TP4 : Application IoT

Gilles Menez - UCA - Dépt. Informatique

November 2, 2022

---

1 Architecture d'une application IoT

Une infrastructure d'IOT repose généralement sur 3 acteurs :

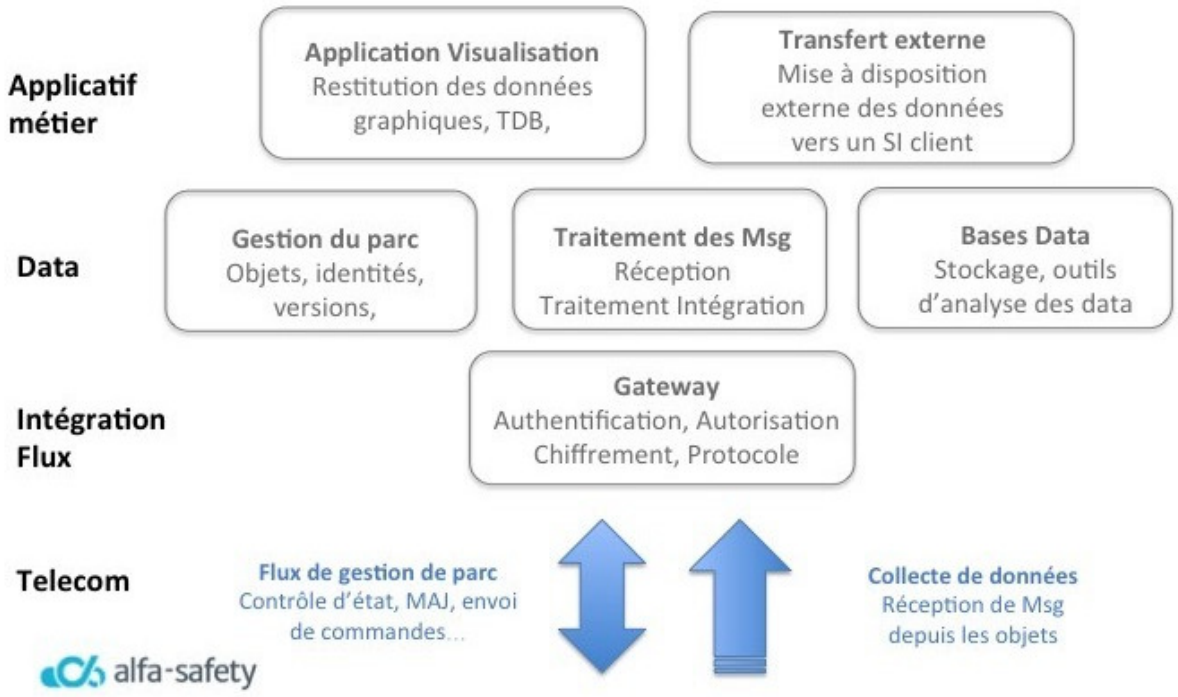
① un parc d'objets connectés fixes ou mobiles, répartis géographiquement,

② un réseau telecom (filaire ou pas ou un peu ou beaucoup) qui va permettre de connecter les objets en transmettant des messages.

③ une application qui collecte les "data" du réseau d'objets pour fournir une information agrégée plus ou moins "intelligente" ou pour contrôler un système (un parc éolien par exemple).

1.1 Fonctionnalités

Une telle application comporte des fonctionnalités "classiques" réparties sur différents plans : métier, data, gestion des flux, télécommunications, ...



1.1.1 Gateway : intégration des flux

La fonction Gateway est la "porte d'échange" des messages entre l'application et le parc des objets, **elle s'interface avec le réseau de télécommunications** (via le support de protocoles "application" de type MQTT / HTTP / ...) et se charge tout particulièrement de la sécurité :

- ✔ Authentifier et autoriser les objets à dialoguer avec l'application.
- ✔ Contrôler l'intégrité des données et donc empêcher que l'on puisse injecter des jeux de données fictives à partir d'objets illégitimes (c'est le minimum !)
- ✔ Préserver la confidentialité des échanges par exemple par un chiffrement.

L'implémentation de la sécurité dépendra du protocole et du mode de dialogue retenu.

Bien souvent cette sécurité viendra alourdir les coûts de traitements et de transmissions et par conséquent le coût énergétique.

- ✔ Il y a là des arbitrages à faire selon la nature de l'application.

En exploitation, on veillera à mettre en oeuvre une supervision applicative adaptée qui permettra de détecter toute anomalie des flux de messages : Le minimum est de détecter les coupures de transmission !?

1.1.2 Le traitement des messages

Dans le contexte des traitements opérés sur l'information qui remonte des objets, **la question de la volumétrie est critique**.

✔ Même si à un instant donnée cette fonctionnalité semble correctement dimensionnée, elle (cette fonction) doit pouvoir absorber (réceptionner, traiter et intégrer) **un volume de messages très fluctuant**.

"Fluctuant" parce les objets transmettent l'état du monde réel/physique et que si cet état évolue, le volume de données à de grande chance d'évoluer car les techniques d'échantillonnages sont souvent (très) réactives à des évenements ou à des variations.

Sur une échelle de temps "plus maitrisable" l'augmentation du parc d'objets engendre aussi une fluctuation de la volumétrie.

✔ Cette évolution peut remettre en cause l'architecture initiale : cf "le besoin MQTT" !

1.1.3 La gestion du parc

Un parc d'objets est en constante évolution : augmentation des capacités techniques, générations de matériels et de logiciels ...

✔ La gestion du parc est donc vital : inventaire, status, mise à jour ...

1.1.4 Les bases de données

Le parc d'objets va alimenter l'application d'un flux données qui doit être analyser.

Or cette analyse est d'autant plus pertinente qu'elle s'appuie sur un horizon temporel adapté.

➢ Pour générer cet horizon, il faut "stocker" ... il faut donc des Bases de Données ... CQFD !

1.1.5 Le serveur d'application

L'objectif d'une application d'IOT est de traiter/analyser puis de présenter les connaissances aux utilisateurs.

➢ Cette "connaissance" peut consister en des données brutes ou en des résultats d'analyses poussées.

➢ Souvent une présentation graphique synthétise cela graphiquement sur un "Dashboard".

➢ Le serveur d'application peut être accéder depuis le Web ou depuis un smartphone ou encore un outil dédié.

L'accès à ce serveur peut être limité si l'audience est ciblée, il peut devenir important sur une audience grand public. Là encore, attention au dimensionnement.

1.1.6 Les transferts externes de data

Un autre mode d'usage des données est de les **transférer vers le SI d'un client qui va à son tour les intégrer pour les exploiter dans son cas d'usage particulier** :

➢ Un opérateur de service IOT propose à ses clients de s'abonner à un service d'information basé sur son réseau d'objets, les données sont remontées dans le SI du client final qui va s'en servir pour produire ses propres services, comme une société de surveillance ou maintenance qui transmet certaines alarmes à un sous-traitant qui se charge d'intervenir sur site.

Enfin, on veillera ici encore à mettre en oeuvre une supervision applicative efficace des flux et notamment sur le plan des droit d'accès.

1.2 Device Management

1.2.1 A ”petite” échelle . . .

Les applications IoT permettant des fonctionnalités IoT ”limitées” sont généralement suffisantes pour les scénarios IoT de base.

Ces application peuvent alors :

- ➢ connecter des dispositifs et des capteurs au nuage/cloud,
- ➢ surveiller et collecter des données,
- ➢ et fournir une visualisation du projet IoT.

Nous voyons un certain nombre de ces plates-formes IoT proposées par des fabricants de matériel qui les introduisent en complément ou dans le cadre de leur offre de matériel.

Les plates-formes IoT de ”base” sont ”suffisantes” pour les projets IoT de petite et moyenne envergure .

La domotique rentre typiquement dans cette catégorie d’applications :

- ➢ peu d’objets,
- ➢ peu de diversités d’objets,
- ➢ peu de réseaux,
- ➢ peu de protocoles,
- ➢ des distances réduites,
- ➢ . . .

en résumé, une complexité et une diversité restreinte.

Si on devait faire une analogie avec le monde des machines, **la domotique est l’équivalent d’une salle machines/PC telle que celle de TP** et on **mesure bien la complexité réelle MAIS limitée** d’une plateforme/application logicielle permettant la gestion d’une telle salle.

1.2.2 A ”grande” échelle

Un tel scénario/configuration peut rapidement évoluer et se compliquer !

Si il y a plusieurs salles machines, sur des sites différents avec des réseaux hétérogènes, avec des machines de natures (Hardware/OS) différentes, avec des fonctions différentes (serveurs, gateway, . . .) . . .

Pour les scénarios IoT à grande échelle (Smartcity, Farming,. . .) on retrouve toutes ces problématiques (amplifiées) :

- ➢ La maintenance du flux d’information sensé remonté vers le centre du réseau nécessite une véritable **plateforme de gestion** des objets.

  Cette plateforme va devoir gérer de nouvelles fonctionnalités telles que la connexion, la gestion et l’intégration de milliers de périphériques et de capteurs.

L'offre commerciale qui suit montre bien les différences entre les fonctionnalités d'une plateforme "générique" ne nécessitant pas et n'abordant pas ces problématiques et une plateforme de plus grande échelle :

Device Management – a MUST component of IoT Platform

Features	Friendly's Device Management	Generic IoT Platform
Provisioning	✓ Fully Automated	✗ Minimal Capability
Management of Devices with Complex Data Models	✓ Yes	✗ No
Types of Managed Devices	✓ Any type of device	✗ MQTT Devices Only
Remote Configuration	✓ Fully Automated	? Minimal Capability
Monitoring & Event Triggering	✓ Yes	✓ Yes
Data Collection	✓ Yes	✓ Yes
Group Update	✓ Yes	✗ No
Firmware Upgrade	✓ Yes	✗ No
Device Diagnostics & Repair	✓ Yes	✗ No
Application	✓ Integration with 3rd Party	✓ Yes

* provisioning : `https://medium.com/alvarium/iot-device-provisioning-671131600ab1`

2 Les tentatives de standardisation

On a évoqué en cours la problématique de solutions IoT développées en silos . . . avec autant de ”standards”.

La complexité de la tâche est grande notamment parce que le problème a plusieurs dimensions :

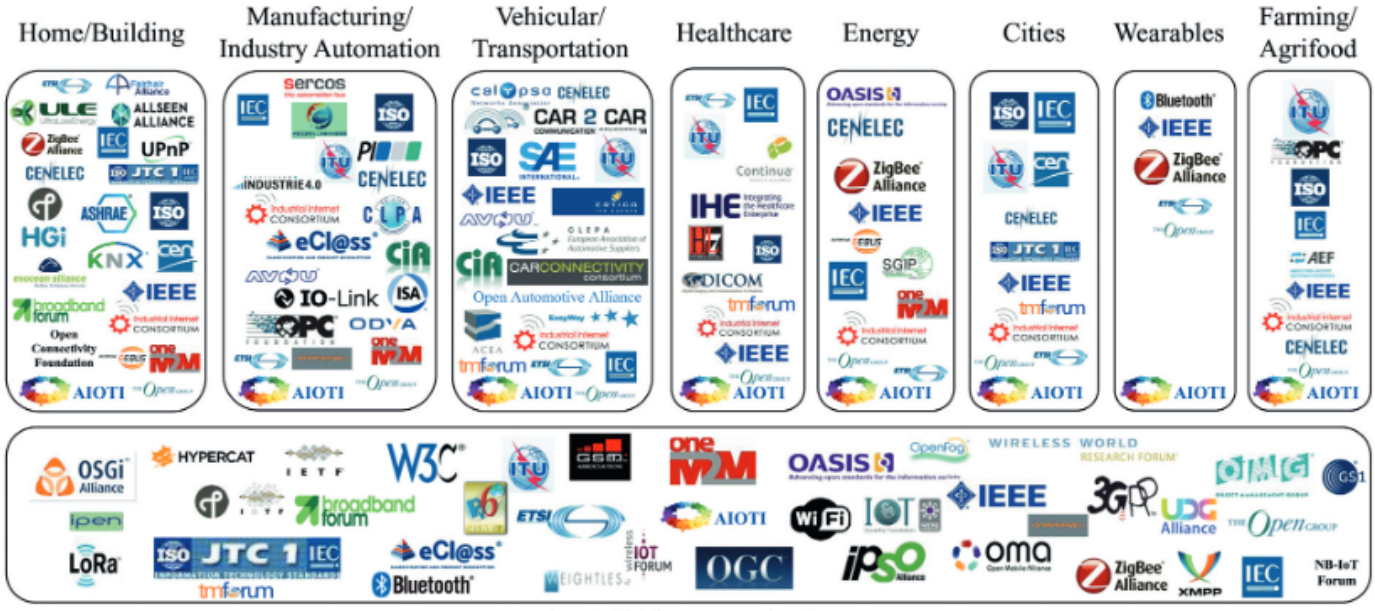
① Il y a les domaines d'applications représentés verticalement.

Ils sont certainement anciens, éventuellement avec des contraintes spécifiques.

② Horizontalement, les infrastructures de télécommunications qui cherchent à répondre aux besoins et à capter ces flux.

Jusqu'au niveau où Internet met tout le monde d'accord, **les technologies et les protocoles sont multiples et potentiellement concurrentes**.

③ Et enfin il y a le ”business”, dimension sous jacente à tout ”marché” qui fait par exemple que l'utilisateur se retrouve avec des dizaines de standards différents de prises USB ;-)



Cette figure liste les acteurs intervenants dans ces domaines.

Deux types d'acteurs :

➢ **Standards Developing Organization** (SDOs) qui développent et proposent des standards (plutôt technologiques) de l'IoT,

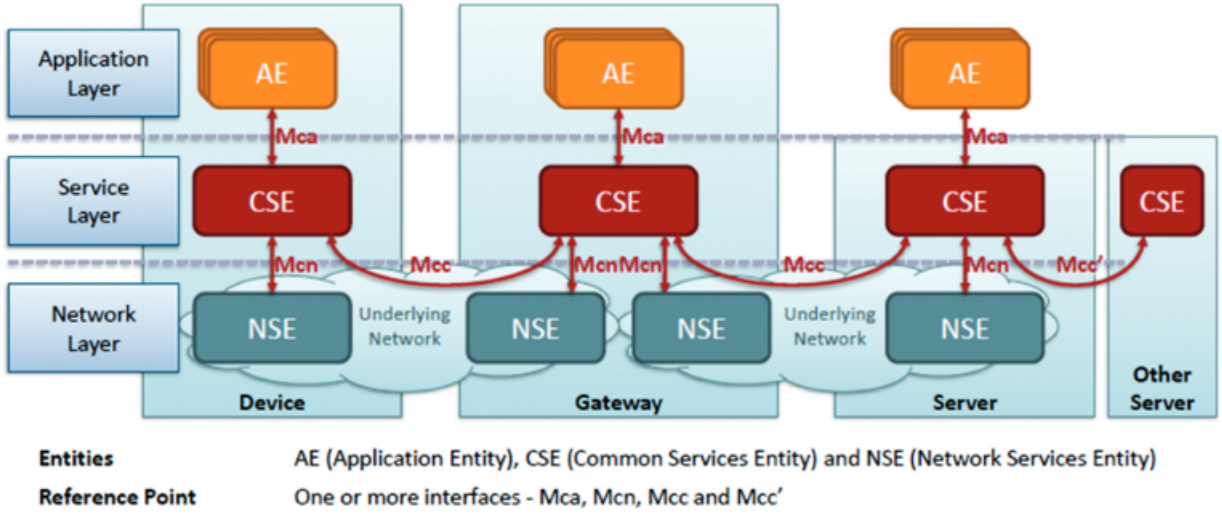
➢ et des **”alliances”** dont certaines ont bien compris qu'un standard pouvait aussi être ”de fait” et qui peuvent servir (lorsqu'elles agglomères des industriels) des objectifs plutôt commerciaux, marketing, promotionnels, . . .

2.1 OneM2M ?

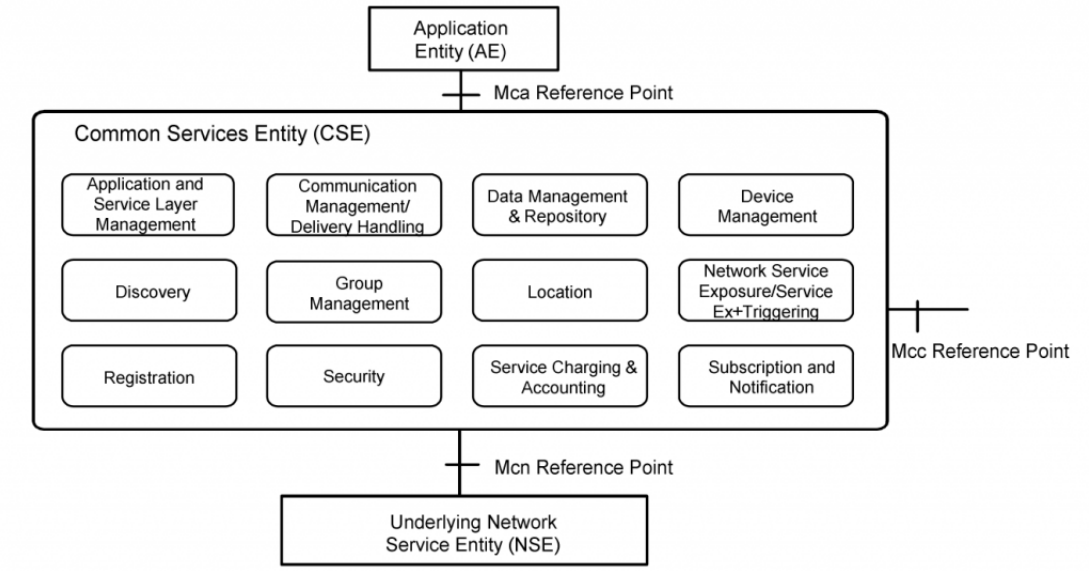
OneM2M est une alliance d'organismes de normalisation (SDOs) qui cherche à développer une plate-forme horizontale unique pour l'échange et le partage de données entre les applications.

`https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6387169/pdf/sensors-19-00676.pdf`

OneM2M permet l'interopérabilité entre les applications IoT, quelle que soit la technologie sous-jacente utilisée.



Pour cela, oneM2M définit une couche de services communs (Common Services Layer) , qui est une couche logicielle située entre le réseau et les applications, que ce soit dans le domaine du réseau étendu ou dans le domaine des fichiers (où les dispositifs et les passerelles sont généralement déployés).

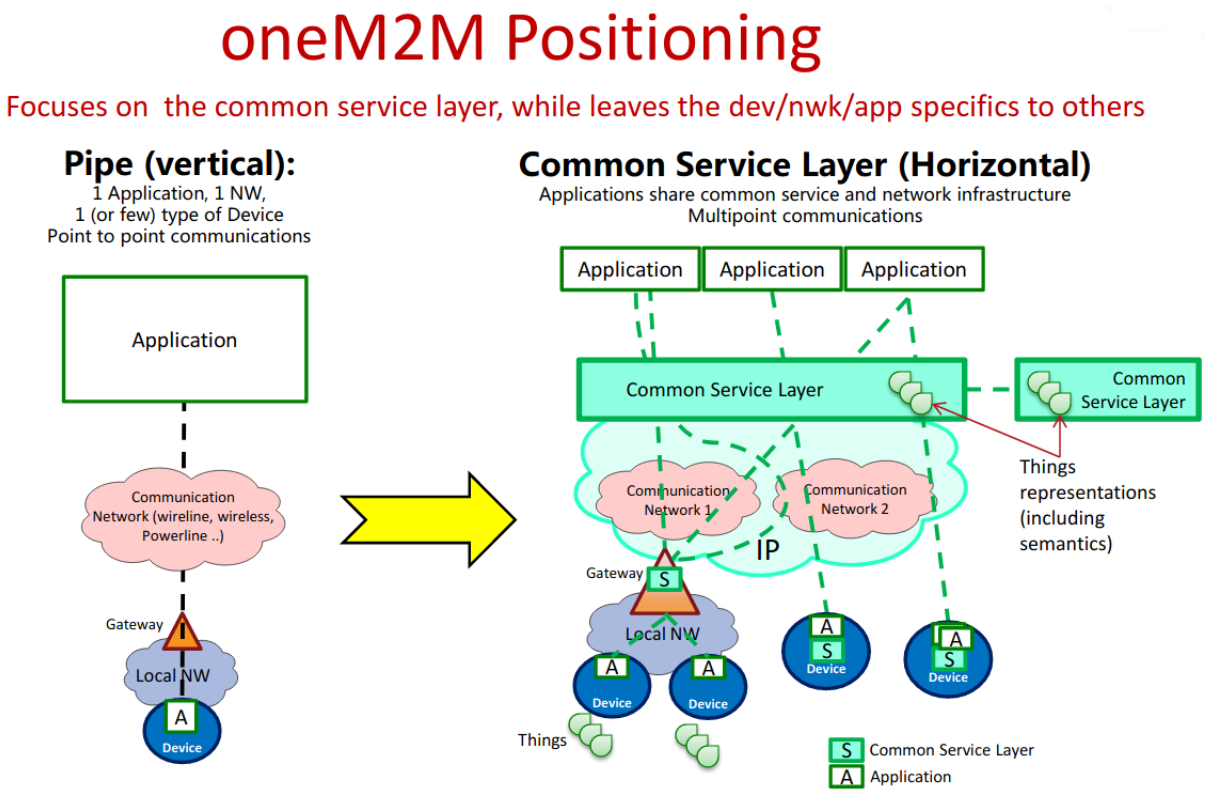


Les fonctions de cette couche de services comprennent :

- ✔ la gestion des appareils,
- ✔ la collecte de données,
- ✔ la conversion et l'interopérabilité des protocoles,
- ✔ la gestion de groupe,
- ✔ la sécurité, etc.

`https://www.onem2m.org/getting-started` :

Ces fonctions peuvent être déployées sur un serveur M2M (ou une gateway ou un device) et sont exposées aux applications (dans le nuage, les passerelles ou les appareils) via des API REST qui peuvent ainsi **interopérer** sur cette "base" de services.



➢ `https://www.w3.org/WoT/IG/wiki/images/a/ae/IoT_Standards_Interworking_%26_Collaboration.pdf`

Plusieurs implémentations OneM2M sont disponibles :

➢ `https://onem2m.org/developers-corner/tools/open-source-projects`

➢ `https://github.com/OpenMTC/OpenMTC`

➢ `https://www.eclipse.org/om2m/`

➢ https://github.com/IoTKETI/Mobius

Ceci étant, OneM2M n'est pas la seule initiative prônant la standardisation et elle n'a pas que des avantages:

➢ https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-oneM2M.html

➢ https://aioti.eu/wp-content/uploads/2017/06/AIOTI-HLA-R3-June-2017.pdf

Un des inconvénients de tout standard est qu'il faut investir en temps pour l'appréhender.

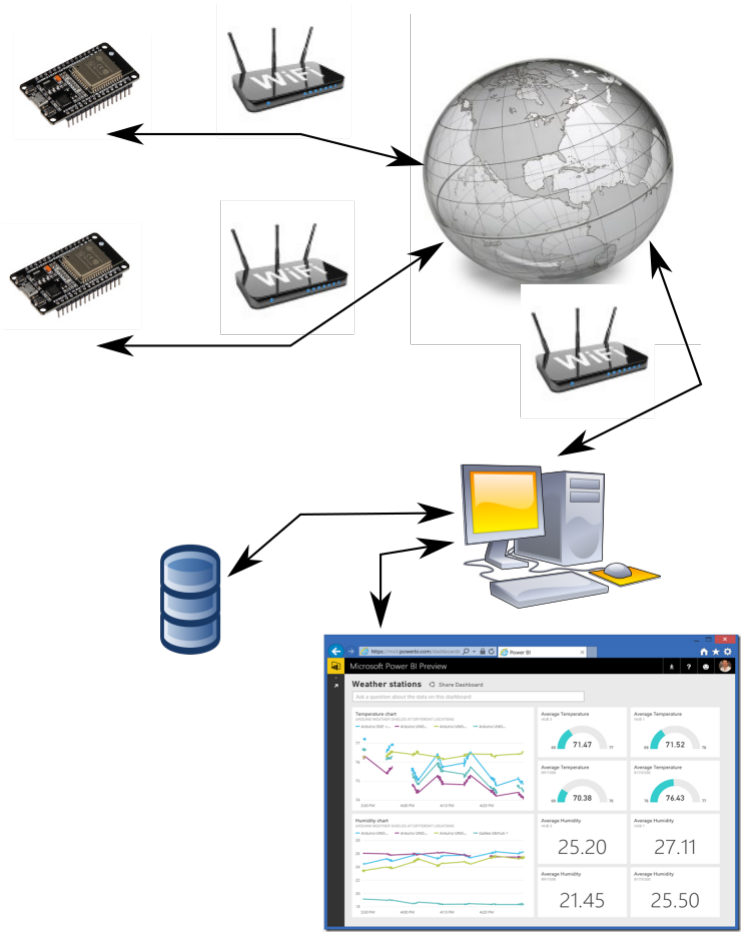
➢ Comme on n'a absolument pas le temps de faire cela durant cette UE, nous allons faire sans . . . c'est dommage . . . mais cela permettra de se confronter aux problèmes que proposent de résoudre les standards.

3 Une application ”générique”

Pour aborder les problématiques informatiques du domaine de l'IoT pourquoi ne pas essayer de programmer une application un peu générique ?

On ”reste” dans une architecture de type :

”des objets (ESP32) ¡-¿ 1 réseau ¡-¿ Station de monitoring (PC)”.



Par rapport aux réalisations précédentes, cette nouvelle application/architecture va se focaliser sur la

① La persistance :

On voit apparaître sur la figure une base de données sans laquelle il n'est pas possible de consolider/-construire un ”savoir” (référence au sommet de la ”pyramide des connaissances” dans le cours).

② Le déploiement dans le ”cloud” :

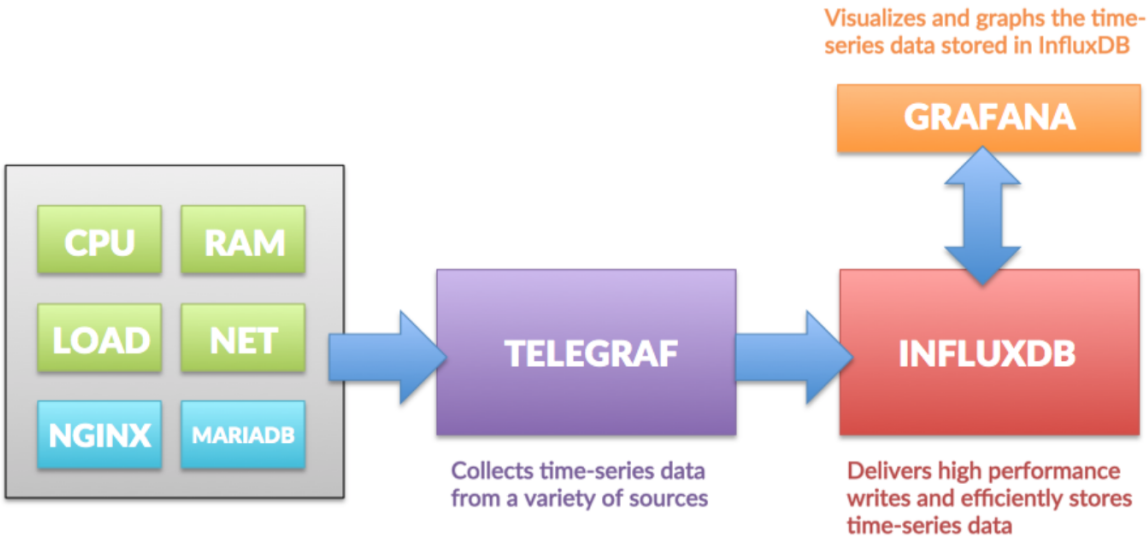
Je n'aime pas trop ce terme mais . . . votre application doit désormais exister sur les ressources mises à disposition dans l'espace Internet.

Son accessibilité devra être universelle et permanente !

3.1 TIG

Ce type d'application est suffisamment courant pour que des "piles d'outils" lui soit dédié.

➢ La stack "TIG" (**T**elegraf, **I**nflux et **G**rafana) est un acronyme désignant une plateforme d'outils open source conçus pour faciliter la collecte, le stockage, la création de graphiques et l'émission d'alertes sur les données de séries temporelles.



① Telegraf est un agent de collecte de métriques.

- ✔ Utilisez-le pour collecter et envoyer des métriques à InfluxDB.
- ✔ L'architecture des plugins de Telegraf permet de collecter des métriques à partir de plus de 100 services populaires dès le départ.

② InfluxDB est une base de données de séries chronologiques haute performance.

- ✔ Elle peut stocker des centaines de milliers de points par seconde.
- ✔ Le langage d'interrogation InfluxDB de type SQL a été conçu spécifiquement pour les séries temporelles.
- ✔ InfluxQL est un langage d'interrogation très similaire à SQL qui permet à tout utilisateur d'interroger ses données et de les filtrer.

  `https://devconnected.com/the-definitive-guide-to-influxdb-in-2019/`

③ Grafana est une plateforme open-source pour la visualisation, le suivi et l'analyse des données.

- ✔ Dans Grafana, les utilisateurs peuvent créer des tableaux de bord avec des panneaux, chacun représentant des métriques spécifiques sur une période donnée.
- ✔ Grafana prend en charge les panneaux de type graphique, tableau, heatmap et texte libre.

Si vous avez des bitcoins peut être que vous aimeriez surveiller leur cours ?

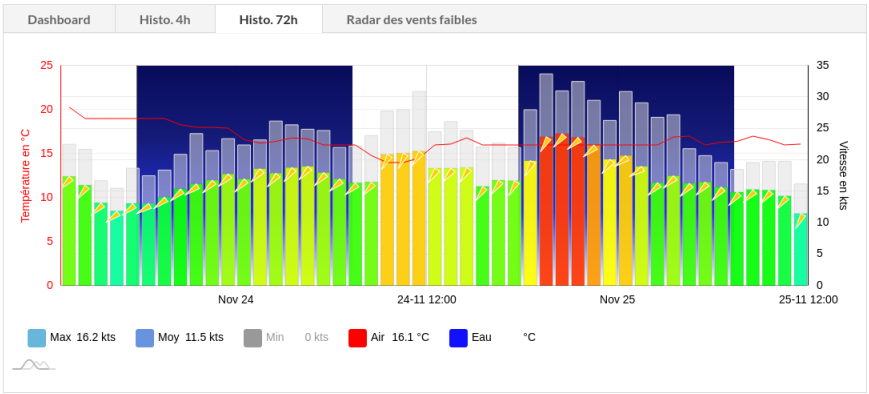
Sinon vous pouvez aussi surveiller les vents sur l'Etang de Thau . . . afin de ne pas rater une session de Kite !

Météo en temps réel de l'étang de Thau

La sonde Fildair du bassin de Thau.

Prévisions de vent sur le bassin de Thau

Notre sonde relève et fournit des informations en temps réel sur le vent et les températures d'eau et d'air au milieu du bassin de Thau dans le couloir de la Conque de Mèze au Lido de Sète.

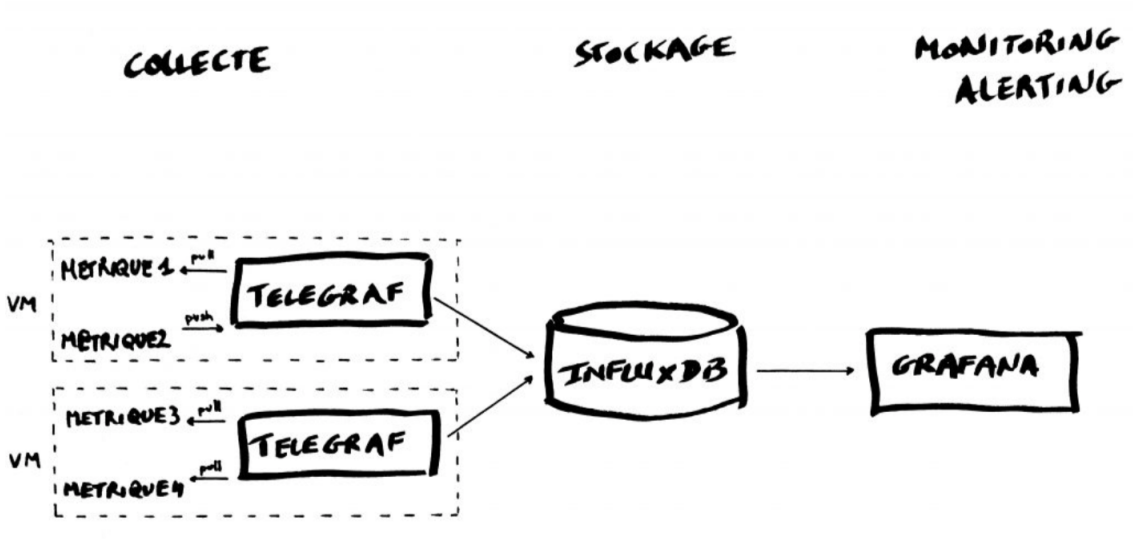


Je trouve le module JS utilisé pour représenter les séries temporelles particulièrement réussi . . . on y reviendra !

La ”surveillance”/supervision est une première fonctionnalité d'une telle application mais très rapidement les utilisateurs aimeraient disposer d'une prédiction de vent ou de cours du bitcoin

Ils aimeraient aussi être informer d'un comportement : une rupture, une chute brutale de cours, un vent anormal . . .

Ceci n'est faisable que si l'on dispose d'un horizon de la série temporelle qui serait par exemple stocké dans une base de données :



Le site suivant :

`https://hackmd.io/@lnu-iot/tig-stack`

n'aborde pas ces thématiques d'IA mais montre comment déployer une application de surveillance TIG **à partir** d'une approche docker.

Si on avait plus de temps, il est certain que je vous aurais demander d'essayer et de m'en faire un compte rendu !

Mais on n'a pas assez de temps :-(pour que cela soit obligatoire. C'est juste ”si” . . . vous avez un peu de temps . . . au lieu de faire une partie de PS5 ? . . .

4 Approche ”par la programmation”

Dans ce qui suit nous allons réaliser quelque chose de similaire en utilisant moins d’”outils” et plus de programmation JS.

➢ Je me méfie toujours un peu des outils car on sait quand on y entre . . . moins quand (et si) on peut en sortir et faire sans ?

Les codes de base/démarrage sont donnés . . . sur le site.

4.1 Programmation de ESP

Première hypothèse de cette application :

➢ L’ESP utilise MQTT pour publier régulièrement son statut.

On aurait pu choisir de mettre en place des requêtes POST (HTTP) sur un serveur jouant le rôle d’un ”broker” . . . why not ? . . . pour se décider, il faudrait pousser plus avant les spécifications. L’idée c’est d’essayer autre chose que le TP prédécent.

Par contre **ne cassez surtout PAS** ce que vous avez mis en place avec le POST périodique du statut vers un hôte spécifié ! Même si je n’ai pas réussi à faire fonctionner simultanément les deux asynchronismes des serveurs HTTP et MQTT. On peut garder le principe/l’architecture si on passe en synchrone.

4.1.1 Fichier : esp32_lucioles/esp32_lucioles.ino

Ce code reprend la structure des programmes déjà réalisés.

➢ La seule partie de code ”un peu nouvelle” est dans la ”loop()”.

```
/*=======================================================================
 * Auteur : G.Menez
 =======================================================================*/
// SPIFFS
#include <SPIFFS.h>
// OTA
#include <ArduinoOTA.h>
#include "ota.h"
// Capteurs
#include "OneWire.h"
#include "DallasTemperature.h"
// Wifi (TLS) https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFiClientSecure
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "classic_setup.h"
// MQTT https://pubsubclient.knolleary.net/
#include <PubSubClient.h>

#define USE_SERIAL Serial

/*===== ESP GPIO configuration ==============*/
/* ---- LED         ----*/
const int LEDpin = 19; // LED will use GPIO pin 19
/* ---- Light       ----*/
const int LightPin = A5; // Read analog input on ADC1_CHANNEL_5 (GPIO 33)
```

```
/* ---- Temperature ----*/
OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
DallasTemperature TempSensor(&oneWire) ; // Cette entite est utilisee par le capteur de temperature

String whoami; // Identification de CET ESP au sein de la flotte

/*===== JSON ================================*/
//StaticJsonBuffer<200> jsonBuffer;

/*===== WIFI ================================*/
#undef TLS_USE
#ifdef TLS_USE
WiFiClientSecure secureClient;     // Avec TLS !!!
#else
WiFiClient espClient;              // Use Wifi as link layer
#endif

/*===== MQTT broker/server and TOPICS ========*/
//String MQTT_SERVER = "192.168.1.101";
String MQTT_SERVER = "test.mosquitto.org";
#ifdef TLS_USE
int MQTT_PORT =  8883; // for TLS cf https://test.mosquitto.org/
#else
int MQTT_PORT = 1883;
#endif

//==== MQTT Credentials =========
/*#define MQTT_CRED*/
#ifdef MQTT_CRED
char *mqtt_id     = "deathstar";
char *mqtt_login  = "darkvador";
char *mqtt_passwd = "6poD2R2";
#else
char *mqtt_id     = "vador";
char *mqtt_login  = NULL;
char *mqtt_passwd = NULL;
#endif

//==== MQTT TOPICS ==============
#define TOPIC_TEMP  "uca/M1/iot/temp"
#define TOPIC_LED   "uca/M1/iot/led"
#define TOPIC_LIGHT "uca/M1/iot/light"
#ifdef TLS_USE
PubSubClient client(secureClient); // MQTT client
#else
PubSubClient client(espClient);      // The ESP is a MQTT Client
#endif

void mqtt_pubcallback(char* topic, byte* message, unsigned int length) {
  /**  MQTT Callback ... if a message is published on this topic.   */

  // Byte list to String ... plus facile a traiter ensuite !
  // Mais sans doute pas optimal en performance => heap ?
  String messageTemp ;
  for(int i = 0 ; i < length ; i++) {
    messageTemp += (char) message[i];
  }

  USE_SERIAL.printf("Message : %s => arrived on topic : %s\n", messageTemp, topic);
  // Analyse du message et Action
  if(String (topic) == TOPIC_LED) { // Par exemple : Message sur topic LED => Changes the LED output state according to the messag
    USE_SERIAL.printf("Action : Changing output to %s", messageTemp);
```

```
    if(messageTemp == "on") {
      set_pin(LEDpin,HIGH);

    } else if (messageTemp == "off") {
      set_pin(LEDpin,LOW);
    }
  }
}

void setup_mqtt_client() {
  /**  Setup the MQTT client  */

  // set server
  client.setServer(MQTT_SERVER.c_str(), MQTT_PORT);
  // set callback when publishes arrive for the subscribed topic
  client.setCallback(mqtt_pubcallback);
}

void mqtt_connect() {
  /** Connection to a MQTT broker  */

#ifdef TLS_USE
  // For TLS
  const char* cacrt = readFileFromSPIFFS("/ca.crt").c_str();
  secureClient.setCACert(cacrt);
  const char* clcrt = readFileFromSPIFFS("/client.crt").c_str();
  secureClient.setCertificate(clcrt);
  const char* clkey = readFileFromSPIFFS("/client.key").c_str();
  secureClient.setPrivateKey(clkey);
#endif

  while (!client.connected()) { // Loop until we're reconnected
    USE_SERIAL.print("Attempting MQTT connection...");

    // Attempt to connect => https://pubsubclient.knolleary.net/api
    if (client.connect(mqtt_id,      /* Client Id when connecting to the server */
                       mqtt_login,   /* With credential */
                       mqtt_passwd)) {
      USE_SERIAL.println("connected");
    }
    else {
      USE_SERIAL.print("failed, rc=");
      USE_SERIAL.print(client.state());

      USE_SERIAL.println(" try again in 5 seconds");
      delay(5000); // Wait 5 seconds before retrying
    }
  }
}

void mqtt_subscribe(char *topic) {
  /**  Subscribe to a topic  */
  if (!client.connected())
    mqtt_connect();

  client.subscribe(topic);
}

/*============= ACCESSEURS ====================*/

float get_temperature() {
  float temperature;
```

```
  TempSensor.requestTemperaturesByIndex(0);
  delay (750);
  temperature = TempSensor.getTempCByIndex(0);
  return temperature;
}
float get_light(){
  return analogRead(LightPin);
}
void set_pin(int pin, int val){
 digitalWrite(pin, val) ;
}
int get_pin(int pin){
  return digitalRead(pin);
}

/*================ SETUP =====================*/

void setup () {

  USE_SERIAL.begin(9600);
  while (!USE_SERIAL); // wait for a serial connection. Needed for native USB port only

  // Connexion Wifi
  connect_wifi();
  print_network_status();
  // Choix d'une identification pour cet ESP
  whoami =  String(WiFi.macAddress());

  // Initialize the LED
  setup_led(LEDpin, OUTPUT, LOW);
  // Init temperature sensor
  TempSensor.begin();

  // Initialize SPIFFS
  SPIFFS.begin(true);
  // MQTT broker connection
  setup_mqtt_client();

  /* On se connecte avant le subscribe */
  mqtt_connect();

  /*------ Subscribe to TOPIC_LED  ---------*/
  mqtt_subscribe((char *)(TOPIC_LED));
}

/*================= LOOP ======================*/

void loop () {
  static uint32_t lasttick = 0;
  char data[100];
  String payload; // Payload : "JSON ready"
  int32_t period = 6 * 10001; // Publication period

  if ( millis() - lasttick < period) {
    goto END;
  }
  USE_SERIAL.println("End of stand by period => new sample !");
  lasttick = millis();

  // Si la connexion avec le broker tombe on se reconnecte
  mqtt_connect();
```

```
  /*------ Publish Temperature periodically ---*/
  payload = "{\"who\": \"";
  payload += whoami;
  payload += "\", \"value\": " ;
  payload += get_temperature();
  payload += "}";
  payload.toCharArray(data, (payload.length() + 1)); // Convert String payload to a char array
  // https://pubsubclient.knolleary.net/api#publish
  if (client.publish(TOPIC_TEMP, data)== true) // publish it
     USE_SERIAL.printf("published %s on topic %s\n", data, TOPIC_TEMP);

  /*------ Publish Light periodically ---------*/
  payload = "{\"who\": \"" + whoami + "\", \"value\": " + get_light() + "}";
  payload.toCharArray(data, (payload.length() + 1));

  if (client.publish(TOPIC_LIGHT, data)==true) // publish it
      USE_SERIAL.printf("published %s on topic %s\n", data, TOPIC_LED);;

END :
  // Process MQTT ... obligatoire une fois par loop()
  client.loop();
}
```

A partir ligne 222 : La loop ...

① L'ESP souscrit au topic LED

② Il publie sur les topics TEMP et LIGHT.

Le message (payload) respecte la notation JSON !

Dans le message, on trouve l'identification ("who" : une string / adresse MAC) et la valeur ("value" : un flottant) du capteur (associé au topic).

```
{"who":"80:7D:3A:FD:E8:E8", "value":20.94}
```

Ce schéma de communication pourrait être remis en cause ... on pourrait faire moins de publish (en fonction de la valeur de la variation !) ... sur un seul topic (en modifiant le schéma JSON) ?

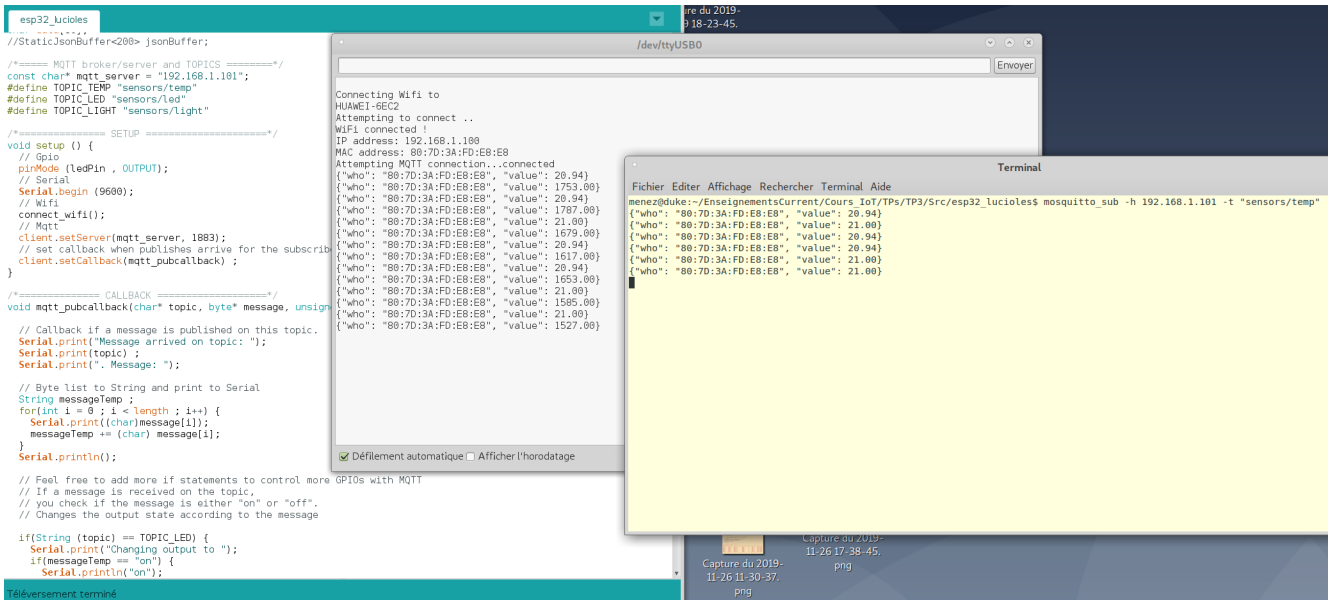
➢ Au niveau des principes, cela ne changerait pas grand chose !

➢ En terme de performances, ... faut voir !?

4.1.2 TODO !

① On lance tout ça et on vérifie que l'ESP émet régulièrement et que l'information arrive jusqu'au PC.

On fait cela avec les commandes Shell de mosquitto :



Remarque :

Sur la figure j'utilise mon broker, **mais dans le code** : "test.mosquitto.org"

② On fait évoluer le schéma JSON pour se conformer à aux schéma que l'on utilise depuis les TPS précédents.

Forcément le topic évolue aussi pour revenir au topic "uca/M1/iot".

4.2 V0 : Gestion des messages MQTT par un nodeJS server

L'objectif de cette première version est de pouvoir mémoriser les échantillons produits par l'ESP :

➢ Pour les traiter, les analyser ou pour les afficher (plot).

On va utiliser pour cela une base de données NoSQL (=¿ MongoDb) et un Node JS serveur dont le rôle sera de récupérer (en tant que subscriber) chaque message MQTT.

➢ Une fois récupérée par ce node server, l'information sera analysée, et placée si il le faut dans la base de données.

4.2.1 MongoDB : Base de données

Pour commencer, je vous laisse installer la base de données sur votre station (le cloud viendra plus loin) :

`https://docs.mongodb.com/manual/administration/install-community/`

```
cf Important on the page :

The mongodb package provided by Ubuntu is not maintained by MongoDB
Inc. and conflicts with the official mongodb-org package. ...
```

---

du coup je prends l'"official" :
`https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/`

Ensuite .. il y a de la documentation !

✔ `https://docs.mongodb.com/`

4.2.2 Node.js : Javascript côté serveur

Je vous laisse installer "nodejs" sur votre station (si ce n'est pas déjà fait ?):

`https://nodejs.org/en/download/`

et "npm" son gestionnaire de paquetages/modules (fait en même temps ?)

Pour Ubuntu :

`https://github.com/nodesource/distributions/blob/master/README.md`

On aura besoin de quelques modules JS supplémentaires :

✔ `http://expressjs.com/` ... bien pratique pour gérer les routes

npm install -g express

(g pour global soit pour toutes les app js)

✔ `https://www.npmjs.com/package/mqtt#install` ... pour que notre serveur parle mqtt.

npm install mqtt

✔ `https://www.npmjs.com/package/mongodb` ... pour pouvoir s'interfacer avec la bd.

npm install mongodb

4.3 node_v0.js

```
// Importation des modules
var path = require('path');

// var, const, let :
// https://medium.com/@vincent.bocquet/var-let-const-en-js-quelles-diff%C3%A9rences-b0f14caa2049

//--- MQTT module
const mqtt = require('mqtt')
// Topics MQTT
const TOPIC_LIGHT = 'uca/M1/iot/light'
const TOPIC_TEMP  = 'uca/M1/iot/temp'

//---  The MongoDB module exports MongoClient,
const {MongoClient} = require('mongodb');
// and that's what we'll use to connect to a MongoDB database.
// We can use an instance of MongoClient to connect to a cluster,
// access the database in that cluster, and close the connection to that cluster.
// syntaxiquement : https://stackoverflow.com/questions/33798717/javascript-es6-const-with-curly-braces

//-------------------------------------------------------------------
// This function will retrieve a list of databases in our cluster and
// print the results in the console.
async function listDatabases(client){
    databasesList = await client.db().admin().listDatabases();

    console.log("Mongo : Databases in Cluster/Server are ");
    databasesList.databases.forEach(db => console.log(` \t \t- ${db.name}`));
};

//-------------------------------------------------------------------
// asynchronous function named main() where we will connect to our
// MongoDB cluster, call functions that query our database, and
// disconnect from our cluster.
async function main_v0(){

    const mongoName = "lucioles"                    // Nom de la base dans la cluster mongo
    const mongoUri = 'mongodb://localhost:27017/'; // connection  URI
    //const uri = 'mongodb://10.9.128.189:27017/';
    //const uri = 'mongodb+srv://menez:mettrelevotre@cluster0-x0zyf.mongodb.net/test?retryWrites=true&w=majority';

    //Now that we have our URI, we can create an instance of MongoClient.
    const mg_client = new MongoClient(mongoUri,
                                      {useNewUrlParser:true, useUnifiedTopology:true});

    // Connect to the MongoDB cluster/server
    mg_client.connect(function(err,  mg_client){
        if (err) throw err; // If connection to DB failed ...
        // else
        mg_client.db("admin").command({ ping: 1 });
        console.log("Mongo : \"mg_client\" connected successfully to server !");

        //================================================
        // Print databases in our cluster
        listDatabases(mg_client);

        //================================================
        // Get a connection to the DB "lucioles" or create
        dbo = mg_client.db(mongoName); // AUTOMATICALLY GLOBAL VARIABLE !!
        console.log("Mongo : DB \"lucioles\" connected/created !");

```

```
61          // Remove "old collections : temp and light
62          dbo.listCollections({name: "temp"}).next(function(err, collinfo) {
63              if (collinfo) { // The collection exists
64                  //console.log('Collection temp already exists');
65                  dbo.collection("temp").drop()
66              }
67          });
68          dbo.listCollections({name: "light"}).next(function(err, collinfo) {
69              if (collinfo) { // The collection exists
70                  //console.log('Collection light already exists');
71                  dbo.collection("light").drop()
72              }
73          });
74          console.log("Mongo : Collection \"temp\" and \"light\" created or erased (dropped) !");
75
76          //=============================================
77          // Connexion au broker MQTT
78          //
79          //const mqtt_url = 'mqtt://192.168.1.101:1883'
80          const mqtt_url = 'mqtt://test.mosquitto.org' // with "mqtt" protocol specification
81          var options={
82              clientId:"deathstar_base",
83              //username:"darkvador",
84              //password:"6poD2R2",
85              clean:true};
86          var client_mqtt = mqtt.connect(mqtt_url,options);
87
88          // Subscription : Successfull Connection raises a "connect" event
89          client_mqtt.on('connect', function () { // => now serveur NodeJS can subscribe to topics !
90              console.log("MQTT : Node JS connected to MQTT broker !");
91              client_mqtt.subscribe(TOPIC_LIGHT, function (err) {
92                  if (!err) {
93                      //client_mqtt.publish(TOPIC_LIGHT, 'Hello mqtt')
94                      console.log('MQTT : Node Server has subscribed to \"', TOPIC_LIGHT, '\"');
95                  }
96              })
97              client_mqtt.subscribe(TOPIC_TEMP, function (err) {
98                  if (!err) {
99                      //client_mqtt.publish(TOPIC_TEMP, 'Hello mqtt')
100                     console.log('MQTT : Node Server has subscribed to \"', TOPIC_TEMP, '\"');
101                 }
102             })
103         })
104
105         //=================================================================
106         // Callback de la reception des messages MQTT pour les topics sur lesquels on s'est inscrit.
107         // => C'est cette fonction qui alimente la BD !
108         //
109         client_mqtt.on('message', function (topic, message) {
110             console.log("\nMQTT Reception of msg on topic : ", topic.toString());
111             console.log("\tPayload : ", message.toString());
112
113             // Parsing du message supposé recu au format JSON
114             message = JSON.parse(message);
115             wh = message.who
116             val = message.value
117
118             // Debug : Gerer une liste de who pour savoir qui utilise le node server
119             let wholist = []
120             var index = wholist.findIndex(x => x.who==wh)
121             if (index === -1){
122                 wholist.push({who:wh});
```

```
123             }
124             console.log("NODE JS : who is using the node server ? =>", wholist);
125 
126             // Mise en forme de la donnee à stocker => dictionnaire
127             // Le format de la date est important => doit etre compatible avec le
128             // parsing qui sera realise par highcharts dans l'UI
129             // cf https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_tolocalestring_date_all
130             // vs https://jsfiddle.net/BlackLabel/tgahn7yu
131             // var frTime = new Date().toLocaleString("fr-FR", {timeZone: "Europe/Paris"});
132             var frTime = new Date().toLocaleString("sv-SE", {timeZone: "Europe/Paris"});
133             var new_entry = { date: frTime, // timestamp the value
134                               who: wh,      // identify ESP who provide
135                               value: val    // this value
136                             };
137 
138             // On recupere le nom basique du topic du message pour s'en servir de key
139             var key = path.parse(topic.toString()).base;
140             // Stocker le dictionnaire qui vient d'etre créé dans la BD
141             // en utilisant le nom du topic comme key de collection
142             dbo.collection(key).insertOne(new_entry, function(err, res) {
143                 if (err) throw err;
144                 console.log("\nMONGO : \t Item : ", new_entry,
145                 "\nhas been inserted in db in collection :", key);
146             });
147 
148             // Debug
149             //dbo.collection(key).find({who:wh}).sort({_id:-1}).limit(10).toArray(function(err, result) {
150                 // if (err) throw err;
151                 // console.log('get on ', key, ' for ', wh);
152                 // console.log(result);
153                 //console.log('end find');
154             // });
155 
156             // Debug : voir les collections de la DB
157             //dbo.listCollections().toArray(function(err, collInfos) {
158                 // collInfos is an array of collection info objects
159                 // that look like: { name: 'test', options: {} }
160             //        console.log("List of collections currently in DB: ", collInfos);
161             //});
162         }) // end of 'message' callback installation
163 
164         //=====================================================================
165         // Fermeture de la connexion avec la DB lorsque le NodeJS se termine.
166         //
167         process.on('exit', (code) => {
168             if (mg_client && mg_client.isConnected()) {
169                 console.log('MONGO : mongodb connection is going to be closed ! ');
170                 mg_client.close();
171             }
172         })
173 
174     });// end of MongoClient.connect
175 }// end def main_v0
176 
177 // Export it to make it available outside ... for V1 next !
178 module.exports = {main_v0};
179 
180 //=====================================================================
181 //==== Demarrage BD et MQTT ======================
182 //=====================================================================
183 main_v0().catch(console.error);
```

4.3.1 Analyse du code du node server

① La phase de connection avec la base Mongo : Lignes 34-45.

Pour l'instant cette base est "en local" ... mais ca ne va pas durer !

La fonction `connect` (API Mongo) est ici utilisée avec un callback qui recevra en second paramètre le client obtenu : ceci fait partie de la spécification de la méthode `connect` .

`https://mongodb.github.io/node-mongodb-native/3.2/api/MongoClient.html#.connect`

Il y a des côtés "intéressants" (mais perturbants) dans Javascript :

`https://nodejs.org/en/knowledge/getting-started/control-flow/what-are-callbacks/`

② Dans le callback de la fonction `connect` , on procède à la connexion et à l'installation du dialogue MQTT (inscription aux topics et callback MQTT de réception).

- ✔ L85 : connexion au broker (le même que celui utilisé par l'ESP)
- ✔ L88 : installation du callback de l'event connect du client_mqtt

  =¿ L90, L96 : Souscription immédiate aux topics qui nous intéressent : lumière et température.
- ✔ L108 : callback des messages MQTT recus.
  - ➢ On doit récupérer un message JSON dont on extrait l'identifiant de flotte et la valeur.

    A ce stade le message JSON est bien "simpliste" ! ... et aussi que ce passerait-il si le message reçu sur ce topic ne respectait pas la syntaxe attendue ?
  - ➢ L131-135 : On fabrique un dictionnaire que l'on va stocker dans la base de données.

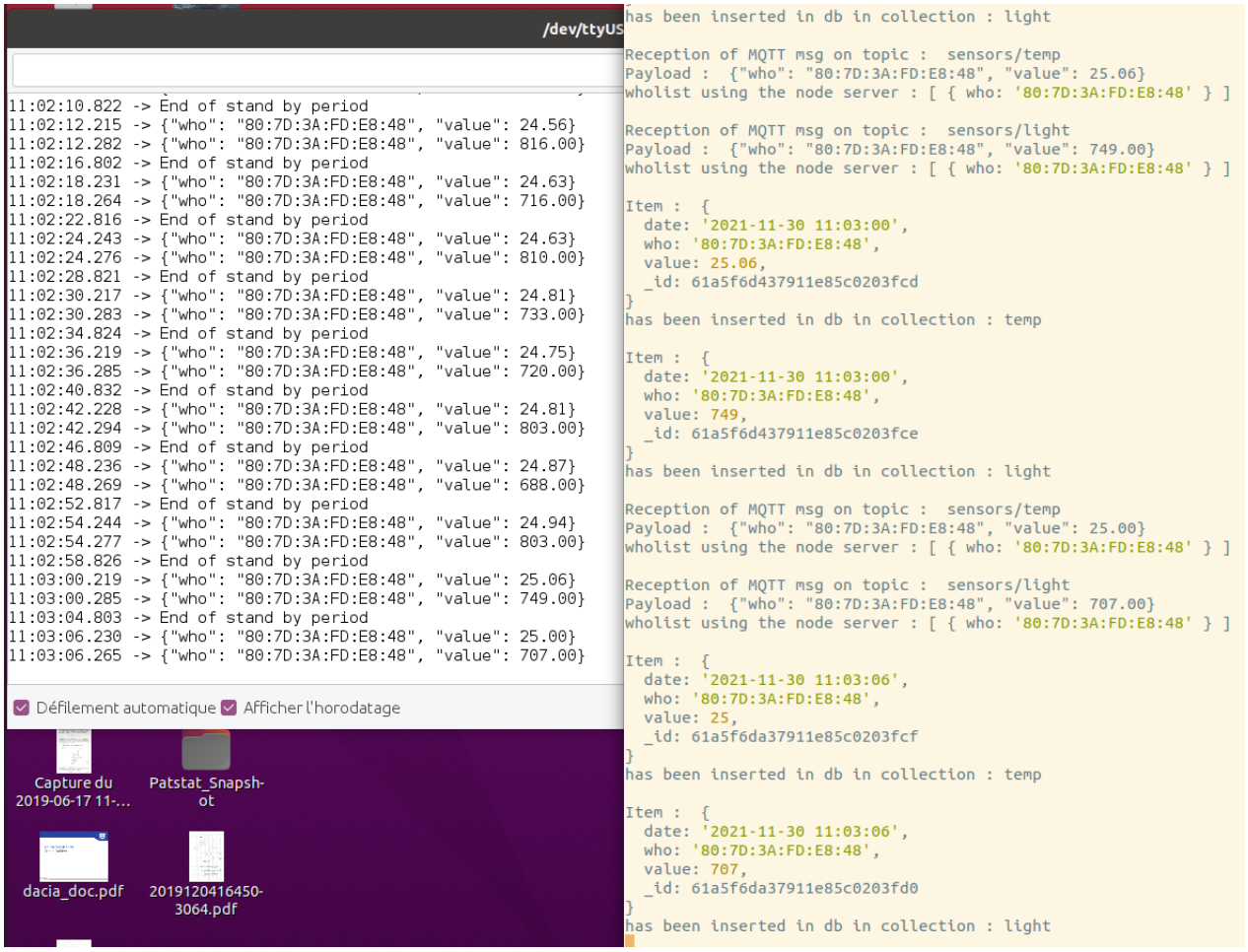
    On intègre un timestamp qui nous donne l'heure d'arrivée.
  - ➢ Le nom du topic sert de clé dans la base en désignant une collection (équivalent Mongo d'une table SQL).
    Est-ce la meilleure organisation pour la base ? c'est vous les spécialistes ? ... à vous de dire !

    Je me demande si cela ne serait pas intéressant de mettre une collection des "clients ESP" en base ? ... gestion des "logins" !
  - ➢ L141 : Insertion dans la collection sélectionnée par la key de cette nouvelle entrée.

```
MONGO :   Item :  {
  date: '2022-03-13 18:55:08',
  who: '80:7D:3A:FD:E8:E8',
  value: 24.63,
  _id: new ObjectId("622e2ffcd2900710a8243464")
}
has been inserted in db in collection : temp

MONGO :   Item :  {
  date: '2022-03-13 18:55:08',
  who: '80:7D:3A:FD:E8:E8',
  value: 1867,
  _id: new ObjectId("622e2ffcd2900710a8243465")
}
has been inserted in db in collection : light
```

4.4 Suivre le flot de données

A chaque fois qu'un message MQTT est reçu par ce Node Server, il est mis en forme et placé dans la collection qui correspond à son topic : soit "light", soit "temp".



Au niveau de la base de donnée, on peut **surveiller son évolution** depuis un terminal : c'est important de "traquer" l'information !

4.4.1 Avant démarrage du Node JS server

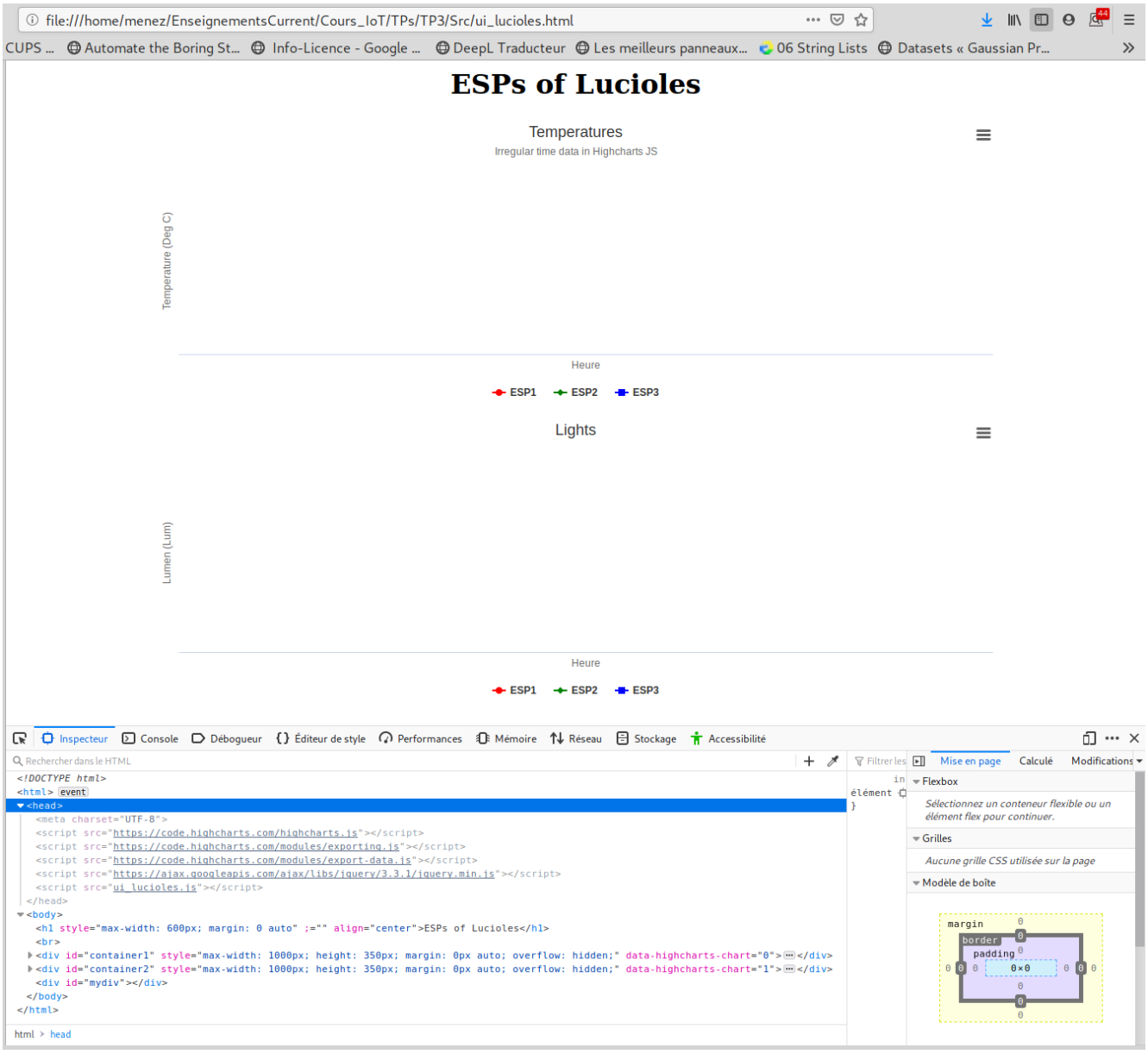
```
# Avant Demarrage du Node JS server
menez@mowgli ~
$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2020-10-06T17:00:09.742+0200 I STORAGE  [initandlisten]
2020-10-06T17:00:09.742+0200 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the Wir
2020-10-06T17:00:09.742+0200 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2020-10-06T17:00:10.861+0200 I CONTROL  [initandlisten]
2020-10-06T17:00:10.861+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-10-06T17:00:10.861+0200 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestrict
2020-10-06T17:00:10.861+0200 I CONTROL  [initandlisten]
> show dbs;
IoT       0.000GB
admin     0.000GB
config    0.000GB
local     0.000GB
lucioles  0.000GB
> use lucioles
switched to db lucioles
> show collections
>
```

4.4.2 Apres démarrage du Node JS server

```
# Après Demarrage du Node JS server
> show collections
light
temp

> db.temp.find().pretty()
{
        "_id" : ObjectId("5f848bcbb87cda30f65e3d93"),
        "date" : "2020-10-12 19:00:59",
        "who" : "80:7D:3A:FD:E8:E8",
        "value" : 24.19
}
{
        "_id" : ObjectId("5f85c8457e9b7f69573eec3e"),
        "date" : "2020-10-13 17:31:17",
        "who" : "80:7D:3A:FD:E8:E8",
        "value" : 24.12
}
{
        "_id" : ObjectId("5f85cc23a466276a0cbc116f"),
        "date" : "2020-10-13 17:47:47",
        "who" : "80:7D:3A:FD:E8:E8",
        "value" : 24.06
}

# Vider une collection
> db.temp.drop()
true
> db.temp.find().pretty()
```

4.5 V1 : Dessiner les séries temporelles

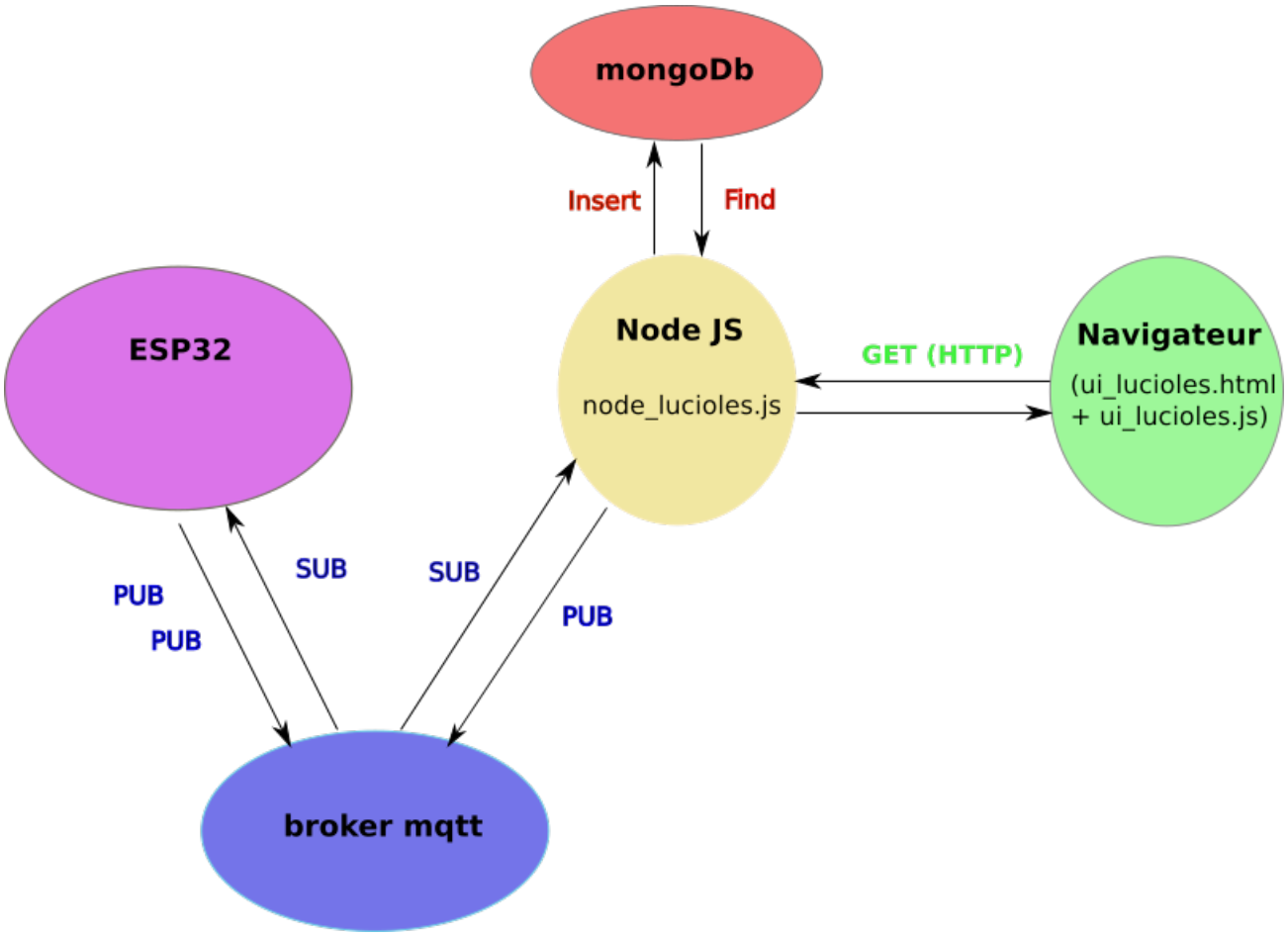
Cette deuxième phase d'évolution de l'outil vise à présenter les données récoltées et stockées dans la database au travers d'une User Interface (un dashboard) accessible par le Web (donc en NodeJS).



On "dessine" la série temporelle mais ce traitement pourrait tout à fait être remplacé par une analyse "intelligente".

4.5.1 Architecture logicielle

Forcément, l'architecture logicielle se complique un peu :

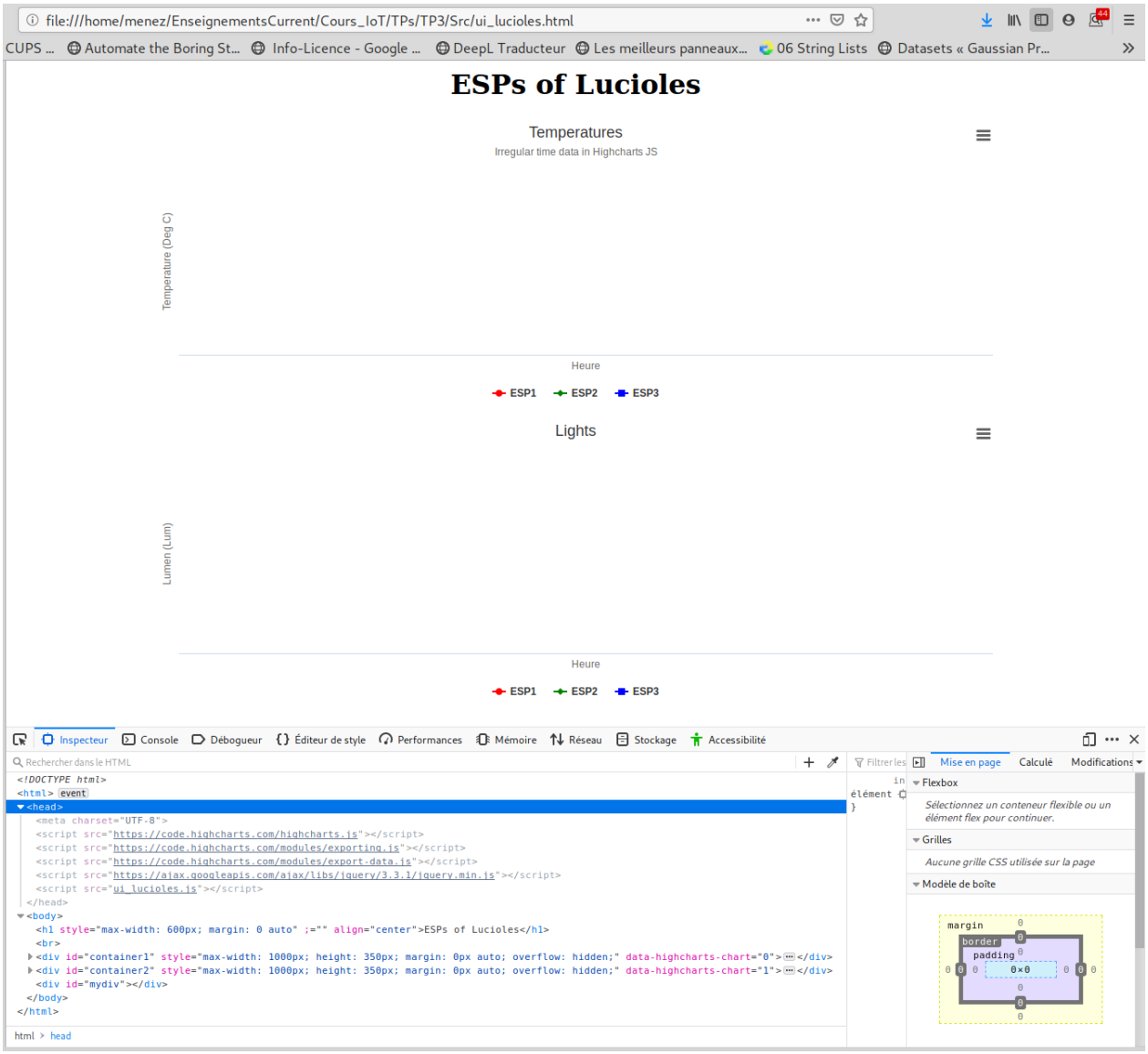


Le noeud central est le Node JS que l'on vient de commencer :

- ➢ Il récupère les informations depuis le broker.
- ➢ Il s'interface avec la base de données (MongoDb).
- ➢ Il répond aux requêtes du navigateur qui lui demande ses dernières données.

On retrouve ces fonctionnalités dans le code JS.

4.5.2 Page Web de l'UI / Dashboard



Dans le but de l'ouvrir avec un navigateur, on crée une page Web `ui_lucioles.html` qui va exploiter le package javascript `code.highcharts.com` pour dessiner des courbes dans deux containers :

- ✔ "container1" pour les températures.
- ✔ "container2" pour les luminosités.

On utilise le package `code.highcharts.com` mais il y a d'autres alternatives ...

`https://www.amcharts.com/`

`ui_lucioles.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script src="https://code.highcharts.com/highcharts.js"></script>
    <script src="https://code.highcharts.com/modules/exporting.js"></script>
    <script src="https://code.highcharts.com/modules/export-data.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="ui_lucioles.js"></script>
  </head>
  <body>
    <h1 style="max-width: 600px; margin: 0 auto" ; align="center">ESPs of Lucioles</h1>

    <br>
    <div id="container1" style="max-width: 1000px; height: 350px; margin: 0 auto"></div>
    <div id="container2" style="max-width: 1000px; height: 350px; margin: 0 auto"></div>
    <div id="mydiv"></div>
  </body>
</html>
```

Juste après avoir chargé les codes JS des graphes, cette page HTML charge un Javascript `ui_lucioles.js` (ligne 9) .

➢ Ce script doit permettre de faire le lien avec le Node serveur que l'on a débuté dans la V0.

`ui_lucioles.js` :

```
1  //
2  // Cote UI de l'application "lucioles"
3  //
4  // Auteur : G.MENEZ
5  // RMQ : Manipulation naive (debutant) de Javascript
6  //
7
8  function init() {
9      //=== Initialisation des traces/charts de la page html ===
10     // Apply time settings globally
11     Highcharts.setOptions({
12         global: { // https://stackoverflow.com/questions/13077518/highstock-chart-offsets-dates-for-no-reason
13             useUTC: false,
14             type: 'spline'
15         },
16         time: {timezone: 'Europe/Paris'}
17     });
18     // cf https://jsfiddle.net/gh/get/library/pure/highcharts/highcharts/tree/master/samples/highcharts/demo/spline-irregular-time,
19     chart1 = new Highcharts.Chart({
20         title: {text: 'Temperatures'},
21         subtitle: { text: 'Irregular time data in Highcharts JS'},
22         legend: {enabled: true},
23         credits: false,
24         chart: {renderTo: 'container1'},
25         xAxis: {title: {text: 'Heure'}, type: 'datetime'},
26         yAxis: {title: {text: 'Temperature (Deg C)'}},
27         series: [{name: 'ESP1', data: []},
28                  {name: 'ESP2', data: []},
29                  {name: 'ESP3', data: []}],
30         //colors: ['#6CF', '#39F', '#06C', '#036', '#000'],
31         colors: ['red', 'green', 'blue'],
```

```
32          plotOptions: {line: {dataLabels: {enabled: true},
33                               //color: "red",
34                               enableMouseTracking: true
35                              }
36                      }
37      });
38      chart2 = new Highcharts.Chart({
39          title: { text: 'Lights'},
40          legend: {title: {text: 'Lights'}, enabled: true},
41          credits: false,
42          chart: {renderTo: 'container2'},
43          xAxis: {title: {text: 'Heure'},type: 'datetime'},
44          yAxis: {title: {text: 'Lumen (Lum)'}},
45          series: [{name: 'ESP1', data: []},
46                   {name: 'ESP2', data: []},
47                   {name: 'ESP3', data: []}],
48          //colors: ['#6CF', '#39F', '#06C', '#036', '#000'],
49          colors: ['red', 'green', 'blue'],
50          plotOptions: {line: {dataLabels: {enabled: true},
51                               enableMouseTracking: true
52                              }
53                      }
54      });
55
56      //=== Gestion de la flotte d'ESP ================================
57      var which_esps = [ // attention au code des : ?
58          "80:7D:3A:FD:E8:E8"
59  //        ,"1761716416"
60  //        "80:7D:3A:FD:C9:44"
61      ]
62
63      for (var i = 0; i < which_esps.length; i++) {
64          process_esp(which_esps, i)
65      }
66  };
67
68
69  //=== Installation de la periodicite des requetes GET============
70  function process_esp(which_esps,i){
71      const refreshT = 10000 // Refresh period for chart
72      esp = which_esps[i];    // L'ESP "a dessiner"
73      //console.log(esp) // cf console du navigateur
74
75      // Gestion de la temperature
76      // premier appel pour eviter de devoir attendre RefreshT
77      get_samples('/esp/temp', chart1.series[i], esp);
78      //calls a function or evaluates an expression at specified
79      //intervals (in milliseconds).
80      window.setInterval(get_samples,
81                         refreshT,
82                         '/esp/temp',     // param 1 for get_samples()
83                         chart1.series[i],// param 2 for get_samples()
84                         esp);            // param 3 for get_samples()
85
86      // Gestion de la lumiere
87      get_samples('/esp/light', chart2.series[i], esp);
88      window.setInterval(get_samples,
89                         refreshT,
90                         '/esp/light',     // URL to GET
91                         chart2.series[i], // Serie to fill
92                         esp);             // ESP targeted
93  }
```

```
94
95
96  //=== Recuperation dans le Node JS server des samples de l'ESP et
97  //=== Alimentation des charts ==================================
98  function get_samples(path_on_node, serie, wh){
99      // path_on_node => help to compose url to get on Js node
100     // serie => for choosing chart/serie on the page
101     // wh => which esp do we want to query data
102
103     node_url = 'http://127.0.0.1:3000';
104     //node_url = 'http://134.59.131.45:3000'
105     //node_url = 'http://192.168.1.101:3000'
106
107     //https://openclassrooms.com/fr/courses/1567926-un-site-web-dynamique-avec-jquery/1569648-le-fonctionnement-de-ajax
108     $.ajax({
109         url: node_url.concat(path_on_node), // URL to "GET" : /esp/temp ou /esp/light
110         type: 'GET',
111         headers: { Accept: "application/json", },
112         data: {"who": wh}, // parameter of the GET request
113         success: function (resultat, statut) { // Anonymous function on success
114             let listeData = [];
115             resultat.forEach(function (element) {
116                 listeData.push([Date.parse(element.date),element.value]);
117                 //listeData.push([Date.now(),element.value]);
118             });
119             serie.setData(listeData); //serie.redraw();
120         },
121         error: function (resultat, statut, erreur) {
122         },
123         complete: function (resultat, statut) {
124         }
125     });
126 }
127
128
129 //assigns the onload event to the function init.
130 //=> When the onload event fires, the init function will be run.
131 window.onload = init;
```

4.5.3 Analyse du code

① La fonction d'init (qui sera invoquée au chargement de la page L131) met en place les graphes ET met en place des requêtes GET périodiques pour certains objets énumérés L57.

➢ L'idée : les graphes demandent au serveur Node les données concernant ces objets de la flotte.

C'est le rôle de la fonction "process_esp()".

Puis le node serveur demandera à la base ces informations.

② La fonction `process_esp()` met en place des requêtes périodiques (fonction "get_samples()") au serveur Node pour les différents éléments de la flotte.

Ces requêtes GET se différencient par l'ESP et par la route/url concernées : L107 et L117.

③ La fonction `get_samples()` réalise une requête GET sur le node serveur dont l'IP est fournie (un peu trop) en dur.

Cette requête concerne un ESP dont l'identité est formulée dans le champ "data" donné qui figurera en paramètre de la requête.

Cette requête utilise AJAX (Asynchronous JavaScript and XML).

- ➢ AJAX is a technique for creating fast and dynamic web pages.
- ➢ AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.

  This means that **it is possible to update parts of a web page, without reloading the whole page**.
- ➢ Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

`https://www.w3schools.com/js/js_ajax_intro.asp`

Lorsque la requête réussi, la liste de valeurs récupérée est placée (L116) dans la série (en paramètre).

4.6 Evolution du nodeJS server

On doit donc compléter le code du Node serveur pour gérer les requêtes en provenance du client/navigateur Web :

➢ Le module `node_v0.js` est augmenté avec l'utilisation du framework `express` pour donner le module `node_v1.js`

Express.js est un framework pour construire des applications web basées sur Node.js.

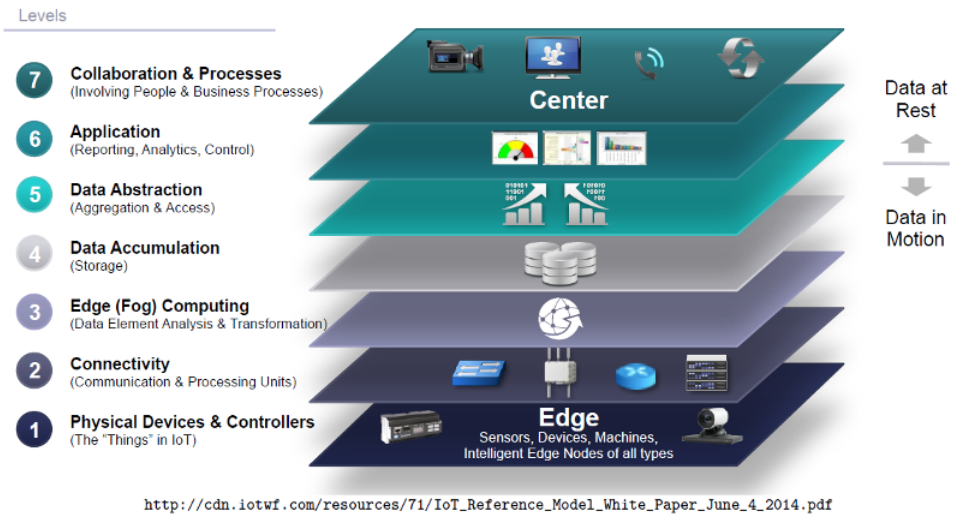
➢ C'est de fait le "framework standard" pour le développement de serveur en Node.js

```
// Importation des modules
var path = require('path');

//=====================================================================
//==== Demarrage BD et MQTT ======================
//=====================================================================
const {main_v0} = require('../AppJS_V0/node_v0');

//====================================
// Utilisation du framework express pour gérér les routes
const express = require('express');
const app = express();

// et pour permettre de parcourir les body des requetes
const bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())
app.use(express.static(path.join(__dirname, '/')));
app.use(function(request, response, next) { //Pour eviter les problemes de CORS/REST
    response.header("Access-Control-Allow-Origin", "*");
    response.header("Access-Control-Allow-Headers", "*");
    response.header("Access-Control-Allow-Methods", "POST, GET, OPTIONS, PUT, DELETE");
    next();
});

//=====================================================================
// Answering GET request on this node ... probably from navigator.
// => REQUETES HTTP reconnues par le Node
//=====================================================================

// Route / => Le node renvoie la page HTML affichant les charts
app.get('/', function (req, res) {
    res.sendFile(path.join(__dirname + '/ui_lucioles.html'));
});


// The request contains the name of the targeted ESP !
//     /esp/temp?who=80%3A7D%3A3A%3AFD%3AC9%3A44
// Exemple d'utilisation de routes dynamiques
//    => meme fonction pour /esp/temp et /esp/light
app.get('/esp/:what', function (req, res) {
    // cf https://stackabuse.com/get-query-strings-and-parameters-in-express-js/
    console.log(req.originalUrl);

    wh = req.query.who // get the "who" param from GET request
    // => gives the Id of the ESP we look for in the db
    wa = req.params.what // get the "what" from the GET request : temp or light ?
```

```
    console.log("\n--------------------------------");
    console.log("A client/navigator ", req.ip);
    console.log("sending URL ",  req.originalUrl);
    console.log("wants to GET ", wa);
    console.log("values from object ", wh);

    // Récupération des nb derniers samples stockés dans
    // la collection associée a ce topic (wa) et a cet ESP (wh)
    const nb = 100;
    key = wa;
    //dbo.collection(key).find({who:wh}).toArray(function(err,result) {
    dbo.collection(key).find({who:wh}).sort({_id:-1}).limit(nb).toArray(function(err, result) {
        if (err) throw err;
        console.log('get on ', key, ' for ', wh);
        console.log(result);
        res.json(result.reverse()); // This is the response.
        console.log('end find');
    });
    console.log('end app.get');
});

//=================================================================
//==== Demarrage du serveur Web  ======================
//=================================================================
// L'application est accessible sur le port 3000

app.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

5 TOTRY

Cette application est un exemple de ce que pourrait être une ”application IoT” :



http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

Elle met en place des traitements sur les différents niveaux du modèle d'architecture d'applications IoT :

- ✔ sur la couche physique (gestion de l'objet),
- ✔ sur les couches Fog et Data accumulation (analyse de message et gestion de la base de donnée)
- ✔ sur la couche application (reporting graphique)

Fonctionnellement c'est un point de départ ... les évolutions viendront plus loin ... ou pas selon le temps que vous mettez ;-)

Le premier effort porte sur la compréhension de l'application ... quelques serveurs et plusieurs machines (ESP, Host, Navigateur).

Ensuite il faut déployer pour faire marcher !

5.1 TODO : Déploiement ”local”

Dans un premier temps, en considérant que tous les acteurs (clients et serveurs) de cette application sont locaux il faut donc :

① Comprendre,

② et faire tourner.