the device

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"
    }
  },
}
```

The device service "edgex-device-rest" is used since this is a REST device.

```
"labels": [
  "Humidity sensor",
  "Temperature sensor",
  "DHT11"
],
"location": "Tokyo",
"service": {
  "name": "edgex-device-rest"
},
"profile": {
  "name": "SensorCluster"
}
}
```

The profile name "SensorCluster" must match the name in the device profile yaml file uploaded in the previous step

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 is to use a Python script to simulate data values continuously.
- Finally, for those who wish to do so, a Raspberry Pi can be used to pull real values from a DHT humidity / temperature sensor and send these to EdgeX every few seconds.

The event counter

- When data is sent to EdgeX it's registered as an event.
- It's possible to view the current event count with a browser (or cURL or Postman, etc.)
 - <http://:48080/api/v1/event/count>
- The page doesn't refresh by itself, so use F5 or the refresh button in the browser to view the latest event count during these examples

Sending data with Postman

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

Method: POST

URI: `http://<edgex ip>: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)

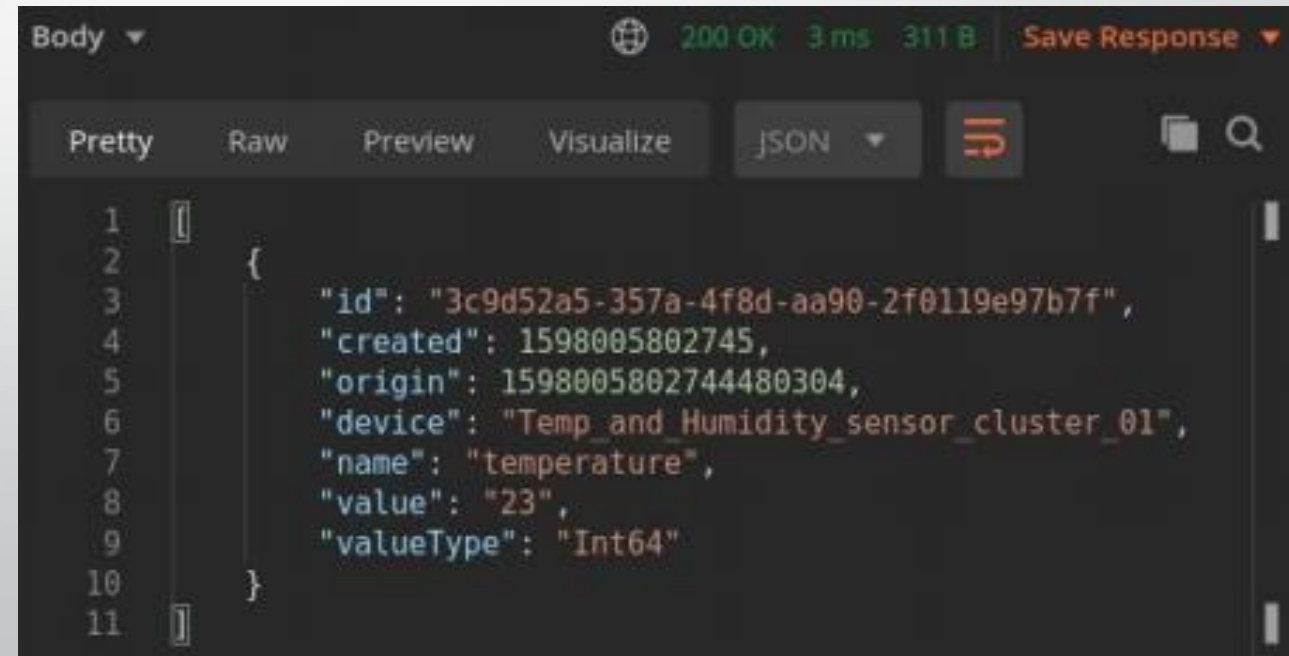
- After submitting the data in Postman, look at the event count in the browser. Has it changed?
- Modify the URI to send a humidity value
- Create another device (sensor_cluster_02 for example) and modify the URI to send data to that device instead

View the data with Postman

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

Method: GET

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



Generate sensor data with Python

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/
```

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
```

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

```
which python3
```

The Python module "requests" need to be installed to run the script. It is advisable to install modules in a separate virtual Python environment

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

Generate sensor data with Python

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

10. To exit the script, use CTRL+C

11. To exit the virtual environment, simply type deactivate

deactivate

Export data stream

- Data export to an external source, like an MQTT topic, AWS or Azure, is generally done using the Application Service.
- This service can be configured by adding options to the docker-compose.yml file.
 - Examples are posted to the EdgeX Foundry documentation page:
<https://docs.edgexfoundry.org/1.2/microservices/application/AppServiceConfigurable/>
- The data can also be exported selectively by using the Rules Engine (Kuiper).
 - In this case an SQL statement can be used to pick up on certain data and export it as desired. Both methods will be shown.

MQTT export using the Application Service

1. On the EdgeX Foundry VM, enter the "geneva" folder
 1. `cd geneva/`
2. Stop EdgeX Foundry (if it is running)
 1. `docker-compose stop`
3. Download the new docker-compose file from GitHub
 1. `wget https://raw.githubusercontent.com/jonas-werner/EdgeX_Tutorial/master/docker-compose_files/docker-compose_step2.yml`
4. Copy the new docker-compose file to "docker-compose.yml"
5. Edit the docker-compose.yml file and enter a unique MQTT topic ID instead of the entry "YOUR-UNIQUE-TOPIC".
 - Anything is fine as long as it's unique and memorable. Avoid spaces and special characters. Don't forget the quotation marks.

```
# Added for MQTT export using app service
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_ADDRESS: broker.hivemq.com
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_PORT: 1883
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_PROTOCOL: tcp
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_TOPIC: YOUR-UNIQUE-TOPIC
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_PARAMETERS_AUTORECONNECT: "true"
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_PARAMETERS_RETAIN: "true"
WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_PARAMETERS_PERSISTONERROR: "false"
# WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_PUBLISHER:
# WRITABLE_PIPELINE_FUNCTIONS_MQTTSEND_ADDRESSABLE_USER:
```

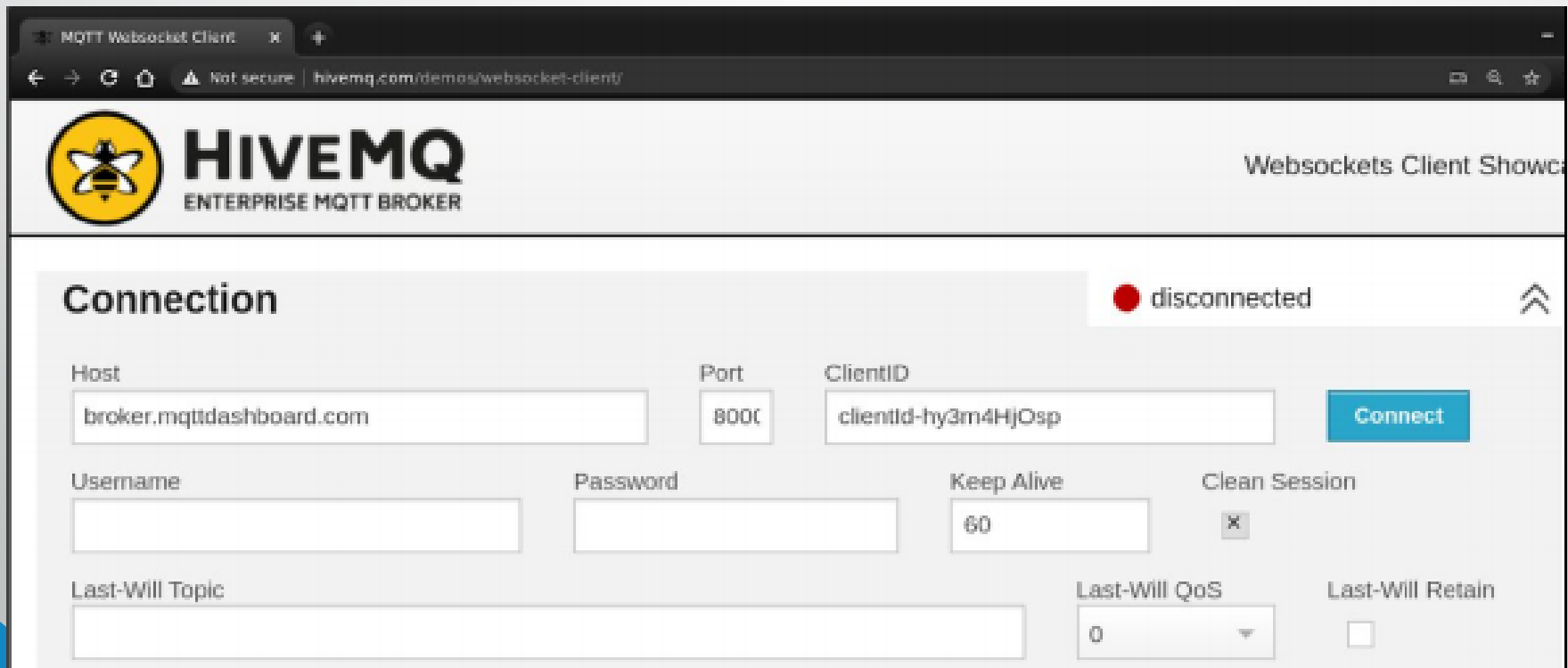
MQTT export using the Application Service

6. Start EdgeX Foundry

docker-compose up -d

7. Open a browser and go to: <http://www.hivemq.com/demos/websocket-client/>

8. Click on Connect (all the default values are fine)



The screenshot shows a web browser window titled "MQTT Websocket Client" at the URL <http://www.hivemq.com/demos/websocket-client/>. The page features the Hivemq logo and the text "ENTERPRISE MQTT BROKER". The main section is titled "Connection" and displays a status indicator "disconnected" with a red dot. Below this, there are input fields for "Host" (broker.mqttdashboard.com), "Port" (8080), and "ClientID" (clientId-hy3m4HjOsp). A "Connect" button is visible. Other fields include "Username", "Password", "Keep Alive" (60), "Clean Session" (checked), "Last-Will Topic", "Last-Will QoS" (0), and "Last-Will Retain" (unchecked).

MQTT export using the Application Service

9. Click on “Add New Topic Subscription” and enter the name of your MQTT topic exactly as it is listed in the docker-compose.yml file.
10. Click “Subscribe”
11. All data sent to EdgeX Foundry will now be processed and sent to this MQTT topic. Generate some data and see it appear in the HiveMQ web console:

The screenshot displays the HiveMQ web console interface, which is divided into three main sections: Publish, Subscriptions, and Messages.

- Publish:** This section contains a form for sending MQTT messages. It includes a "Topic" input field with the value "testtopic/1", a "QoS" dropdown menu set to "0", and an unchecked "Retain" checkbox. A blue "Publish" button is located to the right of these fields. Below them is a "Message" text area.
- Subscriptions:** This section shows the list of active subscriptions. It features a blue button labeled "Add New Topic Subscription". Below this, there is a subscription entry for the topic "edgex-tutorial" with a QoS of 2, indicated by a small 'x' icon.
- Messages:** This section displays a list of received messages. The first message is shown with its timestamp (2020-08-24 12:31:21), topic (edgex-tutorial), and QoS (0). The message payload is a JSON object:

```
{ "id": "480d9be8-16d7-441b-beff-8a0cca52f057", "device": "Temp_and_Humidity_sensor_cluster_01", "origin": "1598239881297907352", "readings": [ { "id": "65664f6c-5dd4-4b33-9b5e-87cd04082d47", "origin": "1598239881297804172", "device": "Temp_and_Humidity_sensor_cluster_01", "name": "humidity", "value": "83", "valueType": "Int64" } ] }
```



Congratulations!

You're now ingesting, processing and exporting data while converting between REST and MQTT protocols.

MQTT export using the Rules Engine

- Data can also be exported more selectively by using Kuiper - the EdgeX rules engine.
 - Kuiper uses the concept of streams.
 - Rules will link with a stream to execute actions based on SQL statements.
- Postman will be used to create a stream and then a rule linking with that stream.
- The rule will be configured to capture all data from the stream and export it using MQTT to a HiveMQ topic.
 - Kuiper runs on port 48075.
 - This can be verified by entering the "geneva" folder and executing:
docker-compose ps | grep 48075

MQTT export using the Rules Engine

1. Creating a Kuiper stream with Postman

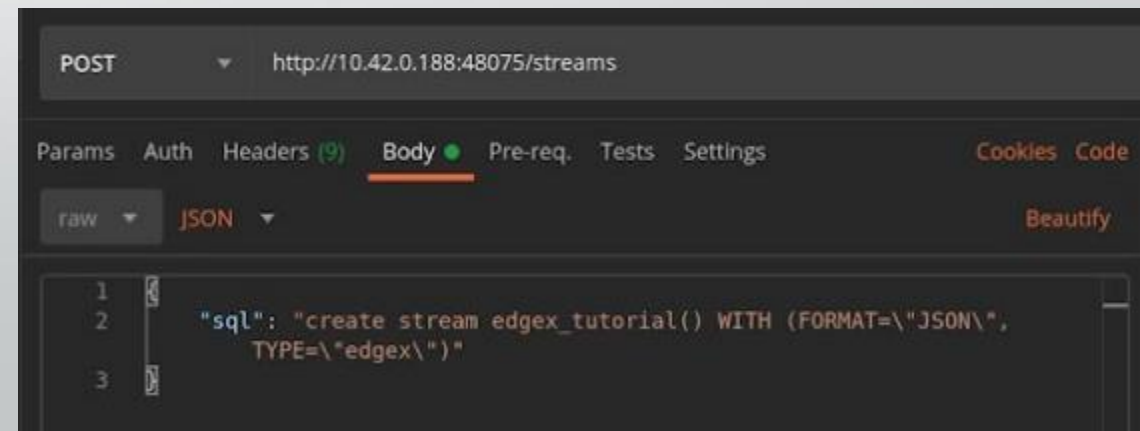
Method: POST

URI: http://:48075/streams

Payload settings: Set Body to "raw" and "JSON"

Payload data:

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



MQTT export using the Rules Engine

2. Creating a Kuiper rule with Postman

Method: POST

URI: `http://<edgex ip>:48075/rules`

Payload settings: Set Body to "raw" and "JSON"

Payload data:

```
{
  "id": "mqtt_export_rule",
  "sql": "SELECT * FROM mqtt_export",
  "actions": [
    {
      "mqtt": {
        "server": "tcp://broker.hivemq.com:1883",
```

```
      "topic": "EdgeXFoundryMQTT_o1",
      "username": "someuser",
      "password": "somepassword",
      "clientId": "someclientid"
    }
  ],
  {
    "log": {}
  }
]
```

MQTT export using the Rules Engine

3. Verifying that HiveMQ is receiving the data

- Visit <http://www.hivemq.com/demos/websocket-client/>
- Click “Connect” (default values are fine)

4. Add the topic subscription and send some data to EdgeX Foundry

The screenshot displays the HiveMQ Websocket Client interface, which is divided into three main sections: Publish, Subscriptions, and Messages.

Publish Section: This section contains a 'Topic' input field, a 'QoS' dropdown menu set to '0', a 'Retain' checkbox, and a 'Publish' button. Below these is a 'Message' input field.

Subscriptions Section: This section features an 'Add New Topic Subscription' button. Below it, a subscription is listed with 'Qos: 0' and the topic 'Unique-and-interest...'. A close button (X) is visible next to the subscription.

Messages Section: This section displays a list of received messages. Each message entry includes a timestamp, the topic, the QoS, and the message payload. The messages shown are:

Timestamp	Topic	QoS	Message
2020-08-24 14:20:58	Unique-and-interesting-topic	Qos: 0	{["humidity":75]}
2020-08-24 14:20:52	Unique-and-interesting-topic	Qos: 0	{["humidity":75]}
2020-08-24 14:20:47	Unique-and-interesting-topic	Qos: 0	{["humidity":75]}

Sending commands

- EdgeX Foundry ability to send commands to devices
- In this section commands will be demonstrated using a test application.
- The application runs in a container and has a web service which can be updated over a REST API.
- This can be used by the app to receive commands from EdgeX Foundry and execute changes to a web interface viewable through a browser.
- Note: The current REST device service doesn't yet support commands. Therefore we'll be using a legacy function of EdgeX to get around this limitation by creating a dummy device service.
- This section is broken up into the following steps:
 - Building and running the test app container
 - Registering the app as a new device
 - Issuing commands via EdgeX
 - Creating a rule to execute a command

Sending commands

Building and running the test app container

1. On the EdgeX Foundry VM, clone the repository for the test app container

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

2. Enter the directory and build the container

```
cd colorChanger/
```

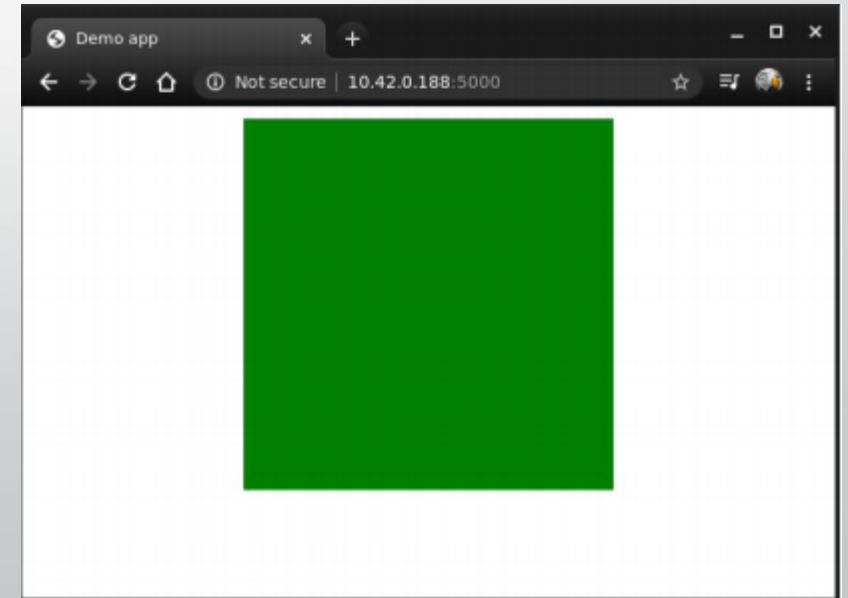
```
docker build -t colorChanger .
```

3. Run the container

```
docker run -d -p 5000:5000 --name colorchanger colorchanger:latest
```

4. Verify that the web interface of the container is accessible by using a web browser:

```
http://<edgex ip>:5000
```



Sending commands

5. Use Postman to test the REST API

Method: PUT

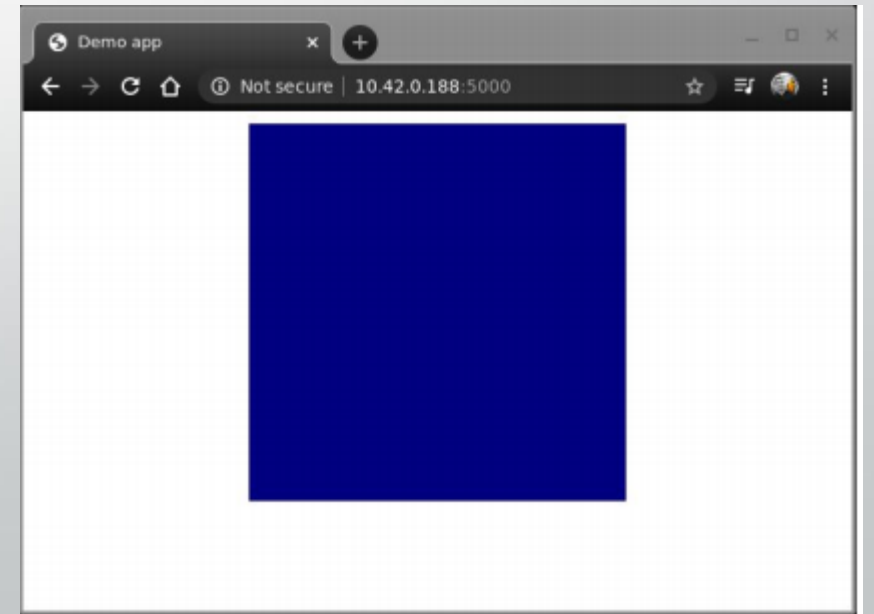
URI: `http://<edgex ip>:5000/api/v1/device/edgexTutorial/changeColor`

Payload settings: Set Body to "raw" and "JSON"

Payload data:

```
{  
  "color": "navy"  
}
```

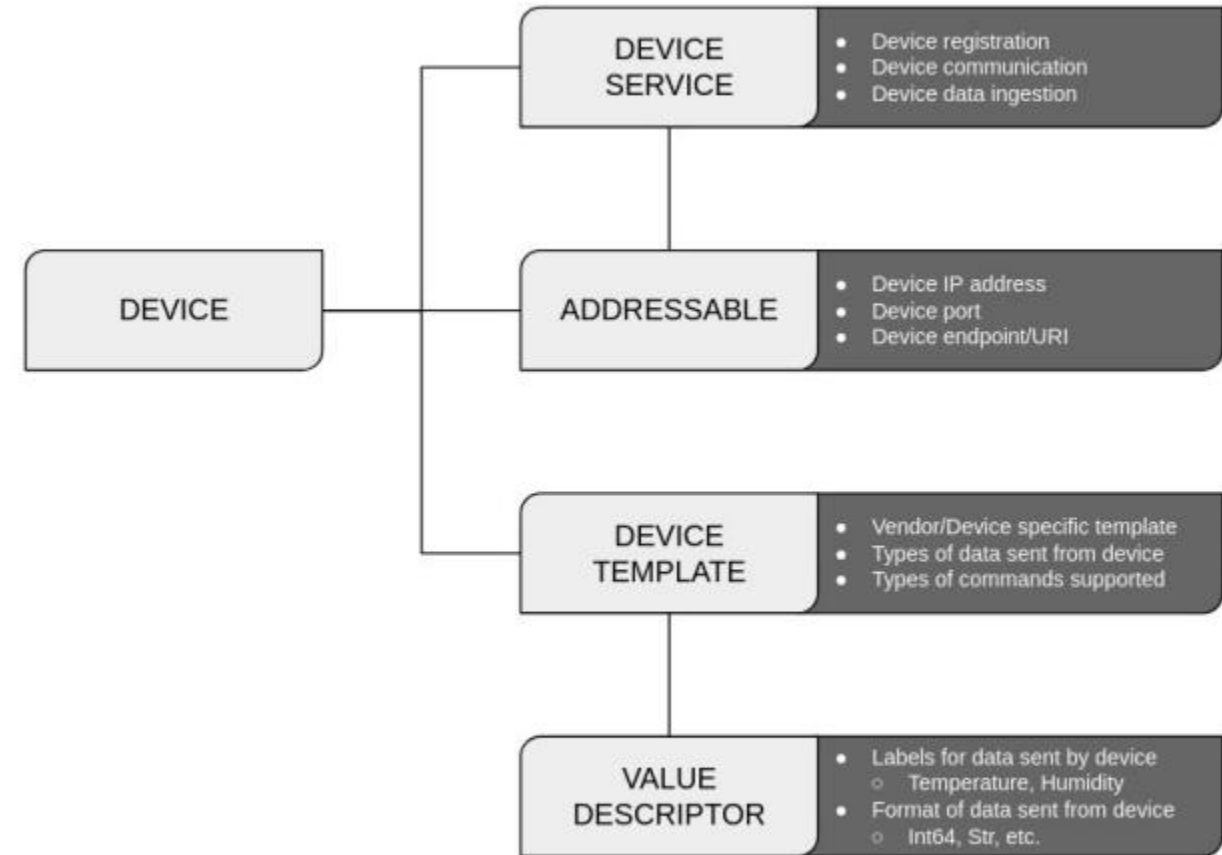
6. The web page should automatically change color



Sending commands

Registering the app as a new device

- EdgeX Foundry needs to know about the test app and how to interact with it.
- To accomplish this a new device template is used together with a Python script to run all commands required to register the app in one go.
 - The script will create the entries and link them together.
 - It includes a few more steps compared with the sensor cluster created previously
 - Since the script does the work it only takes a fraction of a second to complete.



Relation between entities created by the Python script

Sending commands

Registering the app as a new device

Access the EdgeX Foundry VM.

1. If not cloned already, 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 device creation scripts:

```
cd EdgeX_Tutorial/deviceCreation
```

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
```

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

```
which python3
```

The "requests" and "requests-toolbelt" Python modules need to be installed to run the script. Create a virtual Python environment. If a virtual environment has already been created since before, please skip to step 5 and enter the environment.

Sending commands

Registering the app as a new device

7. Install the requests module using pip

pip install requests

8. Install the requests-toolbelt module using pip

pip install requests_toolbelt

9. Run the script

python ./createRESTDevice.py -ip -devip

The "-ip" entry refers to the EdgeX Foundry host IP.
The "-devip" entry refers to the IP of the host running the test app.

Sending commands

Issuing commands via EdgeX

1. View the new TestApp device details using Postman

Method: GET

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

- The device "TestApp" is now visible in the list of registered devices
- Scrolling down reveals a "commands" section which thanks to the Device Profile used has been equipped with both "get" and "put" commands with IDs unique to this device.
- Clicking the "url" for "put" will open a new tab in Postman

```
  "put": {
    "path": "/api/v1/device/{deviceId}/changeColor",
    "responses": [
      {
        "code": "201",
        "description": "set the color"
      },
      {
        "code": "503",
        "description": "service unavailable"
      }
    ]
  },
  "url": "http://edgex-core-command:48082/api/v1/device/01671ea6-b75a-4c7c-8afc-8a660d7dd30c/command/8948b979-cebb-4727-b898-03f76268a136"
```

Sending commands

Issuing commands via EdgeX

2. Click the URL for “put”:

- A new Postman tab is opened.
- Replace “edgex-core-command” with the IP of EdgeX Foundry

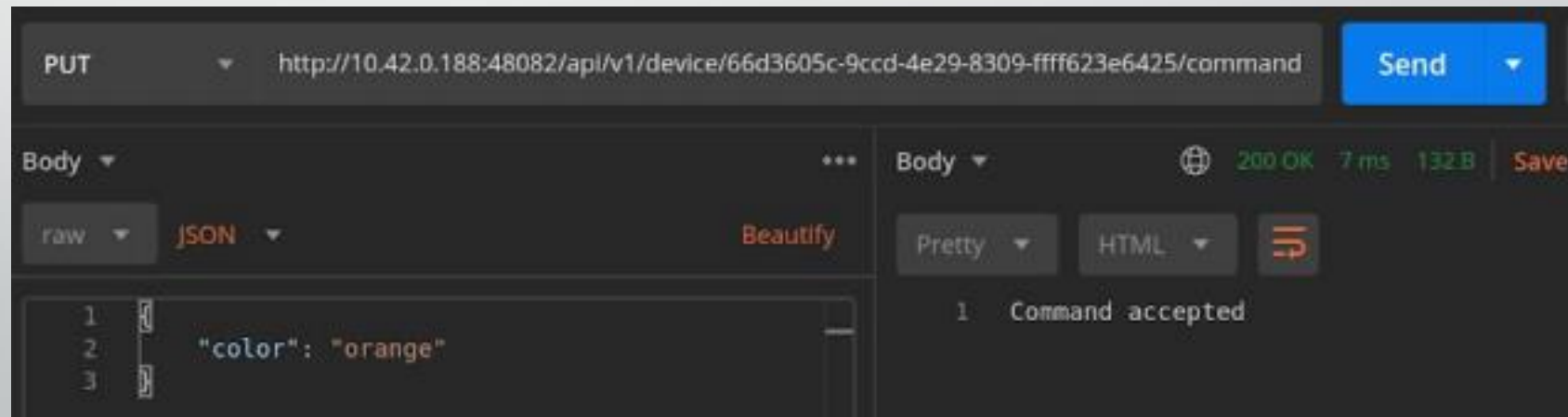
Method: PUT

URI: populated by clicking the “url” link above (unique for each device)

Payload settings: Set Body to “raw” and “JSON”

Payload data:

```
{  
  "color": "orange"  
}
```

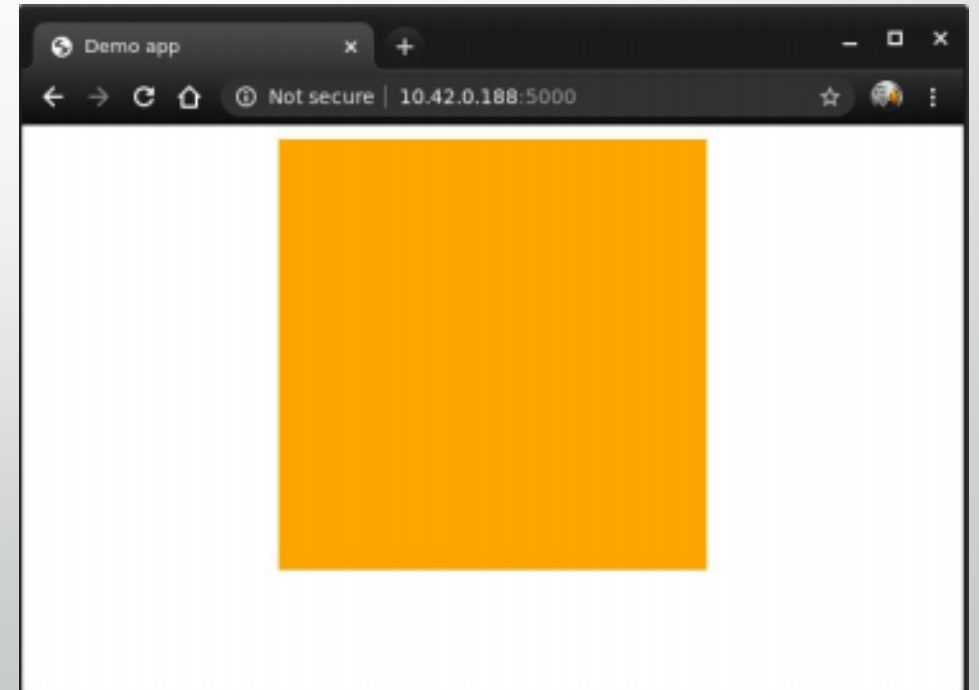


Sending commands

Issuing commands via EdgeX

3. View the TestApp through a browser

`http://<edgex ip>:5000`



Sending commands

Creating a rule to execute commands automatically

1. Create a new Kuiper stream with Postman

Method: POST

URI: `http://<edgex ip>:48075/streams`

Payload settings: Set Body to "raw" and "JSON"

Payload data: `{ "sql": "create stream temp_threshold() WITH (FORMAT=\"JSON\", TYPE=\"edgex\")" }`

2. Create a new Kuiper rule which links to the stream

Method: POST

URI: `http://<edgex ip>:48075/rules`

Payload settings: Set Body to "raw" and "JSON"

Payload data:

```
{  "id": "temp_rule",
  "sql": "SELECT temperature FROM temp_threshold WHERE temperature > 70",
  "actions": [
    { "rest": { "url": "", "method": "put", "retryInterval": -1, "dataTemplate": "{$color\\":\\"red\\"}" },
      "sendSingle": true } },
  { "log":{ } } ] }
```

Sending commands

Creating a rule to execute commands automatically

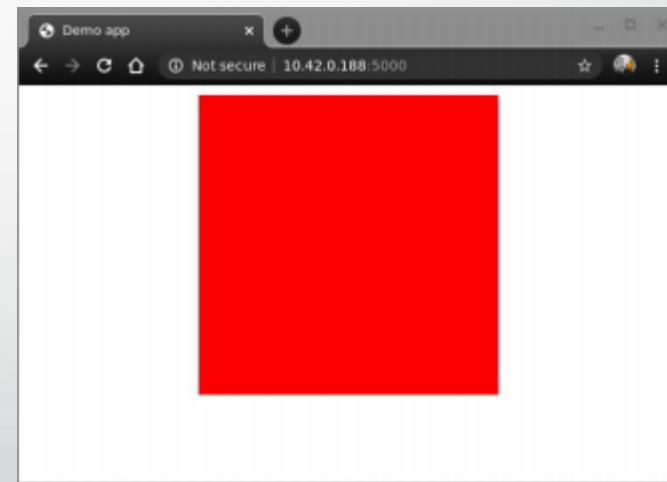
3. Push a temp value over 70 degrees with Postman to trigger the rule

Method: POST

URI: `http://<edgex ip>:49986/api/v1/resource/Temp_and_Humidity_sensor_cluster_o1/temperature`

Payload settings: Set Body to "raw" and "text"

Payload data: 71 (any numeric value over 70 will do)



4. The web app should switch to red

Create another rule to change the color back if the temperature drops below 70 degrees. Flip between the different states by sending different temperature values

Viewing container logs

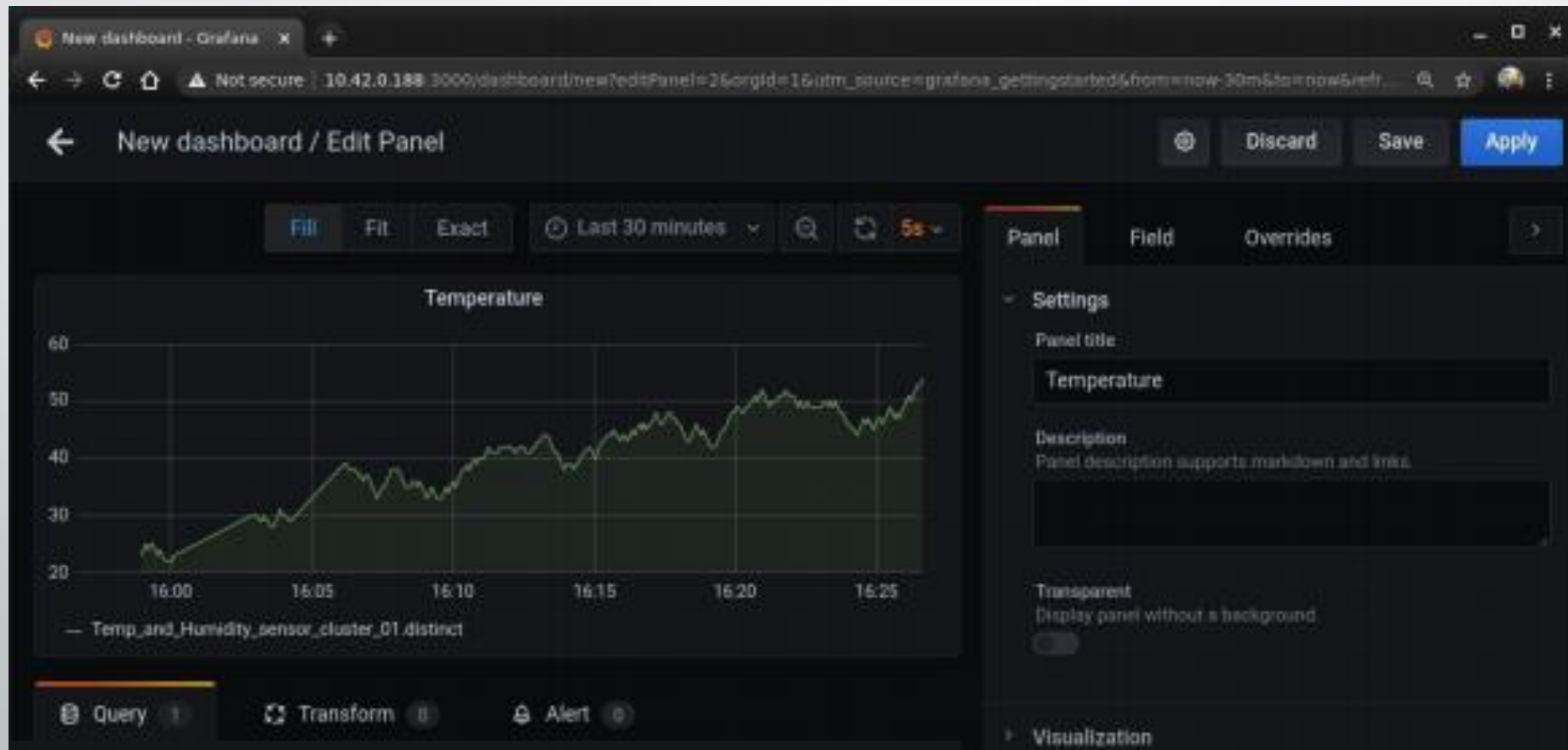
1. Access the EdgeX Foundry VM
2. List up the containers with docker-compose
ps docker-compose ps
3. Note the container names to the left
4. Use the docker logs command to view the log output:
`docker logs <container name>`
For example: `docker logs -f edgex-core-data`
5. To view output continuously (until CTRL+C is pressed), add the “-f” flag
`docker logs -f <container name>`


Redirecting EdgeX to the local MQTT broker

1. Stop EdgeX Foundry if it's running
docker-compose down
2. Enter the "geneva" folder and edit the "docker-compose.yml" file
 1. Change the broker address from: "broker.hivemq.com" to the IP address of the host running the Mosquitto broker.
 2. In this case it's the IP of the host VM running EdgeX Foundry.
3. Start EdgeX Foundry again
docker-compose up -d

Any messages sent to EdgeX Foundry will now be forwarded to the local MQTT broker, captured by the messenger app and entered into the InfluxDB database

Adding Grafana





???