Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

MASTER 2 SSIO UPEM

Web services

Node.js, API REST, MQTT, Broker servers

Samir Bellahsene
IOT & eSIM Technology Manager at Orange/INNOV/D&P
samir.bellahsene@orange.com

December 2022

Contents

#1 Introduction

3. Introduction

- Frontend vs backend development == > we have seen Angular, Typescript, React and React Native for front-end
- We have also used Node.js during the frontend courses.
- We will focus on Node.js for Backend.
- Basics on API REST and MQTT with focus on IOT implementations.

#2 Node.js

2.1. What's Node.js

- ❑ Node.js is an open-source and cross-platform JavaScript runtime environment that let building performant servers
- ❑ Advantage in the possibility of coding the server-side in JavaScript. No need double languages for building an end-to-end (E2E) application.
- ❑ Runs the V8 JavaScript engine outside of the browser (V8 is the core of Google Chrome).
- ❑ A Node.js app run in a single process, without creating a new thread for every request.
- ❑ Non-blocking paradigms.
- ❑ Asynchronous I/O primitives
- ❑ Node.js is suitable for applications that requires reading from the network, accessing a database or the filesystem.
- ❑ Not recommended for applications requiring continues CPU processing, e.g. video processing.
- ❑ Node.js can handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.
- ❑ New ECMA JavaScript standards can be used without problems < ==> no dependency with the versions of browser.

2.2 Building your first Node.js web server

```
// After installing Node.js, create a file named app.js with the following code.
// then run your server with this line on terminal ‘node app.js’
// Visit http://localhost:3000 and you will see a message saying "Hello World".
const http = require("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");
});

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

2.3 Callback function, a foundation of Node.js

- ❑ A callback is a function called at the completion of a given task; this prevents any blocking, and allows other code to be run in the meantime.
- ❑ performing the blocking queries asynchronously gives the ability to continue working and not just sit still and wait until the blocking operations come back. This is a major improvement brought by Node.js.

```
function synchronousProcessData () {
    var data = fetchData ();
    data += 1;
    return data;
  }
```

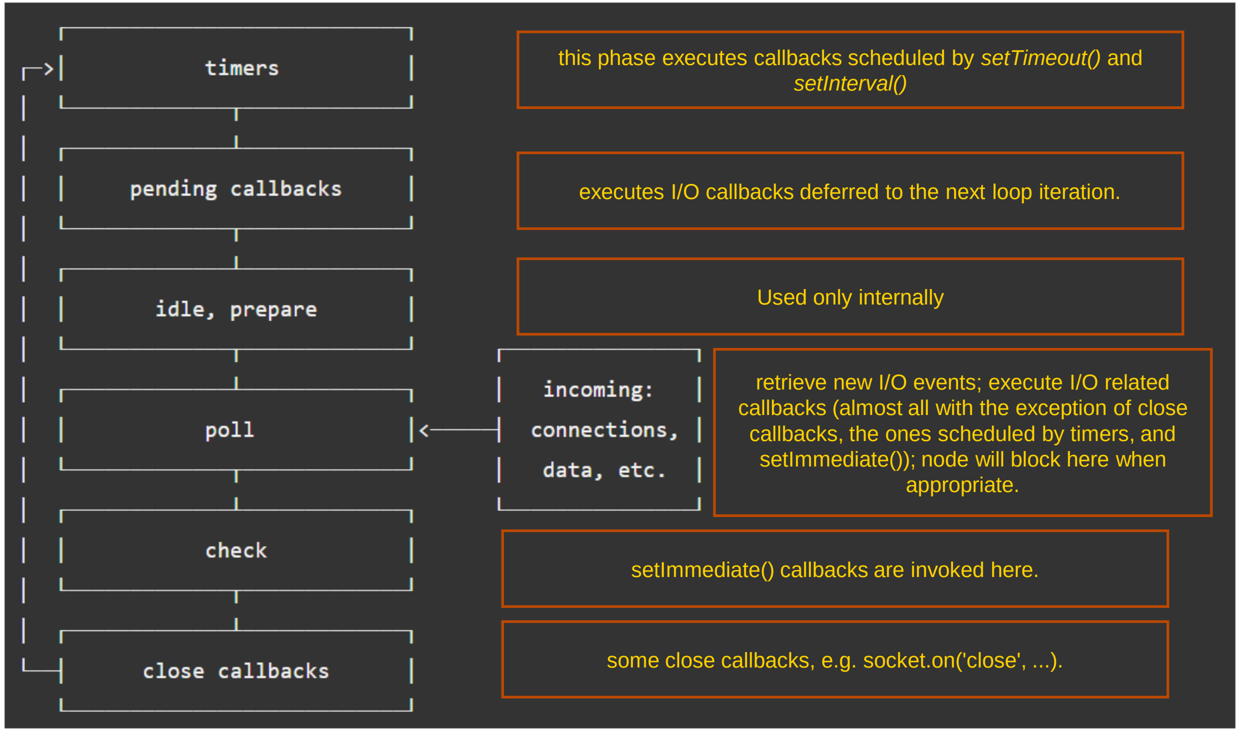
```
function asyncOperation ( a, b, c, callback ) {
    // ... lots of hard work ...
    if ( /* an error occurs */ ) {
      return callback(new Error("An error has occurred"));
    }
    // ... more work ...
    callback(null, d, e, f);
  }

  asyncOperation ( params.., function ( err, returnValues.. ) {
    //This code gets run after the async operation gets run
  });
```

2.4 Blocking vs Non-Blocking calls

- Blocking is when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes. This happens because the event loop is unable to continue running JavaScript while a blocking operation is occurring.
- All of the I/O methods in the Node.js standard library provide asynchronous versions, which are non-blocking, and accept callback functions. Some methods also have blocking counterparts, which have names that end with Sync.
- I/O" refers to interaction with the system's disk and network
- **Concurrency** refers to the event loop's capacity to execute JavaScript callback functions after completing other work.

2.5 Event Loop, Timers, and process.nextTick() (1/4)

- ❑ Event Loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.
- ❑ Since most of modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed.



timers
pending callbacks
idle, prepare
poll
check
close callbacks
incoming:
connections,
data, etc.
this phase executes callbacks scheduled by *setTimeout()* and *setInterval()*
executes I/O callbacks deferred to the next loop iteration.
Used only internally
retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and setImmediate()); node will block here when appropriate.
setImmediate() callbacks are invoked here.
some close callbacks, e.g. socket.on('close', ...).

2.5 Event Loop, Timers, and process.nextTick() (2/4)

- ❑ Timers specify the threshold after which a provided callback may be executed rather than the exact time a person wants it to be executed.
- ❑ The poll phase controls when timers are executed.
- ❑ The code example given here schedule a timeout to execute after a 100 ms threshold, then the script starts asynchronously reading a file which takes 95 ms:

```
const fs = require('fs');

function someAsyncOperation(callback) {
  // Assume this takes 95ms to complete
  fs.readFile('/path/to/file', callback);
}
const timeoutScheduled = Date.now();

setTimeout(() => {
  const delay = Date.now() - timeoutScheduled;

  console.log(`${delay}ms have passed since I was scheduled`);
}, 100);

// do someAsyncOperation which takes 95 ms to complete
someAsyncOperation(() => {
  const startCallback = Date.now();

  // do something that will take 10ms...
  while (Date.now() - startCallback < 10) {
    // do nothing
  }
});
```

2.5 Event Loop, Timers, and process.nextTick() (3/4)

setImmediate() vs setTimeout()

- ❑ **setImmediate()** is designed to execute a script once the current poll phase completes.
- ❑ **setTimeout()** schedules a script to be run after a minimum threshold in ms has elapsed.

We run the following script which is not within an I/O cycle (i.e. the main module), the order in which the two timers are executed is non-deterministic, as it is bound by the performance of the process:

```javascript
// timeout_vs_immediate.js
setTimeout(() => {
    console.log('timeout');
  }, 10);

  setImmediate(() => {
    console.log('immediate');
  });
```

Here we moved the two calls within an I/O cycle, the immediate callback is always executed first:

```javascript
// timeout_vs_immediate.js
const fs = require('fs');

fs.readFile(__filename, () => {
  setTimeout(() => {
    console.log('timeout');
  }, 10);
  setImmediate(() => {
    console.log('immediate');
  });
});
```

2.5 Event Loop, Timers, and process.nextTick() (4/4)

process.nextTick()

There are two main reasons to use it:

- Allow users to handle errors, cleanup any then unneeded resources, or perhaps try the request again before the event loop continues.
- At times it's necessary to allow a callback to run after the call stack has unwound but before the event loop continues.

```
const EventEmitter = require('events');
const util = require('util');

function MyEmitter() {
  EventEmitter.call(this);

  // use nextTick to emit the event once a handler is assigned
  process.nextTick(() => {
    this.emit('event');
  });
}
util.inherits(MyEmitter, EventEmitter);

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
```

2.6 Timers in Node.js

- ❑ "When I say so" Execution ~ setTimeout()
- ❑ "Right after this" Execution ~ setImmediate()
- ❑ "Infinite Loop" Execution ~ setInterval()setInterval

```
const timeoutObj = setTimeout(() => {
    console.log('timeout beyond time');
  }, 1500);

  const immediateObj = setImmediate(() => {
    console.log('immediately executing immediate');
  });

  const intervalObj = setInterval(() => {
    console.log('interviewing the interval');
  }, 500);

  clearTimeout(timeoutObj);
  clearImmediate(immediateObj);
  clearInterval(intervalObj);
```

2.7 Modules in Node.js

- ❑ We import module by using the ‘require’ function
- ❑ The list of Node.js known modules are provided in the official documentation of Node.js == > always refer to the latest stable version if you work on production.

https://nodejs.org/dist/latest-v14.x/docs/api/fs.html

```
  // Example of File System module (fs)
const fs = require('fs');

try {
  fs.unlinkSync('/tmp/hello');
  console.log('successfully deleted /tmp/hello');
} catch (err) {
  // handle the error
}
```

2.7 Exercise:

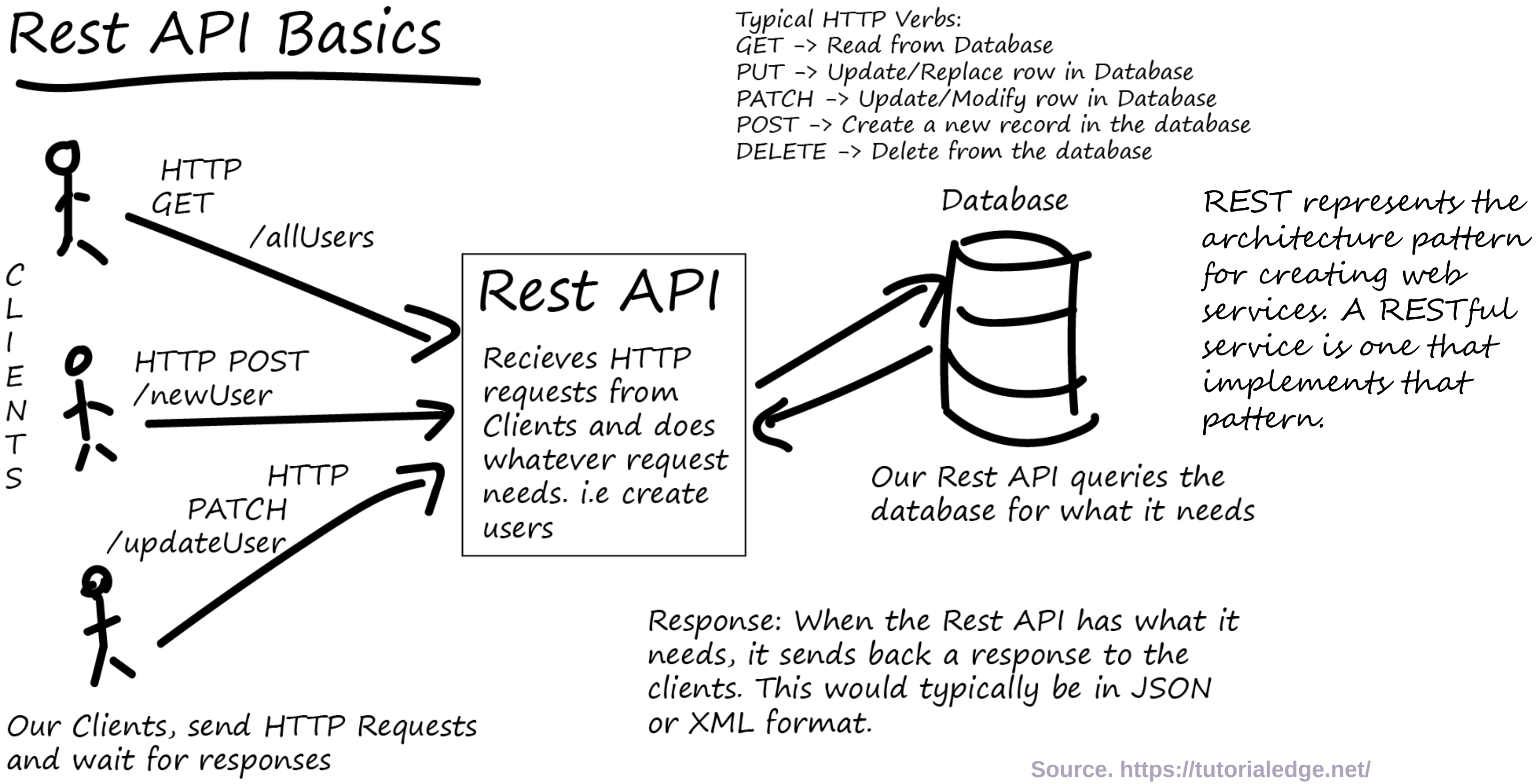
1. **Anatomy of an HTTP Transaction :**

Follow the steps given in the following link to mount and handle HTTP in Nodes.js

https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/

#3 API REST

3. API REST (Representational State Transfer API) (1/2)



Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database

Recieves HTTP requests from Clients and does whatever request needs. i.e create users

REST represents the architecture pattern for creating web services. A RESTful service is one that implements that pattern.

Our Rest API queries the database for what it needs

Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.

Our Clients, send HTTP Requests and wait for responses

Source. https://tutorialedge.net/

3. API REST (Representational State Transfer API) (2/2)

GET /api/products

GET /api/products/1

PUT /api/products/1

DELETE/api/products/1

POST/api/products

#4 MQTT

4. MQTT (Message Queuing Telemetry Transport) (1/2)

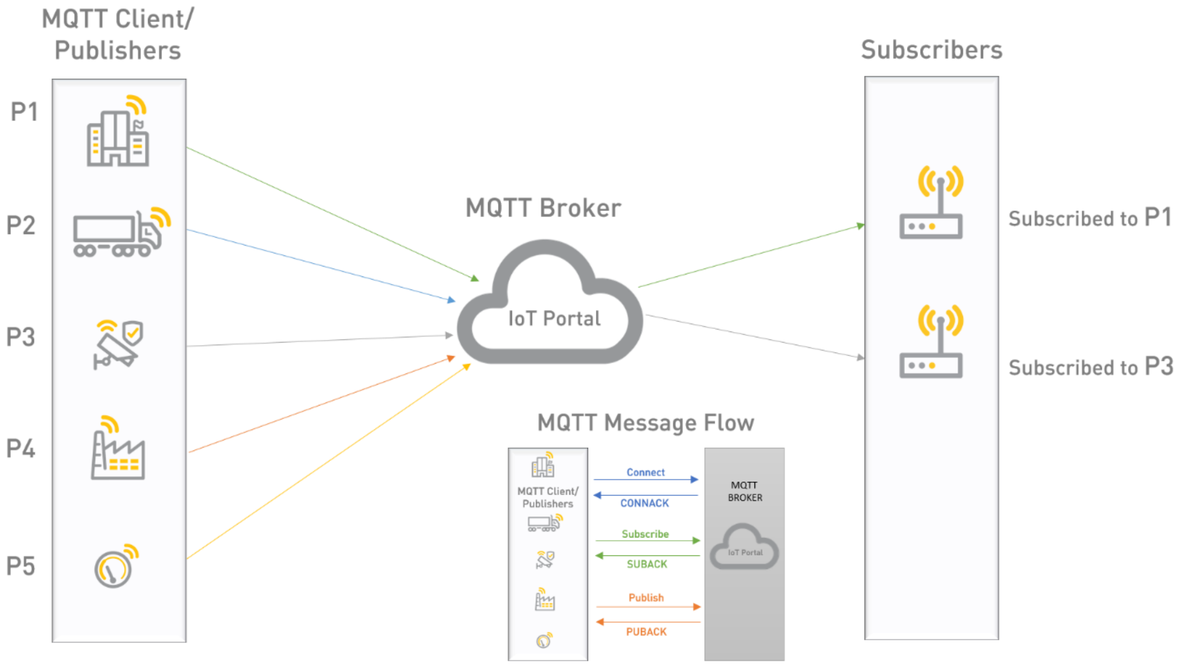
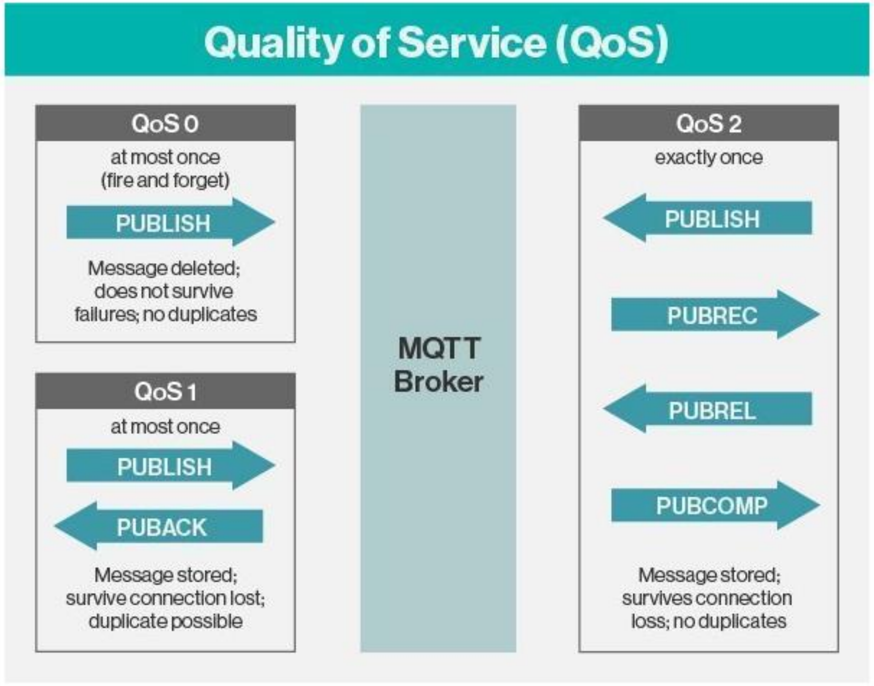
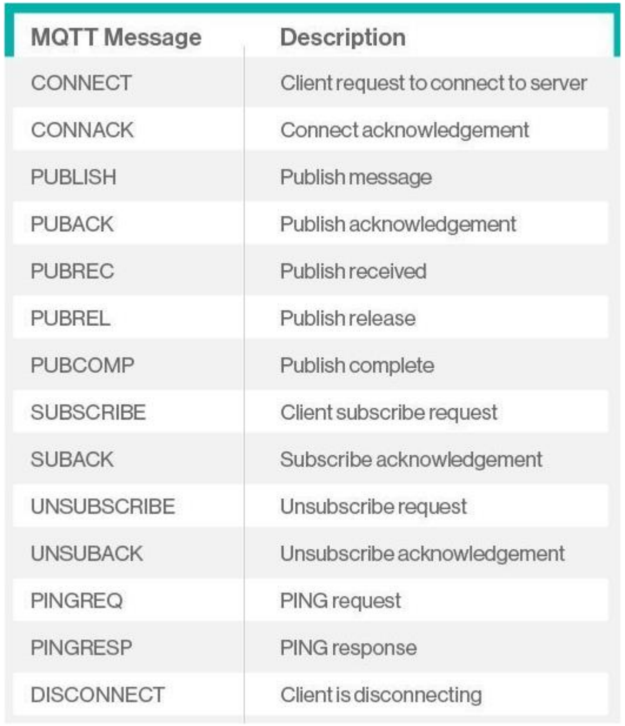


Image source. https://docs.devicewise.com/Content/Products/IoT_Portal_API_Reference_Guide/MQTT_Interface/MQTT-Interface.htm

4. MQTT (Message Queuing Telemetry Transport) (2/2)


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



MQTT Message	Description
CONNECT	Client request to connect to server
CONNACK	Connect acknowledgement
PUBLISH	Publish message
PUBACK	Publish acknowledgement
PUBREC	Publish received
PUBREL	Publish release
PUBCOMP	Publish complete
SUBSCRIBE	Client subscribe request
SUBACK	Subscribe acknowledgement
UNSUBSCRIBE	Unsubscribe request
UNSUBACK	Unsubscribe acknowledgement
PINGREQ	PING request
PINGRESP	PING response
DISCONNECT	Client is disconnecting

Images source. https://www.lemagit.fr/

#5 IOT web brokers : MQTT vs REST

5. IOT web brokers : MQTT vs REST

- ❑ Today, IoT Implementations use REST over HTTP based connectivity from the client to the Server or MQTTT.
- ❑ Limitation with REST when it comes to IOT large scale deployment.
- ❑ REST offers a uni-directional connectivity. The client will reach out to the server when it needs data, or needs to push some data.
- ❑ Servers limit in most case the frequency at which a client can connect.
- ❑ MQTT allows the client and the server to be ‘always connected’, eliminating all delays when the server needs to send a message to the client. This allows the end user to have instantaneous updates, creating a smooth user friendly interaction.
- ❑ Energy: some studies shows that MQTT offers up to 20% energy saving compared to the frequent connection-disconnection mechanisms imposed by REST.
- ❑ Performance: MQTT Keep-alive payload are much smaller compared to the connection-reconnection requests that REST requires

#6 Exercise: building a Node.js based MQTT server

6. Exercise: building a Node.js based MQTT server

- ❑ Think to how to build your MQTT server based on Node.js.
- ❑ Provides which package you have to install and use.
- ❑ Install this package and provide a very short example of and MQTT server saying “Hello IOT World !”

References

- ❑ Node.js official web page: https://nodejs.org/en/docs/
- ❑ Stackoverflow: https://stackoverflow.com/questions/35465664/ibm-iot-foundation-when-to-use-mqtt-and-when-to-use-rest-for-event-submission
- ❑ https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt
- ❑ https://www.redhat.com/fr/topics/api/what-is-a-rest-api

Thanks !

Questions ?

Samir Bellahsene
samir.bellahsene@orange.com