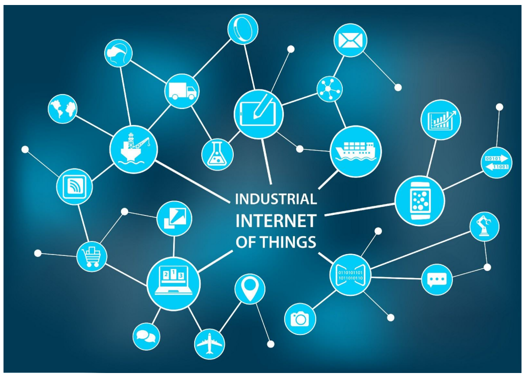
Objets connectés et Données distribuées

génération, transmission, traitement et analyse

F. Morain-Nicolier

S5 BUT GEII - IUT Troyes

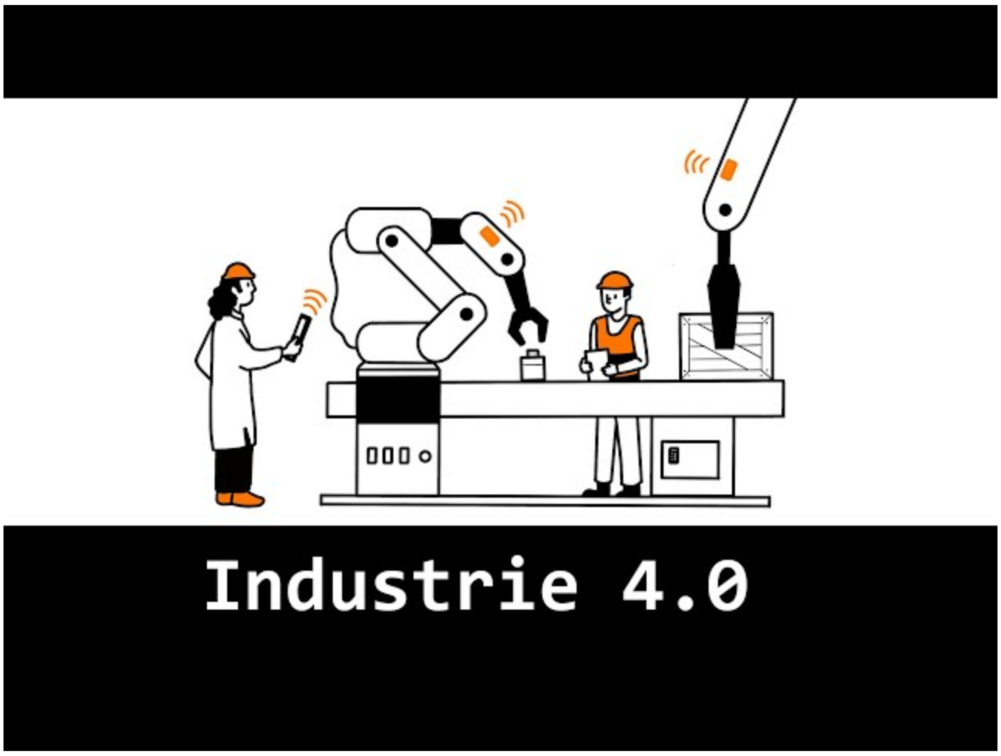


Introduction / Contexte

Industrie 4.0 / IIoT

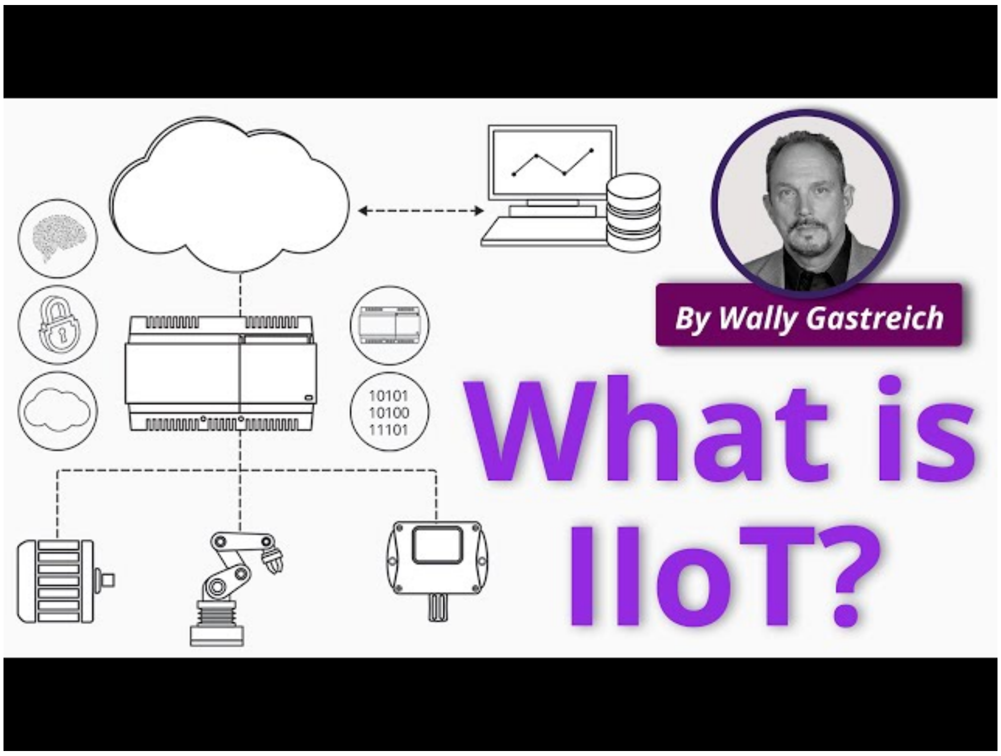
–

[Industrie 4.0 : comprendre l'essentiel en 5 minutes]

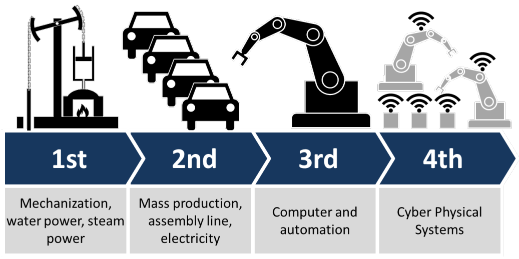


–

[What is the Industrial Internet of Things (IIoT)?]



Industrie 4.0 / IIoT : mots clés en vrac


1st
Mechanization, water power, steam power
2nd
Mass production, assembly line, electricity
3rd
Computer and automation
4th
Cyber Physical Systems

– Systèmes cyberphysiques
– Cobot
– Jumeau numérique
– Capteurs
– IoT ⇔ IIot
– *Big Data / Data Mining*
– Maintenance prédictive
– *Cloud Computing*
– Réalité virtuelle/augmentée
– Fabrication (additive)
– Cybersécurité
– *Machine Learning* ⇔ « IA » (?)

Objectifs généraux du cours

– Comment
  – acquérir des données distribuées (d'objets iot ou de serveurs),
  – les analyser (détecter),
  – les traiter (*ie* les modifier),
  –

  et les redistribuer ?

  ⇒ tout ceci en temps réel.

Programme S5

- – Contexte objets connectés
- – Présentation générale des outils
- – Envoyer/recevoir des données
- – Détection d'anomalies (temps réel)
- – Filtre de données (temps réel)

Programme S6 (estimation)

–

concevoir un capteur IoT : capteurs + SBC

–

appareils Zigbee

–

Tableau de bord/Supervision : HomeAssistant (visualisation / scénarios)

–

lien avec base de données

- – approfondissement : Traitement et analyse de données
  - – Multicapteurs
  - – Apprentissage Artificiel : classification supervisée / non-supervisée (clustering)

Présentation générale des outils

- – MQTT
- – Javascript ⇒ Node.js
- – NodeRed

1. MQTT

Message Queuing Telemetry Transport

MQTT is suitable for

- Asset tracking and management
- Automotive telematics
- Chemical detection
- Environment and traffic monitoring
- Field force automation
- Fire and gas testing
- Home automation
- In-Vehicle Infotainment (IVI)
- Medical
- Messaging
- Point of Sale (POS) kiosks
- Railway Radio-Frequency Identification (RFID)
- Supervisory Control and Data Acquisition (SCADA)
- Slot machines

MQTT was designed to be suitable to support the following typical challenges in IoT

- Be **lightweight** to make it possible to **transmit high volumes of data** without huge overheads

–

Distribute minimal packets of data in huge volumes

–

Support an **event-oriented** paradigm with asynchronous bidirectional low latency push delivery of messages

–

Easily emit data from one client to many clients

–

Make it possible to listen for events whenever they happen (event-oriented architecture)

–

Support always-connected and sometimes-connected models

–

Publish information over **unreliable networks** and provide **reliable deliveries over fragile connections**

–

Work very well with battery-powered devices or **require low power consumption**

–

Provide responsiveness to make it possible to **achieve near real-time delivery of information**

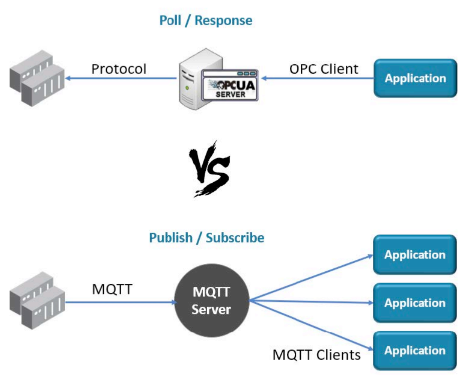
–

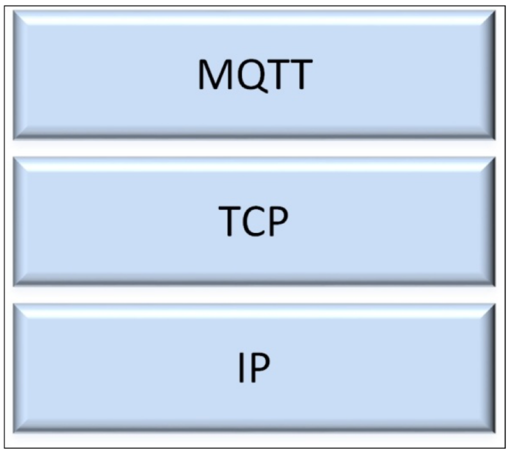
Offer **security and privacy** for all the data

–

Be able to provide the necessary **scalability** to distribute data to hundreds of thousands of clients

Principe général

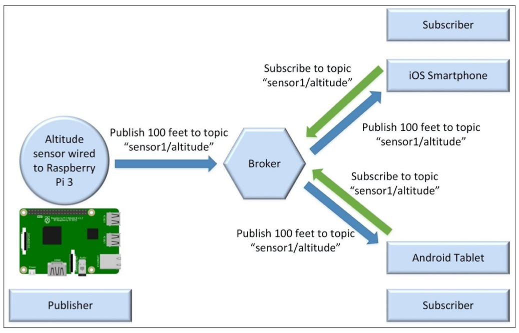




– MQTT permet concrètement aux appareils d'envoyer des informations sur un sujet donné à un serveur qui fonctionne comme un broker de messages.
– Le broker pousse ces informations vers les clients qui se sont précédemment abonnés.

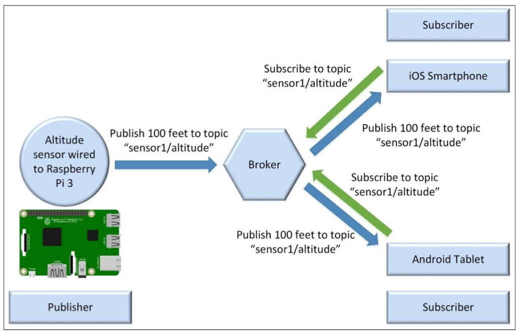
– runs on top of Transmission Control Protocol / Internet Protocol (TCP/IP).

Modèle = publication ↔ abonnement

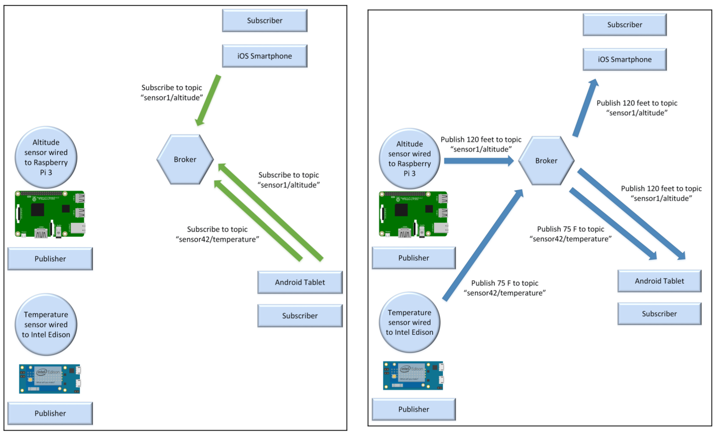

Subscriber
iOS Smartphone
Subscribe to topic "sensor1/altitude"
Altitude sensor wired to Raspberry Pi 3
Publish 100 feet to topic "sensor1/altitude"
Broker
Publish 100 feet to topic "sensor1/altitude"
Subscribe to topic "sensor1/altitude"
Publish 100 feet to topic "sensor1/altitude"
Android Tablet
Publisher
Subscriber

– The publish-subscribe pattern requires a **broker**, also known as server.
– All the clients establish a connection with the broker. The client that sends a message through the broker is known as the **publisher**.
– The broker filters the incoming messages and distributes them to the clients that are interested in the type of received messages.
– The clients that register to the broker as interested in specific types of messages are known as **subscribers**.
– Hence, both publishers and subscribers establish a connection with the broker.

Modèle = publication ↔ abonnement


Subscriber
iOS Smartphone
Subscribe to topic "sensor1/altitude"
Altitude sensor wired to Raspberry Pi 3
Publish 100 feet to topic "sensor1/altitude"
Broker
Publish 100 feet to topic "sensor1/altitude"
Subscribe to topic "sensor1/altitude"
Publish 100 feet to topic "sensor1/altitude"
Android Tablet
Publisher
Subscriber

– A **topic** is a named logical channel and it is also referred to as a channel or subject. The broker will send publishers only the messages published to topics to which they are subscribed.
– The data for a message is known as **payload**. A message includes the topic to which it belongs and the payload.
– In most cases, we will want to take advantage of **asynchronous operations**.


Subscriber
iOS Smartphone
Subscribe to topic "sensor1/altitude"
Altitude sensor wired to Raspberry Pi 3
Broker
Subscribe to topic "sensor1/altitude"
Subscribe to topic "sensor42/temperature"
Publisher
Android Tablet
Subscriber
Temperature sensor wired to Intel Edison
Publisher
Subscriber
iOS Smartphone
Publish 120 feet to topic "sensor1/altitude"
Altitude sensor wired to Raspberry Pi 3
Publish 120 feet to topic "sensor1/altitude"
Broker
Publish 120 feet to topic "sensor1/altitude"
Publish 75 F to topic "sensor42/temperature"
Publisher
Android Tablet
Subscriber
Publish 75 F to topic "sensor42/temperature"
Temperature sensor wired to Intel Edison
Publisher

Asynchronous operations

Fonctionnement

Une session MQTT est divisée en quatre étapes :

1. connexion
2. authentification
3. communication
4. terminaison

Ports standarts :

– 1883 pour la communication non chiffrée

– 8883 pour la communication chiffrée utilisant SSL/TLS.

Authentification

–

MQTT est destiné aux appareils disposant de ressources limitées

⇒ protocole léger : faible empreinte logicielle des messages

– en-tête fixe (2 octets)

– en-tête variable facultatif

– charge utile de message limitée à 256 Mo

– niveau de qualité de service

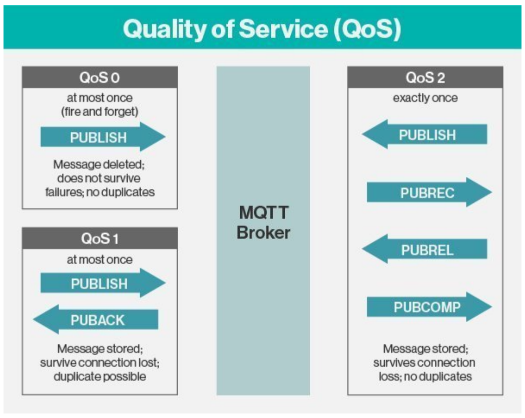
–

SSL/TLS pas toujours disponible ou souhaité.Le client s'authentifie alors en envoyant un nom d'utilisateur et un mot de passe en clair au serveur.

–

Certains *brokers* acceptent les clients anonymes. Dans ce cas, le nom d'utilisateur et le mot de passe sont tout simplement laissés vides.

QoS : qualité de service


Quality of Service (QoS)
QoS 0
at most once
(fire and forget)
PUBLISH
Message deleted; does not survive failures; no duplicates
QoS 1
at most once
PUBLISH
PUBACK
Message stored; survive connection lost; duplicate possible
MQTT Broker
QoS 2
exactly once
PUBLISH
PUBREC
PUBREL
PUBCOMP
Message stored; survives connection loss; no duplicates

Topics

Les chaînes décrivant un sujet forment une arborescence en utilisant la barre oblique (`/`) comme caractère de séparation.

Un client peut s'abonner à des branches entières de l'arborescence d'un sujet (ou se désabonner) à l'aide de

wildcards :

– Le signe `#` remplace n'importe quel nombre de niveaux de topic

– Le signe `+` remplace seulement un niveau de topic
– Exemples :

```
mygreenhouse/sensors/#
+/sensors/temperature
```

Topics : bonnes pratiques
Organisation des Topics :
– `sensorID/temp` plutôt que `temp/sensorID`
– Pas besoin de commencer par un `/`
Exemple :

```
elec/general/courant/i1
elec/mmi/courant/i1
elec/geii/courant/i1
elec/geii/courant/i2
```

En pratique ⇒ Démonstration rapide

- *Brokers* utiles
  - `https://lp-iot.cloud.shiftr.io`
  - `10.98.35.245` sur l'IUT
- MQTT Explorer
- Éxécutables à utiliser (mosquitto)
  - `mosquitto_sub`
  - `mosquitto_pub`
  - `mosquitto` : lancement d'un broker
    (lancé par défaut au démarrage de W10 sur les postes)

2. Javascript

(cf *Learn X in Y minutes*[1])

JavaScript was **created by Netscape's Brendan Eich in 1995**. It was **originally intended as a simpler scripting language for websites**.

JavaScript **isn't just limited to web browsers, though: Node.js**, a project that provides a standalone runtime for Google Chrome's V8 JavaScript engine, is becoming more and more popular.

JavaScript has a ** C-like syntax**, JavaScript's object model is significantly different to Java's.

1. https://learnxinyminutes.com/docs/javascript/

Généralités

– paradigme impératif, fonctionnel et objet (orienté objet à prototype)

– toutes les expressions (identifiants, littéraux et opérateurs et leurs opérandes) sont de **type référence**

– **Attention** fin de ligne = fin d'instruction

```
return
    true;
```

est compris comme deux instructions :

```
return;
true;
```

Spécifités

Comparaisons

```
// Equality is ===
1 === 1; // = true
2 === 1; // = false
// Inequality is !==
1 !== 1; // = false
2 !== 1; // = true
```

Strings

```
// Strings are concatenated with +
"Hello " + "world!"; // = "Hello world!"
"1, 2, " + 3; // = "1, 2, 3"
"Hello " + ["world", "!"]; // = "Hello world,!"
// ...which can result in some weird behaviour...
13 + !0; // 14
"13" + !0; // '13true'
// and are compared with < and >
"a" < "b"; // = true
```

Type coercion

```
// Type coercion is performed for comparisons with double equals...
"5" == 5; // = true
null == undefined; // = true
// ...unless you use ===
"5" === 5; // = false
null === undefined; // = false
```

Variables

```
// Variables are declared with the `var` keyword. JavaScript is dynamically
typed, so you don't need to specify type. Assignment uses a single `=` cha-
racter.
var someVar = 5;

// If you leave the var keyword off, you won't get an error...
someOtherVar = 10;
// ...but your variable will be created in the global scope, not in the
scope you defined it in.

// Variables declared without being assigned to are set to undefined.
var someThirdVar; // = undefined
```

Arrrays

```
// Arrays are ordered lists of values, of any type.
var myArray = ["Hello", 45, true];

// Their members can be accessed using the square-brackets subscript syntax.
// Array indices start at zero.
myArray[1]; // = 45
```

```
// Arrays are mutable and of variable length.
myArray.push("World");
myArray.length; // = 4

// Add/Modify at specific index
myArray[3] = "Hello";
```

Objets

```
// JavaScript's objects are equivalent to "dictionaries" or "maps" in other
languages: an unordered collection of key-value pairs.
var myObj = {key1: "Hello", key2: "World"};

// Keys are strings, but quotes aren't required if they're a valid Java-
Script identifier. Values can be any type.
var myObj = {myKey: "myValue", "my other key": 4};

// Object attributes can also be accessed using the subscript syntax,
myObj["my other key"]; // = 4

// ... or using the dot syntax, provided the key is a valid identifier.
myObj.myKey; // = "myValue"

// Objects are mutable; values can be changed and new keys added.
myObj.myThirdKey = true;

// If you try to access a value that's not yet set, you'll get undefined.
myObj.myFourthKey; // = undefined
```

Fonctions

```
// JavaScript functions are declared with the `function` keyword.
function myFunction(thing){
    return thing.toUpperCase();
}
myFunction("foo"); // = "FOO"

// JavaScript functions are **first class objects**, so they can be reassi-
gned to different variable names and passed to other functions as arguments
- for example, when supplying an event handler:
function myFunction(){
    // this code will be called every 5 seconds
}
setInterval(myFunction, 5000);
// Note: setInterval isn't part of the JS language, but is provided by brow-
sers and Node.js.
```

Fonctions lambda

```
// Function objects don't even have to be declared with a name - you can
write an anonymous function definition directly into the arguments of ano-
ther.
setTimeout(function(){
    // this code will be called in 5 seconds' time
}, 5000);

// ou encore
var fx = function() {
    ...
}
setTimeout(fx, 5000);
```

```
// Fermetures
function f(x) {
    return function (y) {
        return x+y;
    }
}
var z = f(10);
console.log(z(1)); // = 11
```

Portée lexicale

```
// consider the var statement:
var myName = "Kyle";
var age;
// Another similar keyword is let, allowing a more limited access to the va-
riable than var
let myName = "Kyle";
let age;
```

```
var adult = true;
```

```
if (adult) {
    var myName = "Kyle";
    let age = 39;
    console.log("Shhh, this is a secret!");
}

console.log(myName);
// Kyle

console.log(age);
// Error!
```

Objets

```
// Objects can contain functions.
myObj = {
    myString: "Hello world!",
    myFunc: function(){
        return this.myString;
    }
};
myObj.myFunc(); // = "Hello world!"
```

Constructeurs

```
// When you call a function with the `new` keyword, a new object is created,
and
// made available to the function via the `this` keyword. Functions designed
to be
// called like that are called constructors.

var MyConstructor = function(){
    this.myNumber = 5;
};
myNewObj = new MyConstructor(); // = {myNumber: 5}
myNewObj.myNumber; // = 5
```

⇒ « vraie » POO

```
class Publication {
    constructor(title,author,pubDate) {
        this.title = title;
        this.author = author;
        this.pubDate = pubDate;
    }

    print() {
        console.log(`
            Title: ${ this.title }
            By: ${ this.author }
            ${ this.pubDate }
        `);
    }
}
```

Héritage

```
class Book extends Publication {
    constructor(bookDetails) {
        super(
            bookDetails.title,
            bookDetails.author,
            bookDetails.publishedOn
        );
        this.publisher = bookDetails.publisher;
        this.ISBN = bookDetails.ISBN;
    }

    print() {
        super.print();
        console.log(`
            Publisher: ${ this.publisher }
            ISBN: ${ this.ISBN }
        `);
    }
}
```

Pour finir sur Javascript

– Javascript par l’exemple : Learn X in Y minutes Where X=javascript
– Un bon résumé du langage : You Don't Know JS Yet: Get Started - 2nd Edition
– et le chapitre 3 : You Don't Know JS Yet: Get Started - 2nd Edition pour quelques éléments plus poussés
–
Attention à la coercition de type (transtypage implicite)
⇒ Typescript

– `"use strict";` ⇒ garde-fou (à mettre en première ligne)

```
"use strict";
x = 3.14;       // This will cause an error because x is not declared
```

Humour :

Wat, A lightning talk by Gary Bernhardt from CodeMash 2012

3.Node.js

–
Wikipedia[1] :

Node.js est une plateforme logicielle libre en JavaScript, orientée vers les **applications réseau évènementielles hautement concurrentes qui doivent pouvoir monter en charge**.

Concrètement, Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur.
permet de traiter du code JavaScript asynchrone ⇒ Programmation événementielle (cf Signal/Slot en C++/Qt)
– Très bon site web : https://nodejs.org en particulier https://nodejs.org/en/learn

---

1. https://fr.wikipedia.org/wiki/Node.js

Généralités

– A Node.js app **runs in a single process**, without creating a new thread for every request

– set of **asynchronous I/O primitives** in its standard library that **prevent JavaScript code from blocking**
  – When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back
– allows to handle **thousands of concurrent connections with a single server** without introducing the burden of managing thread concurrency

Un exemple

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

–
sauvegarder le fichier `server.js`
–
exécution : `node server.js`
–
first includes the Node.js `http` module.
–
`createServer()` method of `http creates a new HTTP server and returns it.
–
The server is set to listen on the specified port and host name
–
When the server is ready, the callback function is called, in this case informing us that the server is running.

npm[1] = Node.js Package Manager

– Three distinct components:
  – website
  – Command Line Interface (CLI)

– registry

Use the **website** to discover packages, set up profiles, and manage other aspects of your npm experience. For example, you can set up organizations to manage access to public or private packages.

The **CLI** runs from a terminal, and is how most developers interact with npm.

The **registry** is a large public database of JavaScript software and the metainformation surrounding it.

---

1. https://www.npmjs.com/

MQTT

Librairie `mqtt`[1] sur npm :

```
const mqtt = require("mqtt");
const client = mqtt.connect("mqtt://test.mosquitto.org");

client.on("connect", () => {
  client.subscribe("presence", (err) => {
    if (!err) {
      client.publish("presence", "Hello mqtt");
    }
  });
});

client.on("message", (topic, message) => {
  // message is Buffer
  console.log(message.toString());
  client.end();
});
```

Sortie console :

```
Hello mqtt
```

Fourniture d'exécutables en ligne de commande

install MQTT.js globally

```
npm install mqtt -g
```

Then

```
mqtt sub -t 'hello' -h 'test.mosquitto.org' -v
```

and

```
mqtt pub -t 'hello' -h 'test.mosquitto.org' -m 'from MQTT.js'
```

---

1. https://www.npmjs.com/package/mqtt

4. NodeRed

– Comment injecter du code Node.js dans NodeRed ?