Lab 03 – Cloud web server

Develop a Cloud server

Pre-requisits:

- *Gateway shall be operational (see Lab02)*
- *Sensor device shall be operational (see Lab01)*

In this Lab you will learn.

- how to make server logic
- how to develop a web application and client application

Web Development

In this lab we will use Visual Studio Code (VSCode) as a text editor but you can use any other text editor or IDE software. Donwload it from https://code.visualstudio.com/#alt-downloads and install it.

When installation done. Open your project folder from VSCodeon your project folder.

Web Application

Introduction

Nowadays, static HTML web pages are outdated and HTML5 is ruling the world of web sites. The HTML5 code is always executed on the client side to create more dynamic page content and support various type of web browsers, event touch screen.

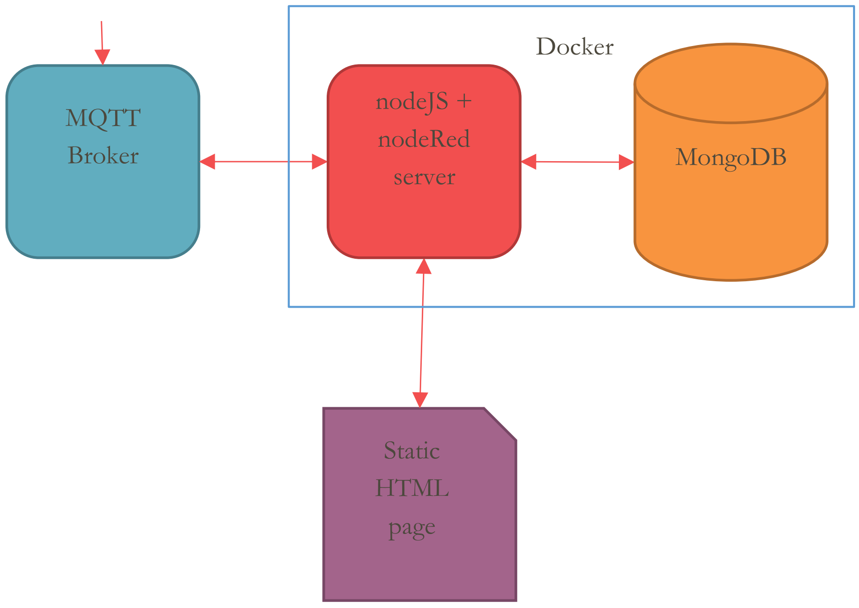
The content of the webpages is hosted on a web server, that runs a php, nodejs, or other webserver framework, allowing creation off HTML5 pages. The content is partially filled by the webserver, and completed on the client side relying on Javascript, jQuery and websockets.

The look and feel of the website and web pages is defined inside css stylesheet file. Various css and javascript frameworks exist to build powerful and user friendly content. One of the most popular to build screen size agnostic content, is bootstrap.

Architecture

The Web Application that we are using today is composed of different items:

- HTML that contains the skeleton of the web page and adapt on the client side to show the values retrieved
- Javascript used to address RESTFull API from client site
- JQuery used to modify on the client side the content of the web page
- Websocket to establish a connection with the webservice callback functions



MQTT Broker

In Lab02 we used an MQTT broker running on the Virtual Machine and we are going to continue, as the Virtual Machine is becoming our Cloud Server.

Install MQTT broker

The MQTT broker should already be installed in the VirtualMachine.

For the Lab purpose you need to install a MQTT broker.

1. Add the line below in your /etc/apt/source.list file.

```
osboxes@osboxes:~$ deb http://ppa.launchpad.net/mosquitto-dev/mosquitto-ppa/ubuntu trusty main
```

2. Then you have to update the local packages cache and installed the mosquitto one

```
osboxes@osboxes:~$ sudo apt-get update && sudo apt-get install mosquito mosquitto-clients
```

The mosquitto daemon start at the installation and listen by default on 1880 port on localhost.

Test the MQTT broker

3. Using the MQTT Spy application, and instructions from Lab02 validate that the BLE data are sent over the network to the MQTT broker.

Node-red and mongoDB

To simplify the deployment of the solution the Node-Red and mongoDB services are hosted into a docker image.

Install, configure and run Docker image

The local services on the target will be managed within docker containers to simplify the deployment of the solution and the maintainability of the system (instructions from https://nodered.org/docs/platforms/docker).

```
osboxes@osboxes:~$ docker pull billounet/node-red-cloud
```

Create a folder called nodes in your home folder.

```
osboxes@osboxes:~$ mkdir nodes
osboxes@osboxes:~$ mkdir nodes/db
osboxes@osboxes:~$ chmod o+w -R nodes
```

Create a docker container from the docker image.-

```
osboxes@osboxes:~$ docker run -it --rm -p 1880:1880 -p 27017:27017 -p
28017:28017  -v  "/home/osboxes/docker/nodes:/data"  --name  cloudnode
billounet/node-red-cloud


> node-red-docker@1.0.0 start /usr/src/node-red
> node  $NODE_OPTIONS node_modules/node-red/red.js -v $FLOWS "--userDir"
"/data"

5 Mar 15:35:09 - [info]

Welcome to Node-RED
===================

5 Mar 15:35:09 - [info] Node-RED version: v0.19.6
5 Mar 15:35:09 - [info] Node.js  version: v6.16.0
5 Mar 15:35:09 - [info] Linux 4.18.0-15-generic x64 LE
5 Mar 15:35:09 - [info] Loading palette nodes
…
```

Press Ctrl-p Ctrl-q to close the terminal.

```
# Restart and stop the container
docker stop cloudnode
docker start cloudnode
```

Database creation

The Database is created in the ~/nodes/db folder of the host computer and there is no action required to get the database ready.

Create Node-Red Flow

Basic node-red

1. Start node-red on the local Virtual Machine from a terminal

```
osboxes@osboxes:~$ node-red
```

2. Open a web browser to the node-red server running inside the Virtual Machine http://127.0.0.1:1880
3. Start a new flow.

All data format expected on websockets in the next Lab state can be identified by inspecting the index.html file that consumes the data.

To get rid of the CORS (Cross Origin Resource Share) some HTTP parameters need to be added to the answer data through a Function node.

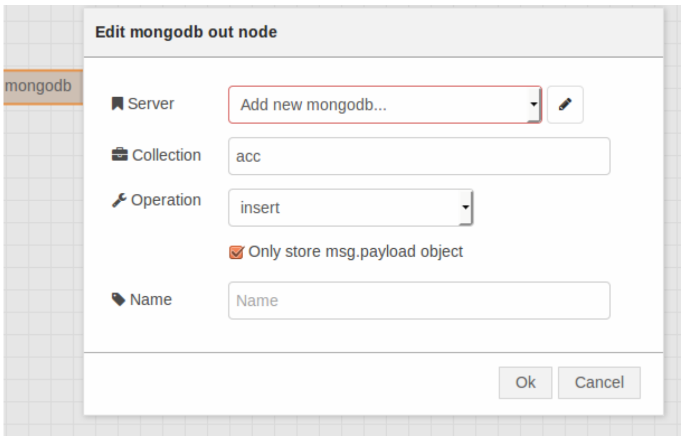
msg.res.set("Access-Control-Allow-Origin","");*
return msg;

Generic usage of MongoDB node and Websocket

The received data are usually in raw in variable msg.payload. We should transform it in storable data for mongo DB. Mongo works with json. We will add the date timestamp before storage, through a custom function node

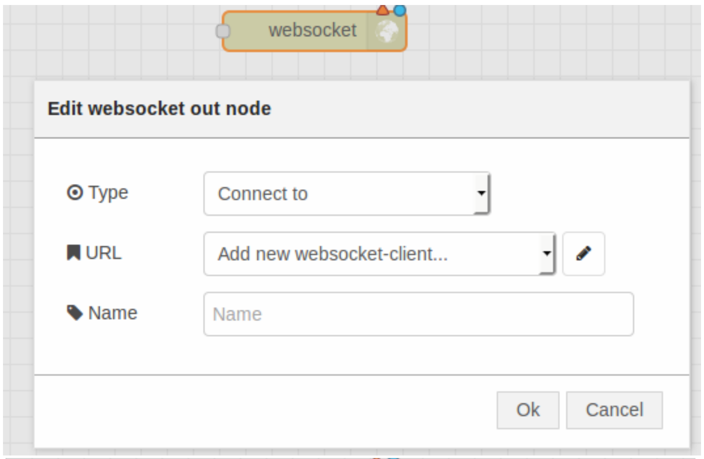
```
var date = new Date();
msg.payload = {acc: msg.payload, date: date.toISOString()};
return msg;
```

Configuration of a mongobd output node is where the collection is the "table" used to store the data.


Edit mongodb out node

Server: Add new mongodb...
Collection: acc
Operation: insert
✓ Only store msg.payload object
Name: Name

In addition to the store we will disseminate information on a websocket. This websocket will be use by the website. Add a websocket node output

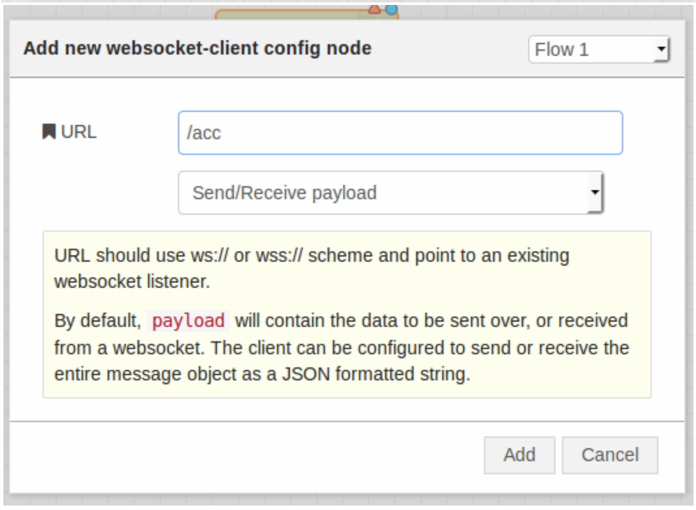


Edit websocket out node

Type: Connect to

URL: Add new websocket-client...

Name: Name



Add new websocket-client config node

URL: /acc

Send/Receive payload

URL should use ws:// or wss:// scheme and point to an existing websocket listener.

By default, `payload` will contain the data to be sent over, or received from a websocket. The client can be configured to send or receive the entire message object as a JSON formatted string.

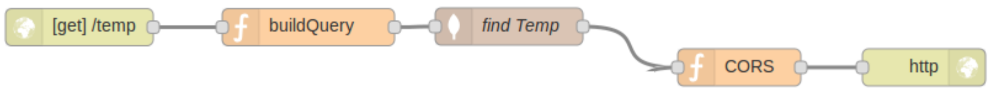
To check if data have been stored in mongodb. In another terminal you can run the following command:

```
osboxes@osboxes:~$ docker ps
CONTAINER ID    IMAGE
7ac5dec792f4    billounet/node-red-cloud:latest   …
osboxes@osboxes:~$ docker exec -it 7ac5dec792f4 /bin/bash
node-red@osboxes:~$ mongo
MongoDB shell version: 3.2.11
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
> show dbs
database  0.000GB
local     0.000GB
> use database
switched to db database
> show collections
temp
>db.temp.find()
```

```
>db.temp.find().count()
```

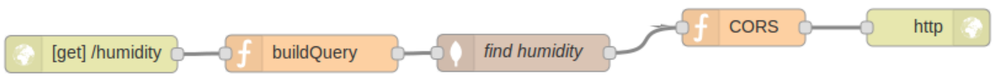
MQTT Temperature Flow

4. Start by adding an MQTT node subscribing to the temperature topic used on Lab 02
5. Add a function that will manipulate the receive data and adapt the format in order to be able to store it to our mongoDB Database. Data to be stored shall be timestamped with the current date and time of the server.
6. Add a mongoDB node to store the value in Database
7. In addition to store in database, we need the values to be published on a WebSocket out node, to allow dynamic data update on a web page.
   Expecting websocket value to be reachable on /temp url
8. In addition to forward and publish the values to allow dynamic data update on a web page, we need to retrieve the last 5 minutes values from database on connection to the websocket.
   Expecting websocket value to be reachable on /temp url



MQTT Humidity Flow

1. Start by adding an MQTT node subscribing to the humidity topic used on Lab 02
2. Add a function that will manipulate the receive data and adapt the format in order to be able to store it to our mongoDB Database. Data to be stored shall be timestamped with the current date and time of the server.
3. Add a mongoDB node to store the value in Database
4. In addition to store in database, we need the values to be published on a WebSocket out node, to allow dynamic data update on a web page.
   Expecting websocket value to be reachable on /humidity url
5. In addition to forward and publish the values to allow dynamic data update on a web page, we need to retrieve the last 5 minutes values from database on connection to the websocket.
   Expecting websocket value to be reachable on /humidity url

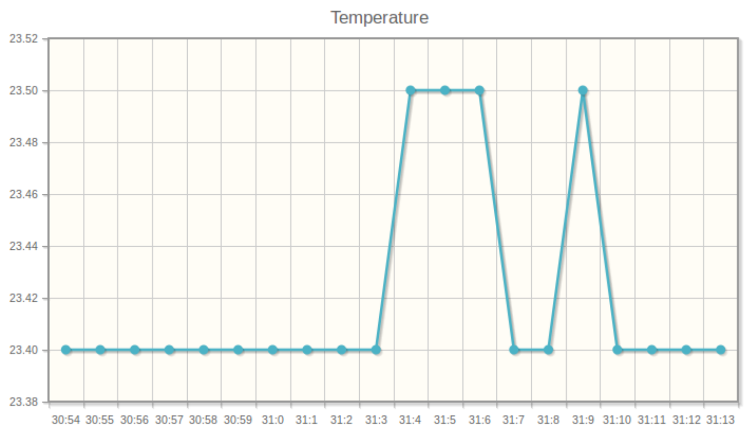


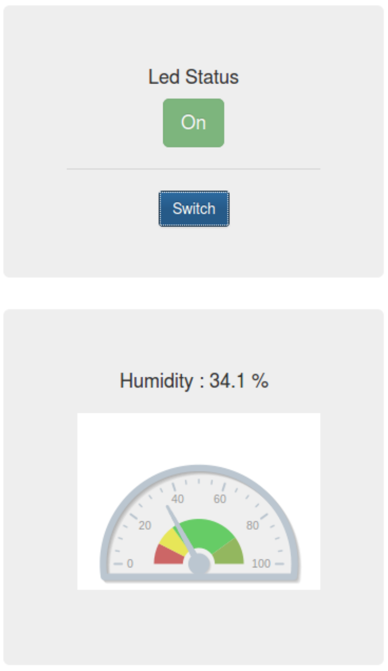
MQTT LED Flow

1. Start by adding an MQTT node publishing to the LED set State topic used on Lab 02
2. Add one trigger node to turn ON the LED
3. Add one trigger node to turn OFF the LED
4. Add a WebSocket in node to receive the ON and OFF commands from web page
   Expecting websocket value to be reachable on /ledstatus url
5. Connect the web socket to request the right LED status transition
6. Start by adding an MQTT node subscribing to the LED get State topic used on Lab 02
7. Report the LED State on a WebSocket out node to allow dynamic data update on a web page.
   Expecting websocket value to be reachable on /ledstatus url

Test the web service

8. Open the index.html file in your web browser.
9. Check that data are displayed in the browser.





Accelerometer support

node-red, MQTT broker, Gateway and BLE device should be up and running.

For now the provided web service supports the LED, thermometer and humidity sensor, and you have to develop the Accelerometer. Based on the existing work, you have to implement the web service and websocket interface that will grab the MQTT data and stores into the mongoDB. The static web page will be competed to add this accelerometer support.

1. Add a dedicated Flow to handle the Accelerometer values
2. Update the web page to display the X,Y and Z value in "real time"

Debugging the web page

Firefox/Chrome as a debugging tool

Modern browser allows HTML page inspections and debugging, which helps a lot of to fine tune the websites. By hitting the F12 key, you can access web developer tools.

1. Open the web developer tools and inspect the web page source code, check the stylesheets.
2. Open the Debugger tab and select index.html from the Sources list.
3. Scroll to the Accelerometer function

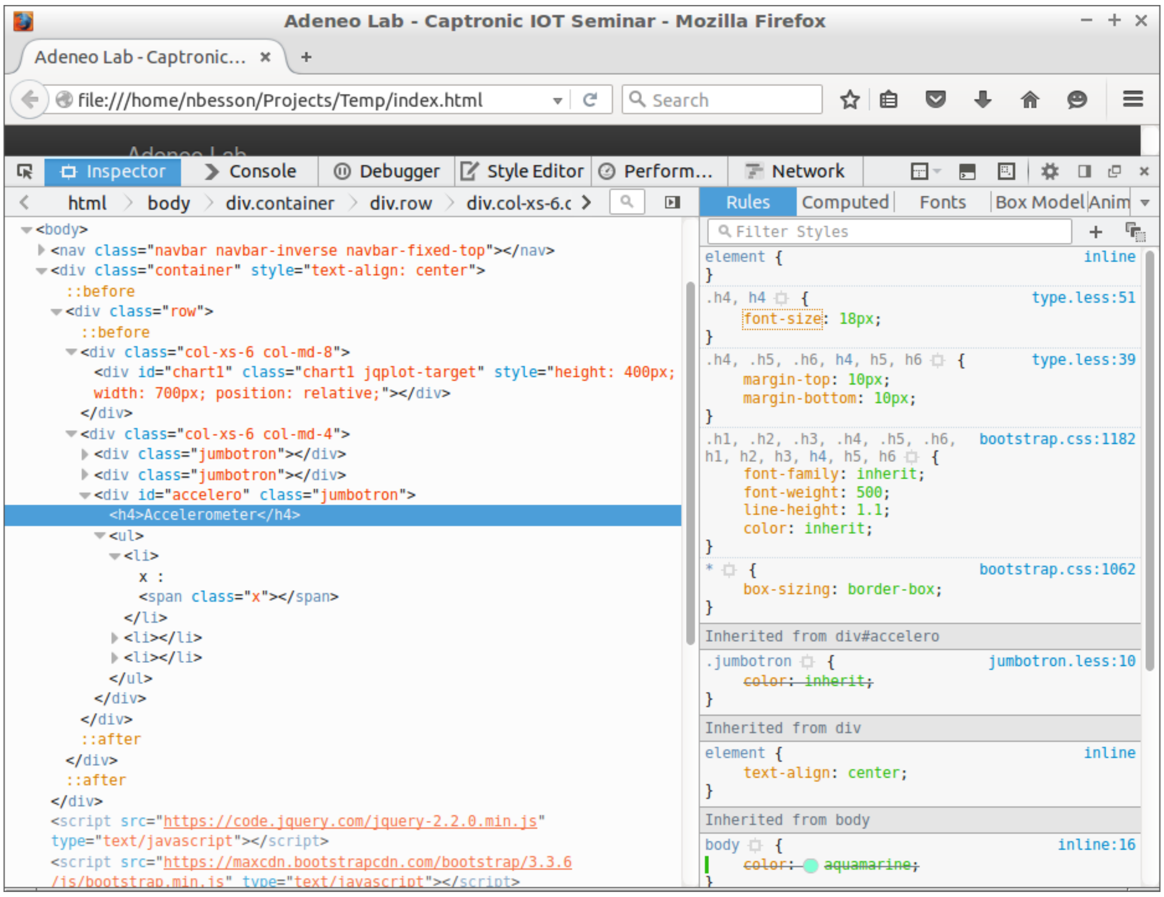
```
worker.socketHum.onmessage
```

4. Select the first line of the function
5. Right click and select Add Breakpoint
6. When break point is reached, inspect the variable values and perform step by step debugging.

Firefox as a designer tool

Using the developer tool, you can address the stylesheets of the DOM elements. An HTML tag can have a or multi class properties, that then can be used to target set of tags to apply styles properties.

7. From the web developer tools, open the Inspector tab.
8. On the web page, select the Accelerometer text, right click and choose the Inspect Element (Q) menu.



```
<body>
  <nav class="navbar navbar-inverse navbar-fixed-top"></nav>
  <div class="container" style="text-align: center">
    ::before
    <div class="row">
      ::before
      <div class="col-xs-6 col-md-8">
        <div id="chart1" class="chart1 jqplot-target" style="height: 400px;
        width: 700px; position: relative;"></div>
      </div>
      <div class="col-xs-6 col-md-4">
        <div class="jumbotron"></div>
        <div class="jumbotron"></div>
        <div id="accelero" class="jumbotron">
          <h4>Accelerometer</h4>
          <ul>
            <li>
              x :
              <span class="x"></span>
            </li>
            <li></li>
            <li></li>
          </ul>
        </div>
      </div>
      ::after
    </div>
    ::after
  </div>
  <script src="https://code.jquery.com/jquery-2.2.0.min.js"
  type="text/javascript"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6
  /js/bootstrap.min.js" type="text/javascript"></script>
```

```
element {                                        inline
}
.h4, h4 {                                 type.less:51
    font-size: 18px;
}
.h4, .h5, .h6, h4, h5, h6 {               type.less:39
    margin-top: 10px;
    margin-bottom: 10px;
}
.h1, .h2, .h3, .h4, .h5, .h6,     bootstrap.css:1182
h1, h2, h3, h4, h5, h6 {
    font-family: inherit;
    font-weight: 500;
    line-height: 1.1;
    color: inherit;
}
* {                               bootstrap.css:1062
    box-sizing: border-box;
}
Inherited from div#accelero
.jumbotron {                         jumbotron.less:10
    ~~color: inherit;~~
}
Inherited from div
element {                                        inline
    text-align: center;
}
Inherited from body
body {                                        inline:16
    ~~color: aquamarine;~~
}
```

9. The Accelerometer element should be highlighted in the Inspector, and the style sheet is displayed on the right side of the screen.
10. Select the h4 entry in the style sheet description and below font-size type:

```
color: cyan;
```

11. The result should be directly visible in the Firefox window.

Note that all H4 (header) elements are impacted by the modification. If you need to impact only that specific H4 element, then you have to specify a path to the concerned H4 element.

12. Specify a path to the H4 element of the #accelero div, by clicking on the + button located in the top right menu of the style sheet editor.
13. Enter the following configuration:

```
#accelero h4 {
      color: #008000;
}
```

The modifications applied are temporary, and will be lost when the page will be reloaded. So you need to develop your stylesheet file for the current webpage and include it at page loads.

Edit the index.html file on your hard disk in order to load a custom stylesheet file. With all H4 elements to be in cyan and the #accelero h4 element to be displayed as green (#008000).

Animating the cube

Get inspired by this technical article (https://www.awwwards.com/creating-3d-cube-a-practical-guide-to-three-js-with-live-demo.html)and make dreams come true, by having the device changing the position of the cube on the screen.