UNIVERSITY OF CAGLIARI

DIEE - Department of Electrical and Electronic Engineering

HTTP Server

Claudio Marche
claudio.marche@outlook.it

MCLab

Summary

- HTTP protocol
- Heroku and Flask
- Web Server
  - Turn ON/OFF a LED

MCLab

HTTP Lifecycle

(1) User issues URL from a browser
http://host:port/path/file

(5) Browser formats the response and displays

Client (Browser)

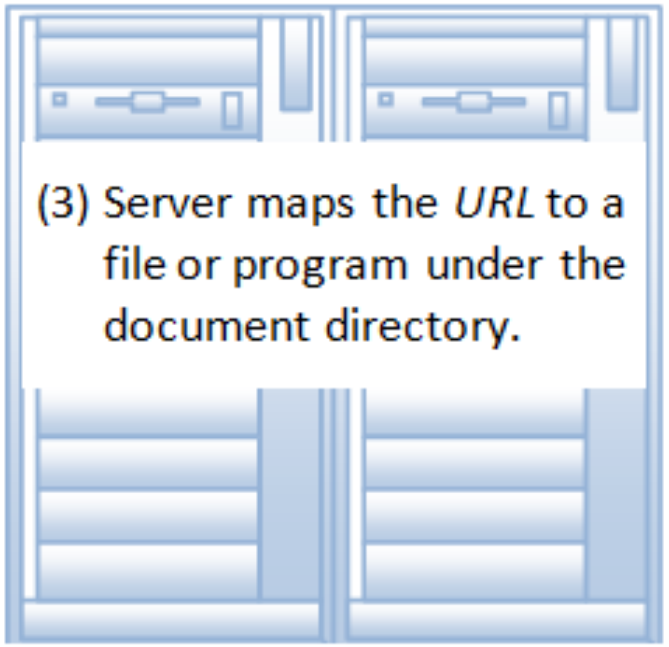
(2) Browser sends a request message

```
GET URL HTTP/1.1
Host: host:port
.................
.................
```

(4) Server returns a response message

```
HTTP/1.1 200 OK
.................
.................
.................
```

HTTP (Over TCP/IP)



Server (@ host:port)

MCLab

HTTP Request

1. **Starts with the Request Line**, that contains
   - The request method (GET, POST, PUT, …)
   - The URL
   - The HTTP version protocol
2. **A list of Request Headers**:
   - *requestHeader1*: *value1*
   - *requestHeader2*: *value2*
3. **Ends with an empty line**

HTTP Request - Example

GET /led/off **HTTP/1.1**
Host: 192.168.1.114
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://192.168.1.114/led/on
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7

MCLab

HTTP Response

1. **Starts with the Status Line**, that contains
   - The HTTP version protocol (HTTP/1.1)
   - The HTTP status code (200, 404, …)
   - The status code message (OK, NOT FOUND, …)
2. **A list of Response Headers**:
   - *responseHeader1*: *value1*
3. **An empty line**
4. **The Response Body** (the HTML document)
5. **Ends with an empty line**

HTTP Response - Example

HTTP/1.1 200 OK
Content-type: text/html
Connection: close

```
<!DOCTYPE html>
<html>
  <body>
    <h1>ESP8266 Web Server</h1>
    <p>LED State ON </p>
    <p><a href="/led/off">OFF</a></p>
  </body>
</html>
```

MCLab

HTML document

1. **It's a markup language**
   - Start markup <markupCode>
   - End markup </markupCode>
2. **Start with** <!DOCTYPE html>
3. **All the document is contained by** <html> </html>
4. **The** <head> </head> **defines the header** and contains information about the document (title, libraries,…)
5. **The** <body> </body> **contains the document** to be displayed

HTML document - Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>ESP8266 Web Server</title>
  </head>
  <body>
    <h1>ESP8266 Web Server</h1>
    <p>LED State ON </p>
    <p><a href="/led/off">OFF</a></p>
  </body>
</html>
```

MCLab

Query string

A query string is the part of a URL containing data that does not fit conveniently into a hierarchical path structure.

- Start with "**?**"
- A field and the respectively value is separated by "**=**"
- Every field-value pair is separated by "**&**"

Example:

www.foo.com/myPage?*field1*=*value1*&*field2*=*value2*

Java Script Object Notation (JSON)

JSON is an open–standard file format that uses human–readable text to transmit data objects consisting of key–value pairs in the format ***"key":value***

Every key-value object is separated by a comma ","

All the JSON data is contained inside a curly bracket pair **{}**

The official MIME type for JSON text is "application/json"

> HTTP Response Header**: Content-type: application/json**

MCLab

JSON Data types (1)

- **Number:** numbers can be signed / with fractional part
  - `"age": 41`
- **String:** strings are delimited with double-quotation marks and can be zero length
  - `"name": "Steve"`
- **Boolean:** booleans can be **`true`** or **`false`**
  - `"isAlive": true`
- **Null element:** an empty value, defined by **`null`** word
  - `"car": null`

JSON Data types (1)

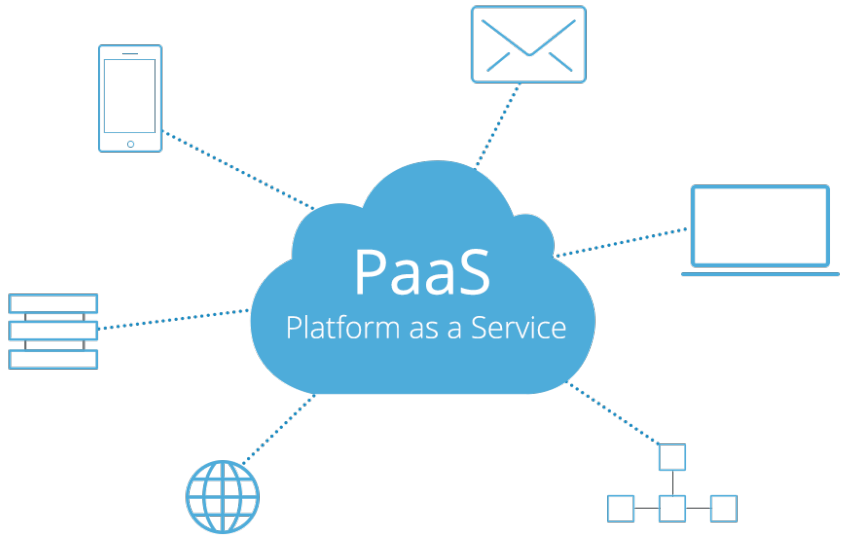
- **Arrays:** Arrays use square bracket notation and elements are comma-separated
  - ```
    "phoneNumbers": [
        "333 26189172",
        "070 389172"
    ]
    ```
    ```

MCLab

JSON Data types (2)

- **Object:** Objects are delimited with curly brackets and use commas to separate each pair
  - ```
    "address": {
        "streetAddress": "Via Roma 2",
        "city": "Cagliari",
        "state": "Italy",
        "postalCode": "09121"
    }
    ```
    ```

MCLab

Heroku

Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage, and scale modern apps. The platform is flexible and easy to use, offering developers a simplest path to getting their apps to market.

HEROKU



Flask

Flask is a web framework.

Flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Web Server (1)

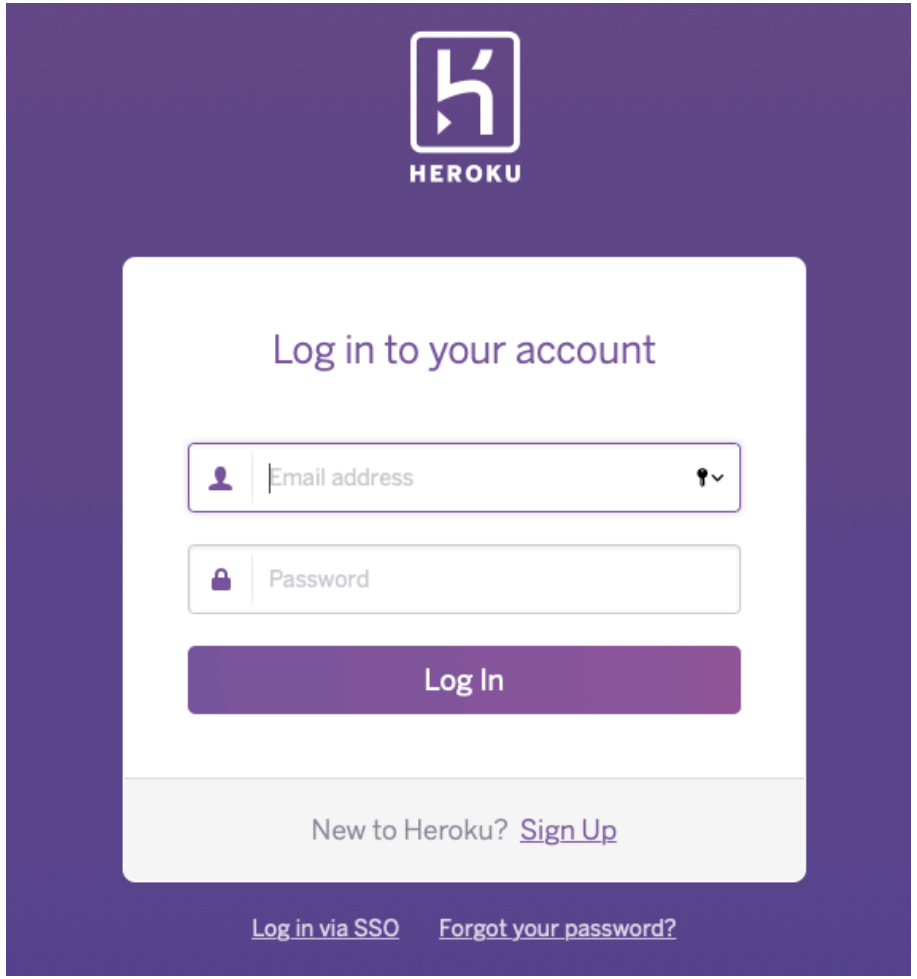
Getting Started:

- Install Python <= 3.7
- Install Pycharm Professional (free student licence)
- Install the Heroku Command Line Interface (CLI)

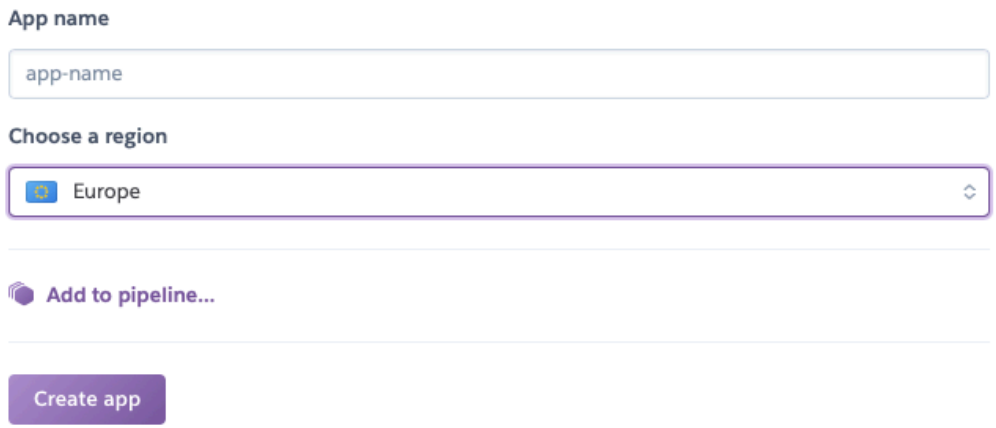
  https://devcenter.heroku.com/articles/heroku-cli

  Heroku CLI requires Git

Web Server (2)

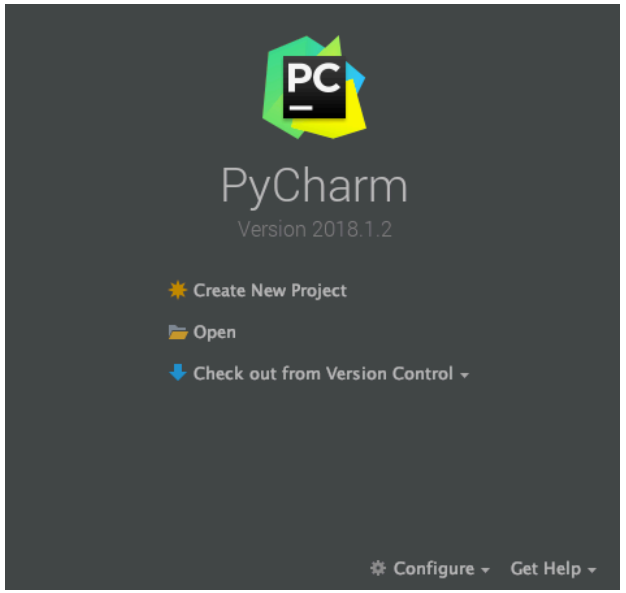


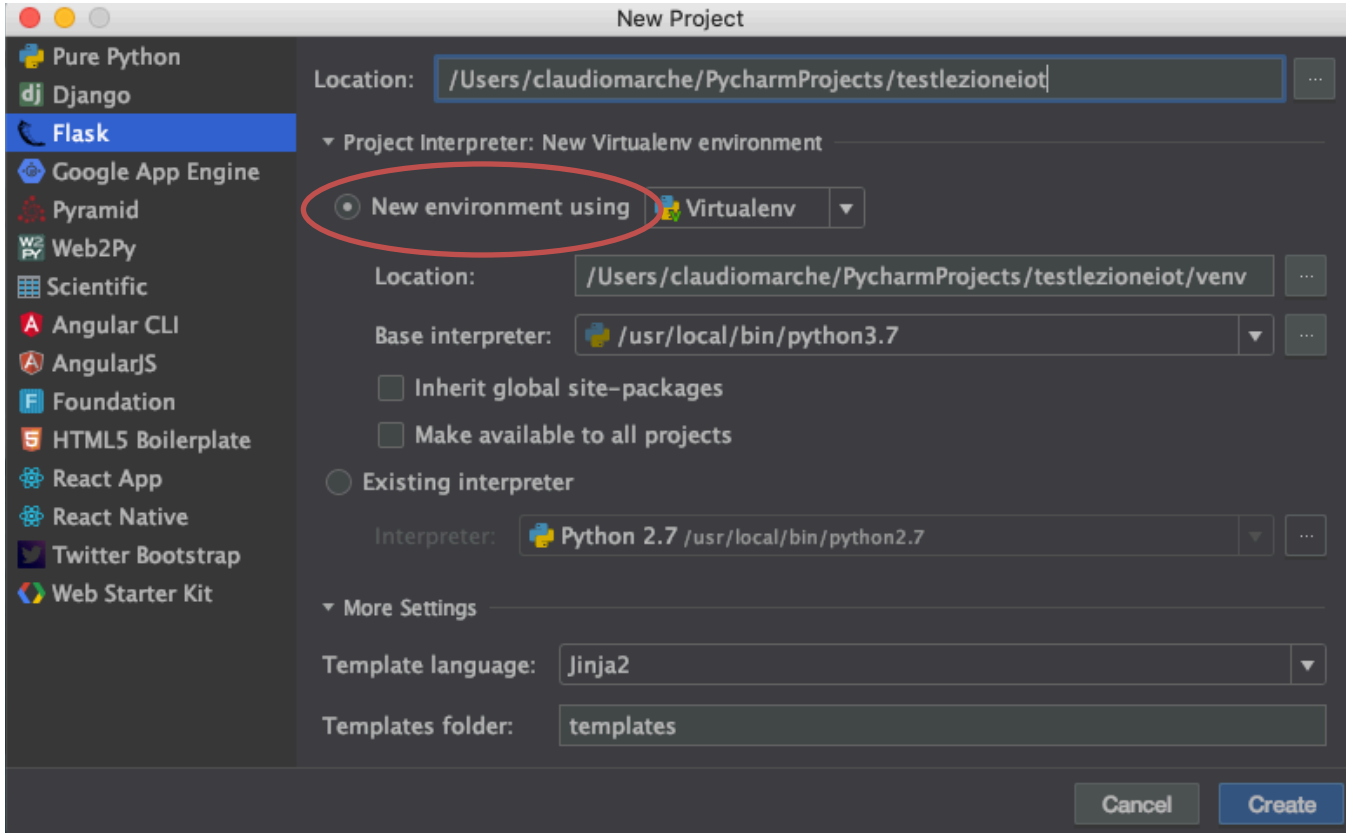
- Sign In or Sign Up
- New - > Create New App



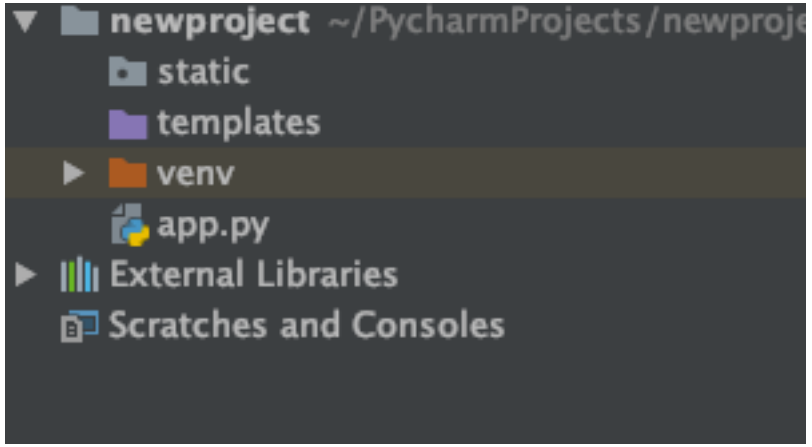
Web Server (3)

Creating a simple Flask Project





Web Server (5)

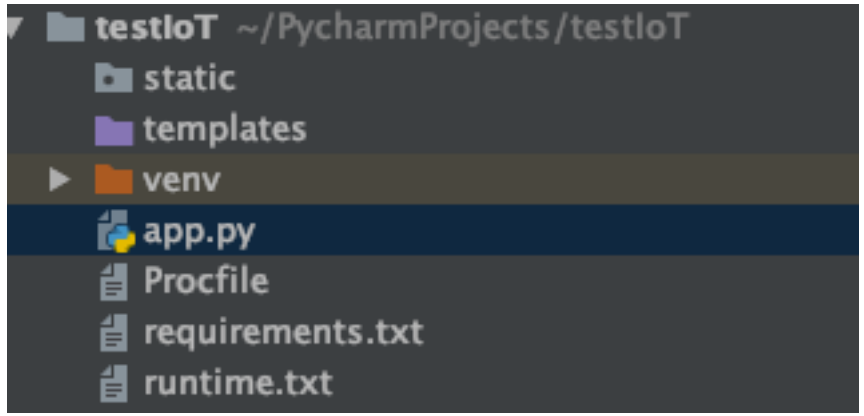


- Create file «runtime.txt» with your python version (es. python-3.7.0)
- Create file «Procfile» with:
  web: gunicorn app:app

Web Server (5)

- Open Terminal (View -> Tool Windows -> Terminal) and create file «requirements.txt»

```
pip install gunicorn
pip freeze > requirements.txt
```



Web Server (6)

- Deploy your application

(in Pycharm Terminal)

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/
$ git init
$ heroku git:remote -a testlezioneiot
```

App name

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

- Open app -> your_app_name.herokuapp.com

Adding functionalities (1)

Send data to nodemcu

- app.py

Remember

Install and import Python Library:

- paho-mqtt

```
broker = "test.mosquitto.org"
port = 1883
```

```
@app.route('/turn_on')
def turn_on():
    client1 = paho.Client("Test")    # unique
    client1.connect(broker, port)
    ret = client1.publish("mclab", "1")
    return 'Success!'


@app.route('/turn_off')
def turn_off():
    client1 = paho.Client("Test")    # unique
    client1.connect(broker, port)
    ret = client1.publish("mclab", "0")
    return 'Success!'
```

Adding functionalities (2)

NodeMCU Code

- Install PubSubClient Library in Arduino
  https://github.com/knolleary/pubsubclient

- Copy the library in Arduino -> libraries (folder)

Adding functionalities (3)

- Open and run mqtt_esp8266 example
  - with topic: mclab (in reconnect function)
  - with loop function:

```
void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

- Serial Monitor

```
WiFi connected
IP address:
192.168.2.34
Attempting MQTT connection...Message arrived [mclab] 0
Message arrived [mclab] 1
Message arrived [mclab] 0
Message arrived [mclab] 1
```