Eclipse Ditto

Building with Apache Maven

In order to build Ditto with Maven, you'll need:

- JDK 8 >= 1.8.0_92 (due to a bug in older versions of the JDK you'll get a compile error),
- Apache Maven 3.x installed,
- a running Docker daemon (at least version 18.06 CE).
- Docker Compose installed (at least version 1.22)

Install Apache Maven

```
$ sudo apt update

$ sudo apt install maven

$ mvn -version
```

Output:

```
Apache Maven 3.5.2

Maven home: /usr/share/maven

Java version: 10.0.2, vendor: Oracle Corporation

Java home: /usr/lib/jvm/java-11-openjdk-amd64

Default locale: en_US, platform encoding: ISO-8859-1

OS name: "linux", version: "4.15.0-36-generic", arch: "amd64", family: "unix"
```

Install Docker

```
$ sudo apt update

$ sudo apt install apt-transport-https ca-certificates curl gnupg-agent
software-properties-common

$ 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 $(lsb_release -cs) stable"
```

Installing Docker CE

```
$ sudo apt update
```

```
$ sudo apt install docker-ce
```

To install a specific version, first list the available versions in the Docker repository:

```
apt list -a docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ sudo systemctl status docker
```

Executing the Docker Command Without Sudo

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

Working with Docker Images

To check whether you can access and download images from Docker Hub, type:

```
$ docker run hello-world
```

Install docker-compose

```
$ which docker-compose
```

```
$ sudo rm /usr/local/bin/docker-compose
```

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.25.0
```

```
/docker-compose-`uname -s`-`uname -m` -o /usr/bin/docker-compose
```

```
$ sudo chmod +x /usr/bin/docker-compose
```

```
$ docker-compose --version
```

Ditto Build

Clone eclipse-ditto to the working directory

```
$ git clone https://github.com/eclipse/ditto.git

$ cd ditto
```

if you have the docker daemon running on your machine and you are running on Unix

```
$ mvn clean install -Pdocker-build-image -Ddocker.daemon.url=unix:///var/

run/docker.sock

$ cd deployment/docker/
```

the "dev.env" file contains the SNAPSHOT number of Ditto, copy it to ".env" so that docker compose uses it:

```
$ cp dev.env .env

$ docker-compose up -d
```

Possible Errors:

1. `Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:3.0.0-M1:test (default-test) on project ditto-services-connectivity-messaging`

Solution:

Try changing the maven-surefire-plugin version to the stable version (3.0.0-M4) in pom.xml [ditto root folder]

2. `Failed to execute goal io.fabric8:docker-maven-plugin:0.26.0:remove (docker-build-image) on project ditto-services: Cannot create docker access object : Cannot extract API version from server unix://127.0.0.1:1 : Permission denied -> [Help 1]`

The Docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user root and other users can only access it using sudo. The Docker daemon always runs as the root user.

If you don't want to preface the docker command with sudo, create a Unix group called docker and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the docker group.

This is not a Docker related issue, but more a linux socket permission problem (very common issue between php-fpm and nginx).

Docker is a classic HTTP client/server application, the client will speak via a socket or IP to the server. The socket is a linux file, with permission, it belongs to an user and a group.

- Create the docker group.

```
$ sudo groupadd docker
```

- Add your user to the docker group.

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

- Log out and log back in so that your group membership is re-evaluated.

```
$ sudo gpasswd -a ${USER} docker # not sure about this command
```

- Restart the docker daemon

```
$ sudo service docker restart
```

- Run the following command to find out what groups you belong to

```
$ groups $USER
```

```
avatar@avatar-X556UQK:~$ groups $USER
avatar : avatar adm cdrom sudo dip plugdev lpadmin sambashare docker
avatar@avatar-X556UQK:~$
```

- Run the following command to check the socket file permission

```
$ ls -l /var/run/docker.sock
```

```
avatar@avatar-X556UQK:~$ ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Nov 23 16:31 /var/run/docker.sock
avatar@avatar-X556UQK:~$
```

For more info:
https://docs.docker.com/install/linux/linux-postinstall/

Setup Grove Control Demo

Setting up your machine

- Clone the ditto examples project into your machine

```
$ git clone https://github.com/eclipse/ditto-examples.git

$ cd ditto-examples/grove-ctrl
```

- **Start Eclipse Ditto.**
- When you have the Eclipse Ditto services up and running, we can move on and create the Thing that we will use for playing around with our Raspberry. The structure of the Thing can be found inside resources/thing-model.json. To be able to use it, we will use the REST API of Eclipse Ditto to create the Thing.

```
$ curl -u ditto:ditto -X PUT -d '@resources/thing-model.json'
'http://localhost:8080/api/1/things/org.eclipse.ditto.example:raspberry'
```

- The above command will create the Thing in our running Eclipse Ditto instance. The default user/password is ditto:ditto to create our Thing.
- Create a new user “raspberry”.
- We will use the password “raspberry” for our user “raspberry”.
- Create a hashed password using openssl tool

```
$ openssl passwd -quiet

 Password: <enter password>

 Verifying - Password: <enter password>
```

- The above command generates a hashcode. Append the printed hash in the **nginx.httpasswd** file (inside deployment/docker folder)placing the username who shall receive this password in front like this:

  raspberry:<hash code>

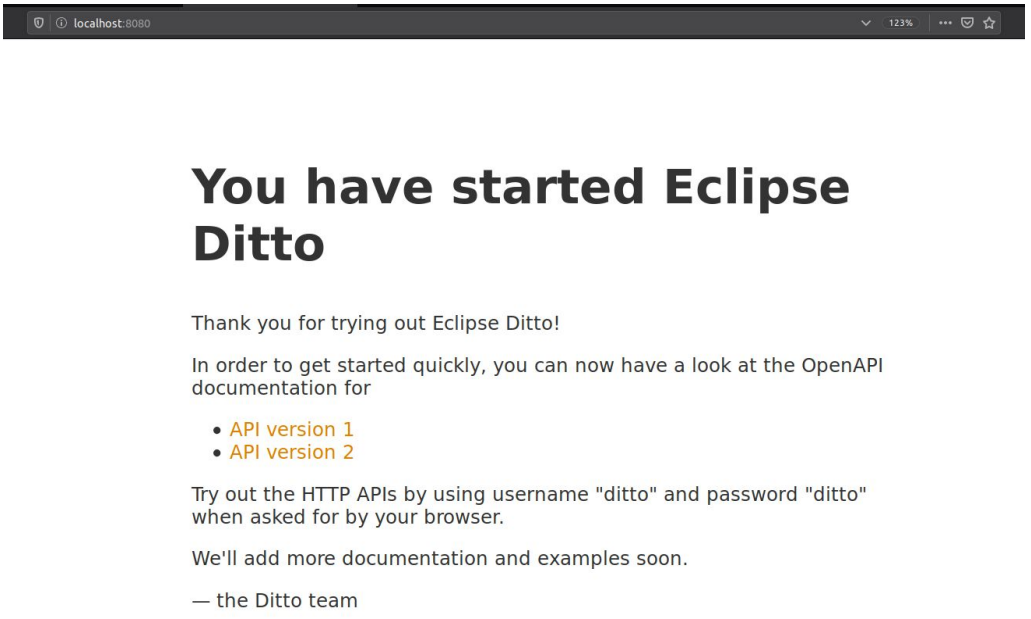
- The user should automatically be available after changing the contents of the nginx.httpasswd file. We can check it by using curl to retrieve the Thing with the raspberry user:

```
$ curl -u raspberry:raspberry -X GET
'http://localhost:8080/api/1/things/org.eclipse.ditto.example:raspberry'
```

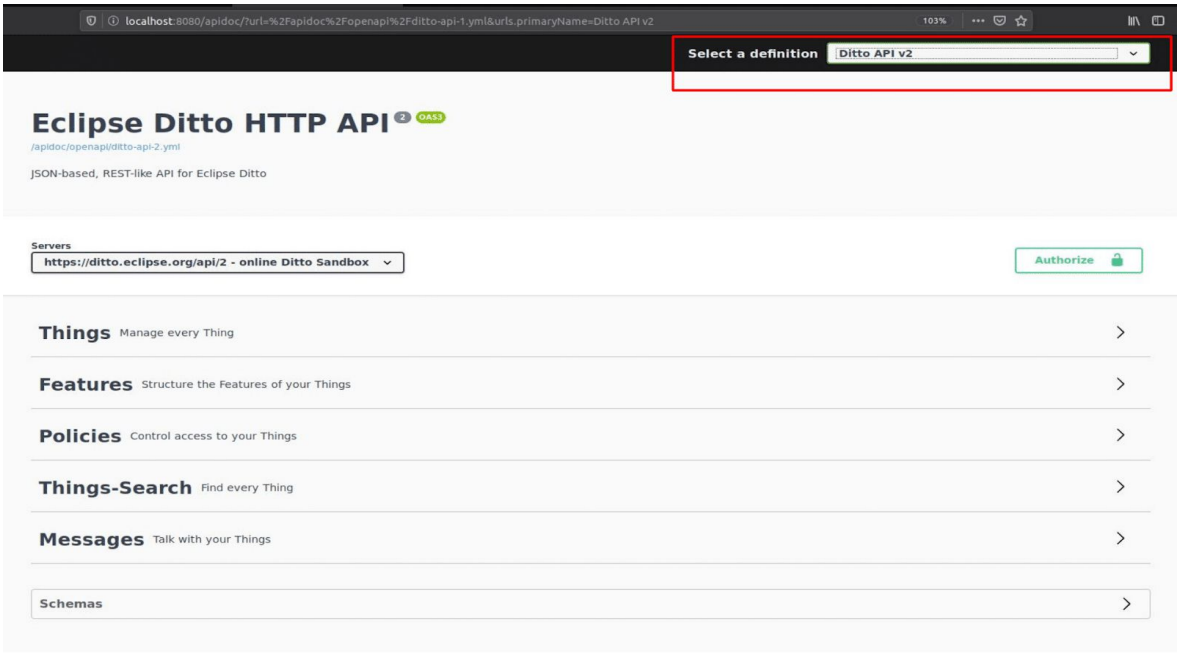
- The above command should return JSON equal to the thing-model. If it returns a 403 error, you should restart your Eclipse Ditto containers (`docker-compose restart`).

Test your ditto instance

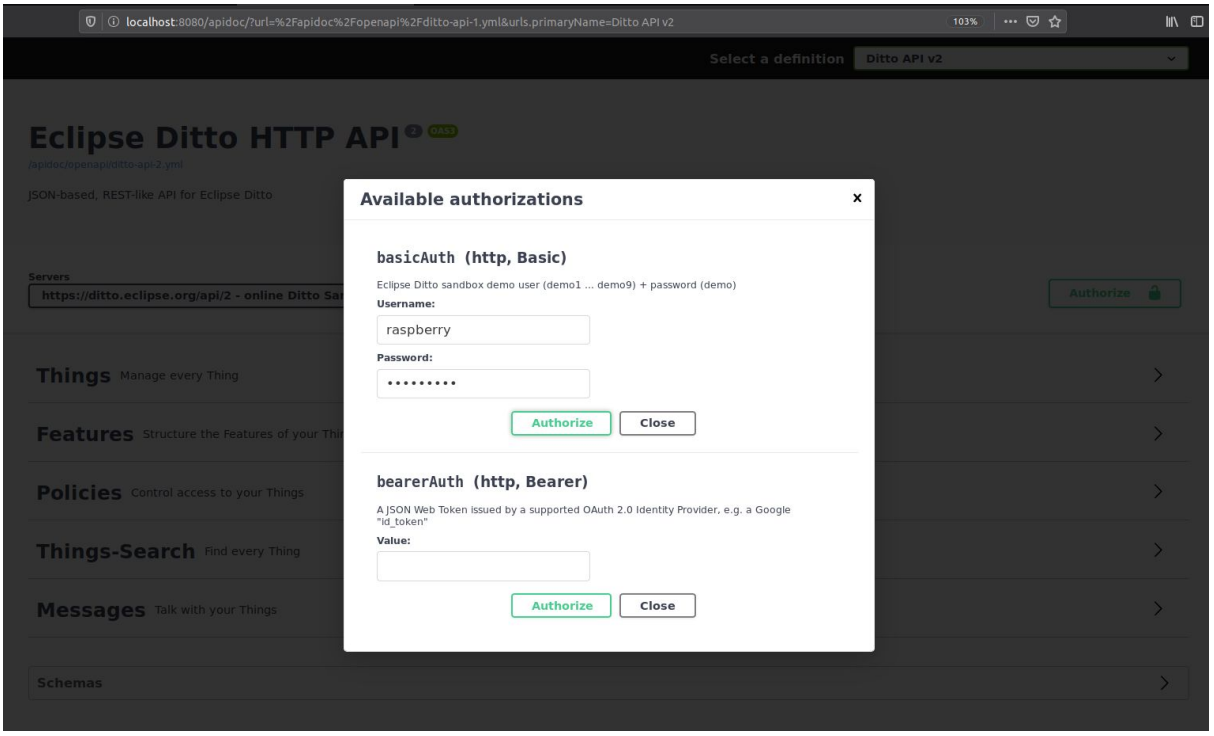
- Go to **http://localhost:8080/**



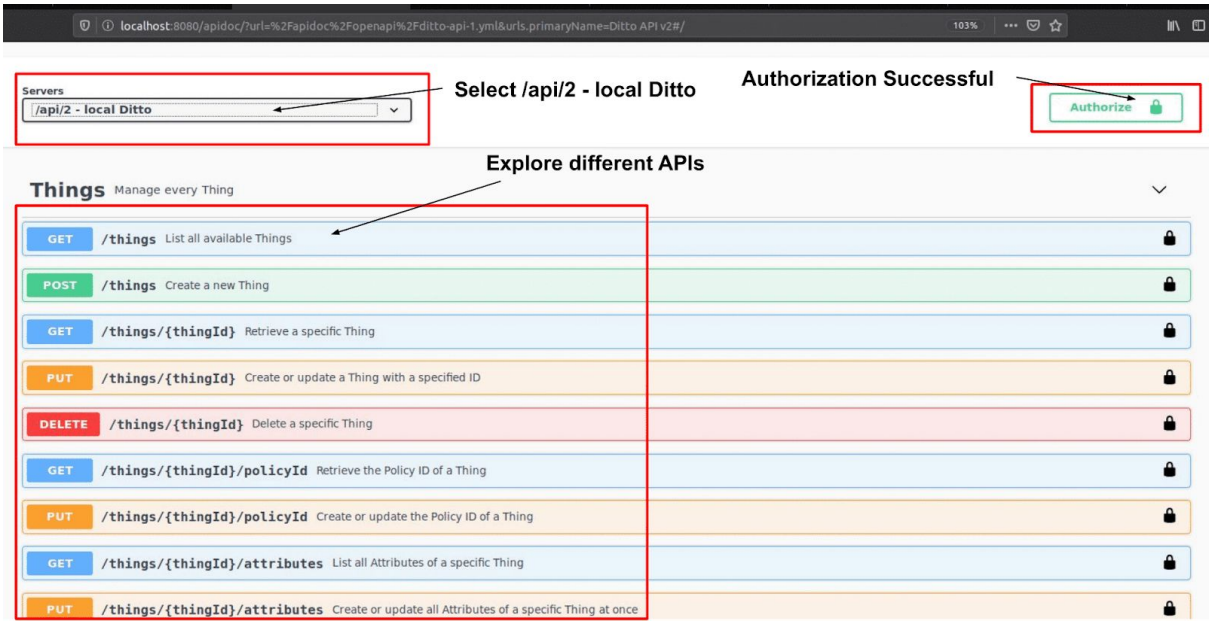
- Click on API version 2 and select Ditto API v2 definition



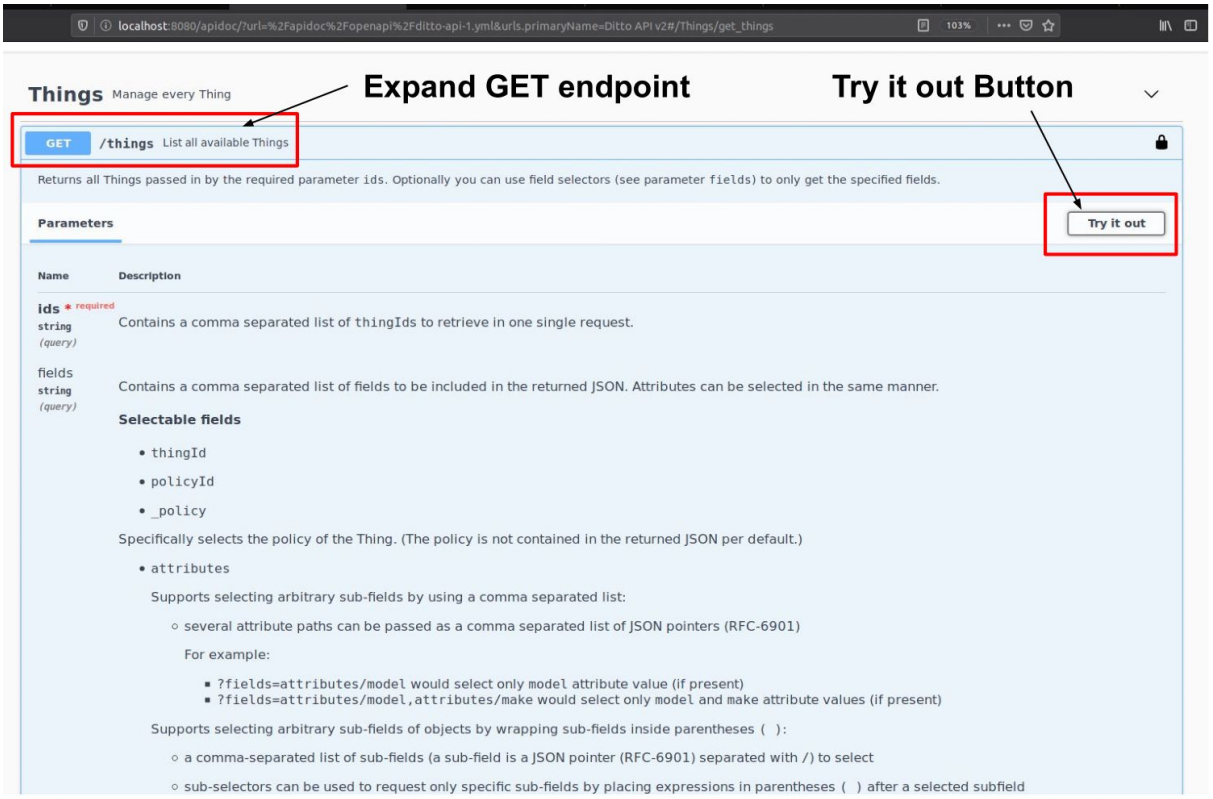
- Click on Authorize button >>> enter username and password (raspberry:raspberry) >>> click on Authorize button >>> close the dialog.



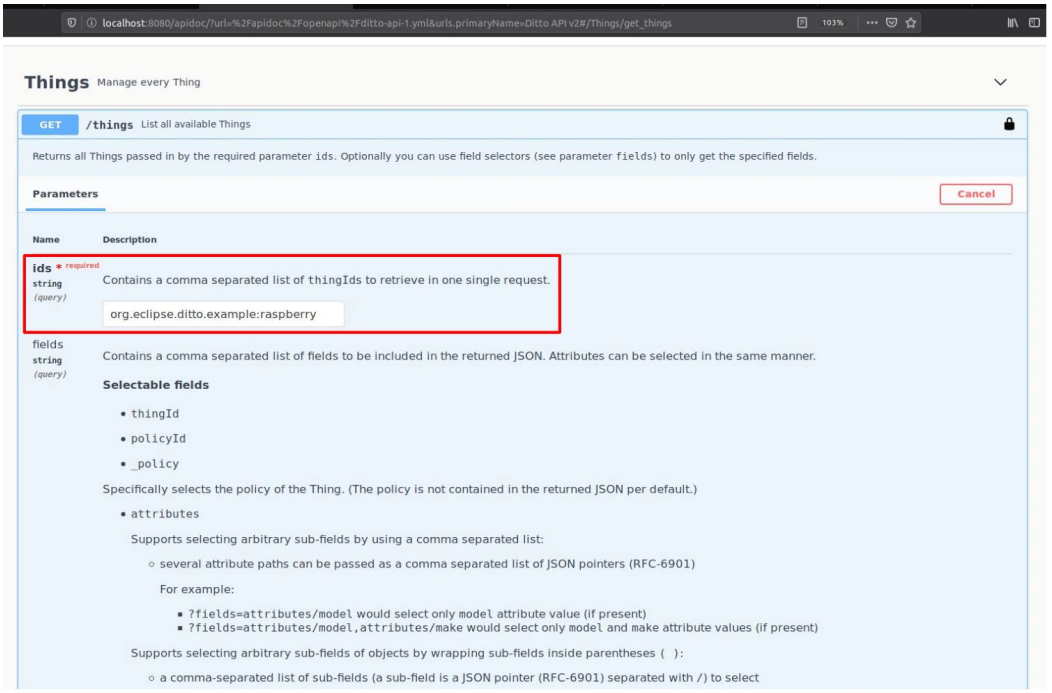
- Explore different APIs



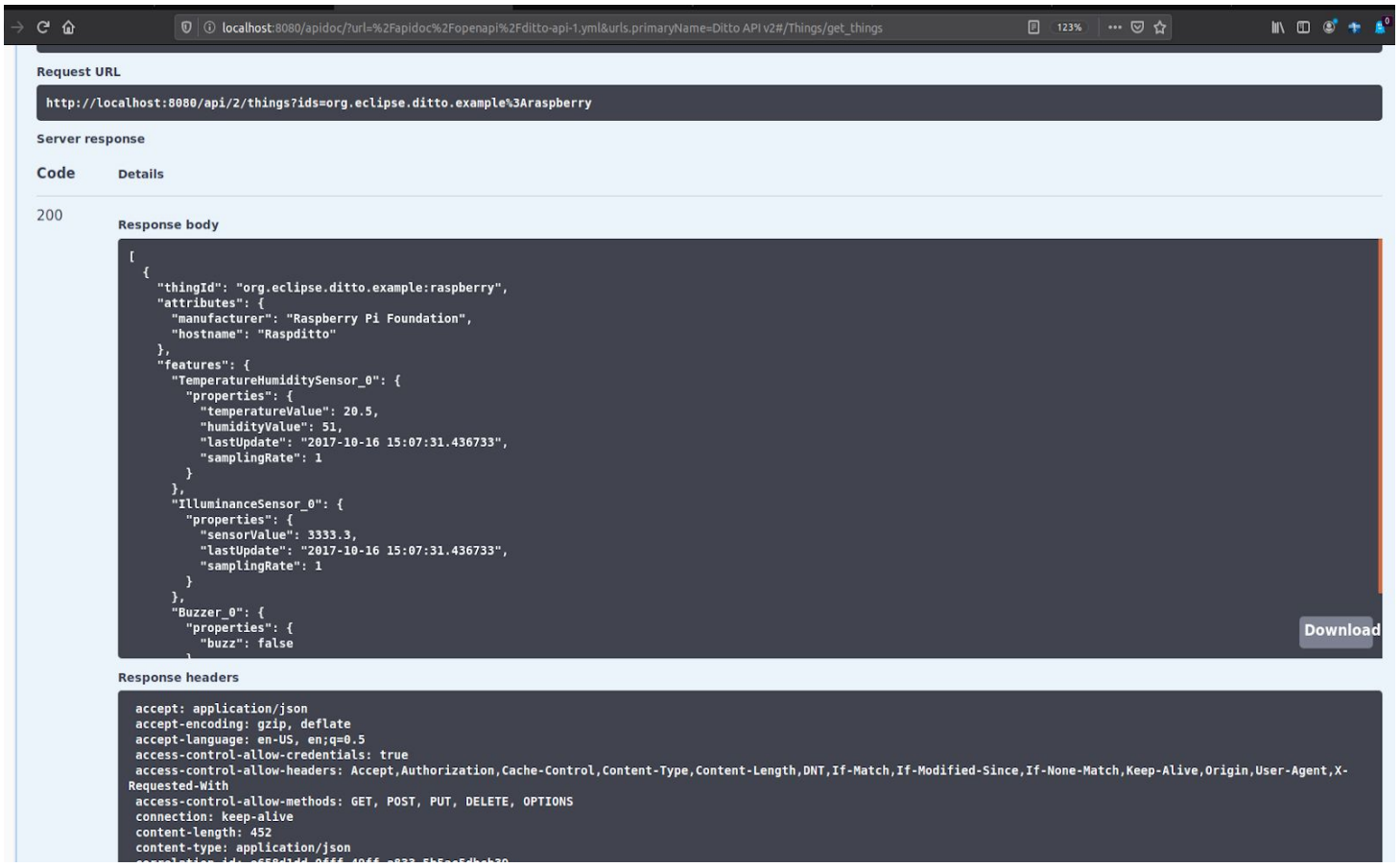
- Let us try to get list of all available things. Now let's make a request:
    1. Expand the GET endpoint.
    2. Click Try it out.



3. After you click Try it out button, Ids in the Request Body field becomes editable.
4. In the Ids field, enter thingId (org.eclipse.ditto.example:raspberry).



5. Click Execute. Swagger UI submits the request and shows the curl that was submitted. The Responses section shows the response.



Setting up Raspberry Pi

- Clone the ditto examples project

```
$ git clone https://github.com/eclipse/ditto-examples.git

$ cd ditto-examples/grove-ctrl/python
```

- This folder contains several python files that are used to represent the Raspberry as our Thing: *grove_temp_sensor.py, grove_buzzer.py, grove_light_sensor.py, raspberry_thing.py, grovepi_mock.py, ditto_grove_demo.py.*
- Set the correct address and port of the running Eclipse Ditto instance in ditto_grove_demo.py

  ```
  DITTO_IP = "192.168.1.100"

  DITTO_PORT = "8080"
  ```

- Make sure the Raspberry Pi and your machine are part of the same network and the port DITTO_PORT of your machine is available to other machines on the same network. Then we can start the python script:

  ```
  $ python ditto_grove_demo.py
  ```

Playing around with the WebUI

The WebUI uses NPM for dependency management. Therefore you have to run npm install from the webapp root (/webapp) to fetch the WebUI dependencies before you start.

```
$ cd ditto-examples/grove-ctrl/webapp

$ npm install
```

To monitor the sensor values and interact with the Raspberry Pi, view the WebUI in your Browser. Simply open /webapp/index.html.

Testing the script without a GrovePi+

You can run the scripts without having the GrovePi+ connected to your Raspberry Pi. Though your Ditto instance has to be up and running.

Just replace the import from “**import grovepi**” to “**import grovepi_mock as grovepi**” inside grove_buzzer.py, grove_light_sensor.py and grove_temp_sensor.py.

```
# To 'mock' the calls to grovepi script, change ‘import grovepi’ line to
'import grovepi_mock as grovepi'

import grovepi_mock as grovepi
```