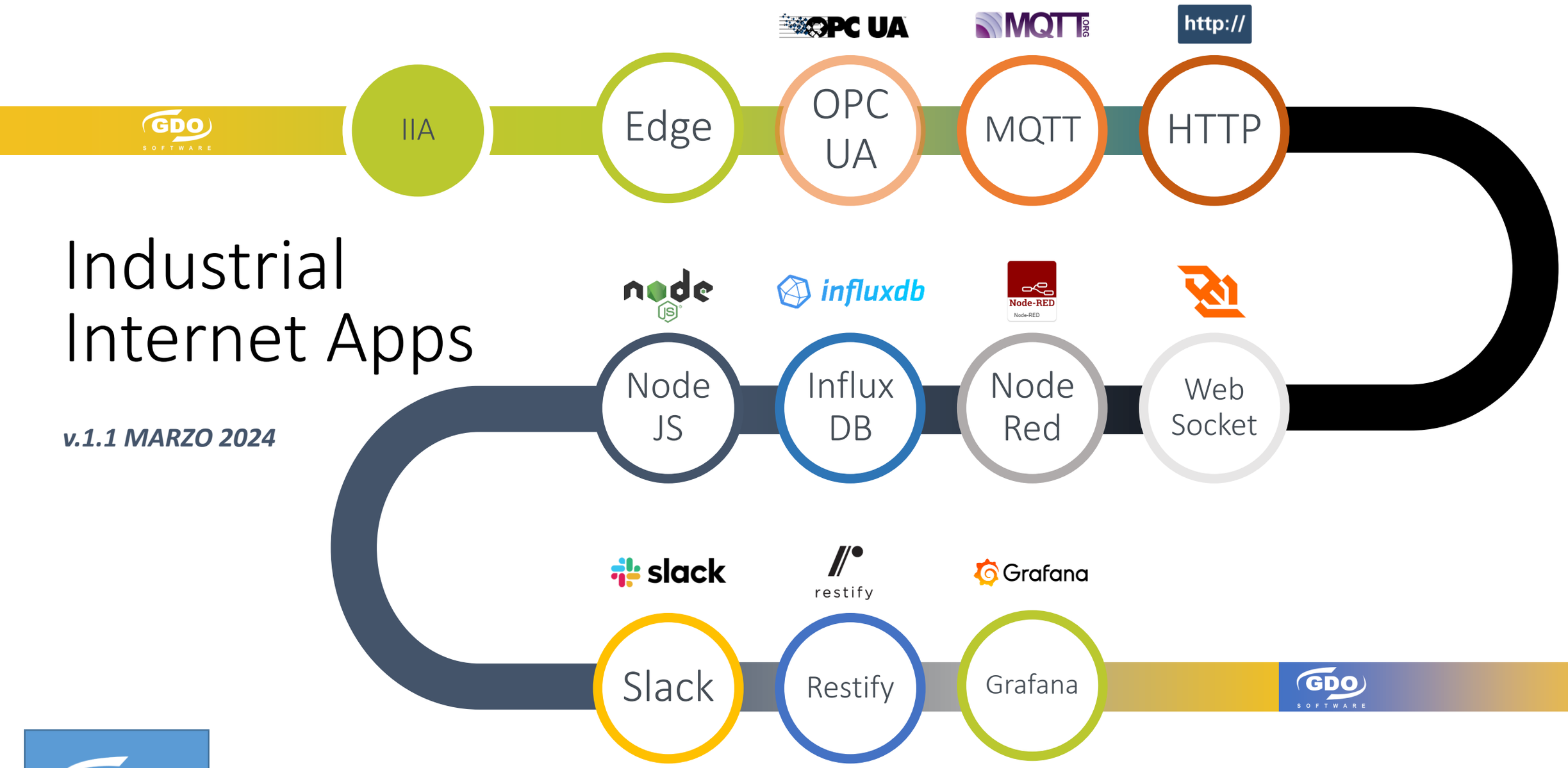
CAPÍTULO 7. Industrial Internet Apps

v.1.1 MARZO 2024

Ricardo Moraleda Gareta

[Director departamento de software de GDO Software]



Industrial Internet Apps

v.1.1 MARZO 2024

Industrial Internet Apps

Contenido

1. Arquitectura general Industrial Internet of Things (IIoT)

2. Edge
   - 2.1. Lectura de datos por OPC UA con Node-Red
   - 2.2. Envío de datos a Cloud por MQTT
   - 2.3. Envío de datos a local server por HTTP
   - 2.4. Envío de datos a local server por WebSocket

3. Base de datos InfluxDB

4. Ejemplo IIoT #1 (Node-Red, InfluxDB, MQTT, Slack)

5. Ejemplo IIoT #2 (Node-Red, InfluxDB, MQTT, Restify, Slack)

6. Ejemplo IIoT #3 (Node-Red, InfluxDB, MQTT, Grafana, Slack)

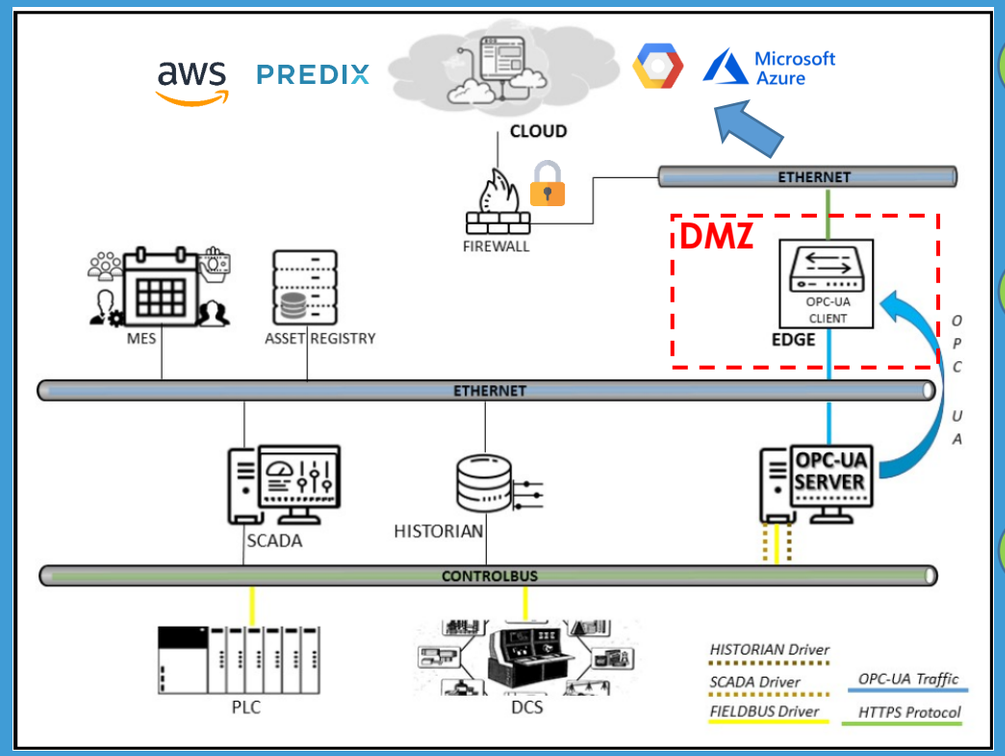
7. Aplicación local IIoT #4 (Node-Red, InfluxDB, MQTT)

8. Resumen

Industrial Internet Apps

Arquitectura general

Una arquitectura típica de fábrica es esta (1). Adquiriendo datos de campo (PLC) a través de un OPC UA Server, controlados y representados por un SCADA e historizados por un Historian.



El objetivo de **Industrial Internet of Things (IIoT)** es captar los datos (1), tratarlos en Edge (2) y mandarlos al Cloud (3) para compartirlos, filtrarlos, mostrarlos y analizarlos.

En la nube existen plataformas completas (PaaS) para este fin con repositorios de datos, aplicaciones de analítica y aplicaciones de representación como dashboards, etc. (AmazonWS, Predix, GoogleCP, Azure, …)

aws PREDIX Microsoft Azure

Hay que extremar la seguridad en este punto ya que estamos sacando datos de on-premise a internet. Entre los dispositivos (1) y cloud (3) se colocará un **EDGE** (2) que hará de Gateway entre ambos mundos. De manera resumida, todo el proceso aquí se llama **Edge Computing**.

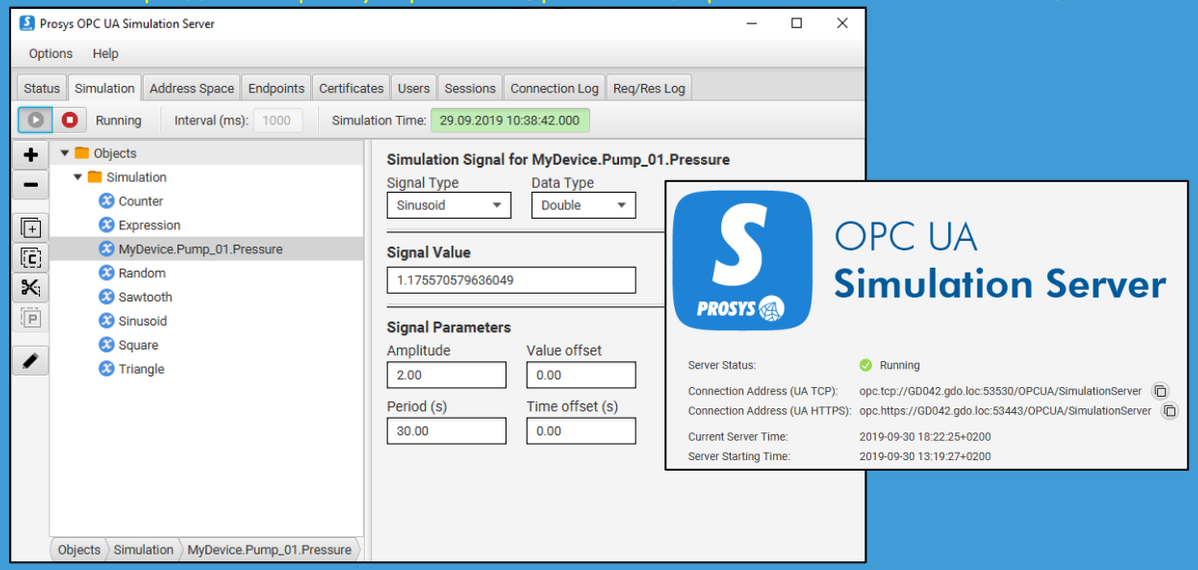
3 componentes clave de IIoT

EDGE: OPC UA

Configurar OPC UA Server

Dentro de la arquitectura tengo un OPC UA Server. Puede ser KepServerEx u otro. En este caso he utilizado Prosys opc ua simulation server.

https://www.prosysopc.com/products/opc-ua-simulation-server/

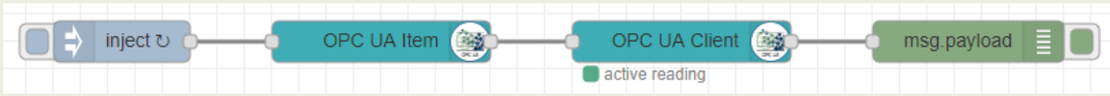


Configurando una señal "MyDevice.Pump_01.Pressure" de tipo sinusoidal.

Como servidor OPC se podría utilizar KepServerEx. Incluso envío MQTT.

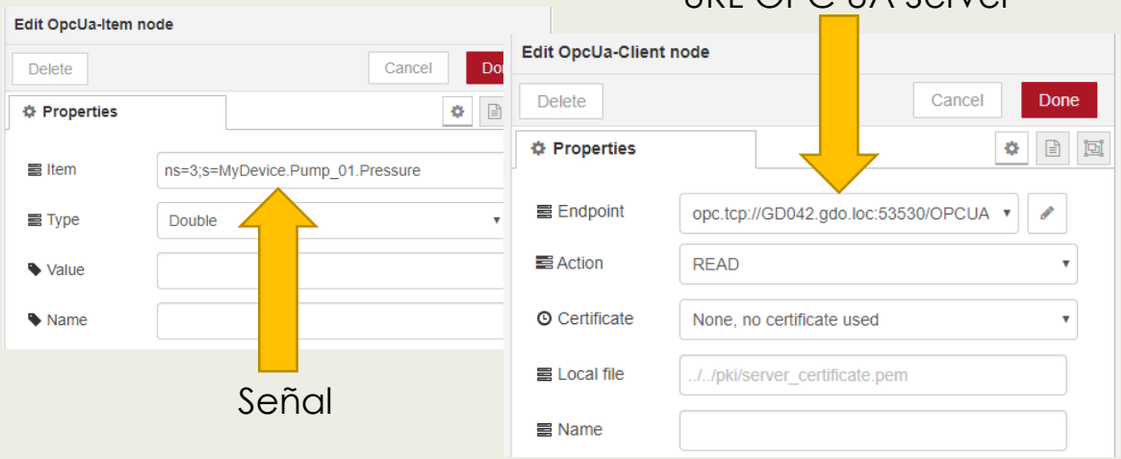
https://www.youtube.com/watch?v=yTaoGtQ5JkA

Edge Gateway: Node-Red



Con los nodos de OPC UA (node-red-contrib-opcua) como cliente podemos leer las variables del OPC UA Server.

URL OPC UA Server



A continuación, 3 maneras de enviar datos a un servidor local/cloud a través de los protocolos MQTT, HTTP y WS.

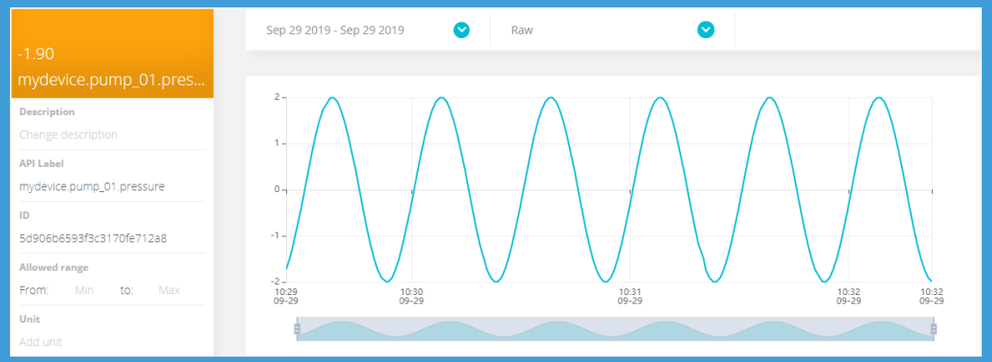
EDGE: OPC UA-MQTT

Edge Gateway: Publish MQTT broker (Cloud)

Arquitectura Edge por encima de SCADA (red de planta) para enviar datos al Cloud. Lo haré a través del protocolo MQTT enviando datos al broker Ubidots (ver Capítulo 1 - IoT).

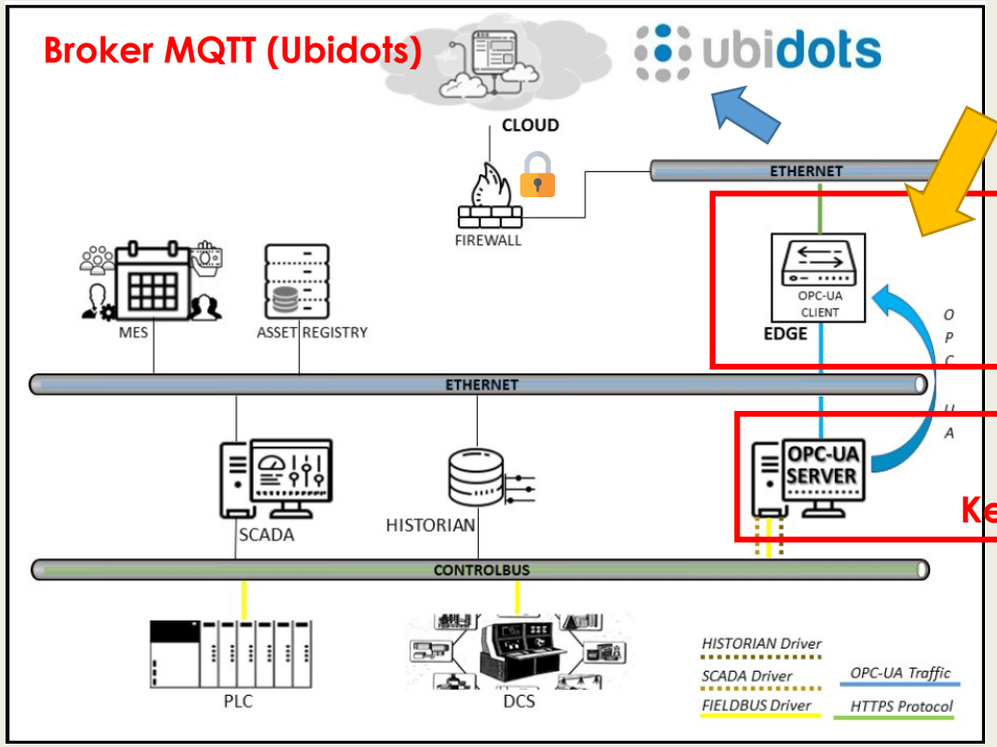
En Node-Red tal cual leemos el dato lo enviamos al Cloud.

Si vamos al broker MQTT (Ubidots en el Cloud - industrial.ubidots.com) podemos ver la lectura de los valores enviados desde el Edge Gateway.



Arquitectura SCADA-OPC UA Server-Edge Gateway-MQTT

Esta es la arquitectura ejemplo seguida. Una vez los datos en el broker, los clientes se pueden suscribir para leer esos datos (ver Capítulo 1).



Hardware: Puede ser una Raspberry (Capít. 2)

Node-Red (OPC UA Client y MQTT)

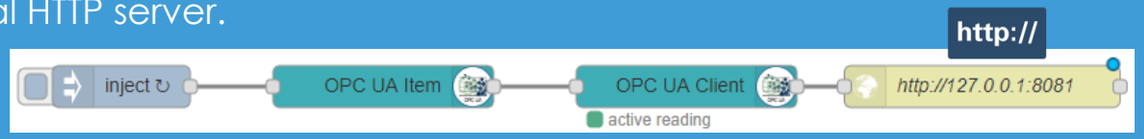
Prosys / KepServerEx...

EDGE: OPC UA-HTTP

Edge Gateway: Publishing via HyperText Transfer Protocol (HTTP)

Siguiendo la misma arquitectura, en lugar de enviar los datos por MQTT lo haré a través del protocolo HTTP enviando datos a un servidor que tengo en local (debería estar en remoto - cloud).

En Node-Red tal cual leemos el dato (Method = POST) lo enviamos al HTTP server.



En el otro lado, el servidor HTTP lo implemento en Node.js. Escucha por el puerto 8081.

package.json: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
{
  "name": "receiver",
  "version": "1.0.0",
  "description": "",
  "main": "index2.js",
  "scripts": {
    "start": "node index2.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
}
```

index2.js: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
var http = require('http');
var querystring = require('querystring');
http.createServer(function (req, res) {
req.on('data', function (chunk) {
var data = querystring.parse(chunk.toString());
console.log(data);
});
req.on('end', function () {
res.writeHead(200, 'OK', {'Content-Type': 'text/html'});
res.end('Data received.')
});
}).listen(8081);
```

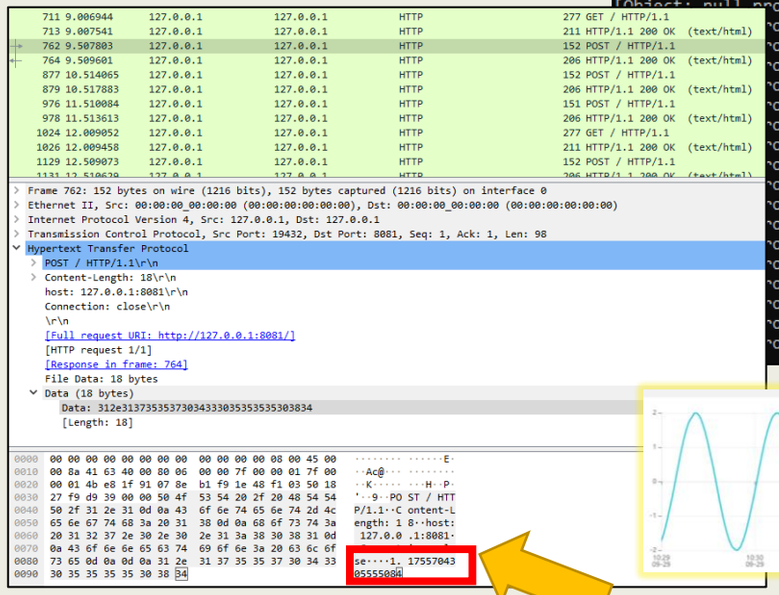
HTTP Server Receiving data

En CMD ejecuto "node index2.js" para iniciar el servidor y que vaya printando los datos recibidos (enviados por el Edge Gateway).

Analizado con Wireshark.

Símbolo del sistema - node index2.js

```
C:\receiver>node index2.js
[Object: null prototype] { '-1.9021130028322535': '' }
[Object: null prototype] { '-1.7320507535911507': '' }
[Object: null prototype] { '-1.9890437818889728': '' }
[Object: null prototype] { '-1.9890437983656948': '' }
[Object: null prototype] { '-1.9021130515423077': '' }
[Object: null prototype] { '-1.732050832405674': '' }
[Object: null prototype] { '-1.4862896763932762': '' }
[Object: null prototype] { '-1.1755705259113245': '' }
[Object: null prototype] { '-0.8134732995848137': '' }
[Object: null prototype] { '-0.41582338461702206': '' }
[Object: null prototype] { '-1.1060103084828562e-7': '' }
[Object: null prototype] { '0.4158232848530362': '' }
[Object: null prototype] { '0.8134732064097793': '' }
[Object: null prototype] { '1.1755704433974377': '' }
[Object: null prototype] { '1.4862896081467905': '' }
[Object: null prototype] { '1.732050781409288': '' }
[Object: null prototype] { '1.9021130200248084': '' }
[Object: null prototype] { '1.9890437877045484': '' }
[Object: null prototype] { '1.9890437925501223': '' }
```



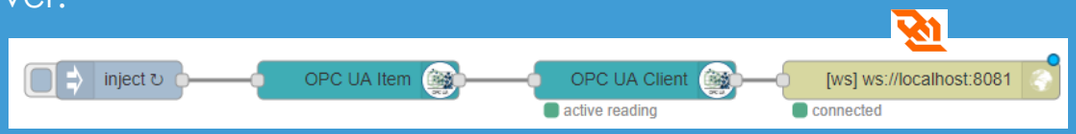
Payload o valor enviado/recibido

EDGE: OPC UA-WS

Edge Gateway: Publishing via WebSocket (WS)

Siguiendo la misma arquitectura, en lugar de enviar los datos por MQTT o HTTP lo haré a través del protocolo WebSocket (TCP) enviando datos a un servidor que tengo en local (debería estar en remoto - cloud).

En Node-Red tal cual leemos el dato lo enviamos al WebSocket server.



En el otro lado, el servidor WebSocket lo implemento en Node.js. Escucha por el puerto 8081.

package.json: Bloc de notas

```
{
  "name": "receiver",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ws": "^2.3.1"
  }
}
```

index.js: Bloc de notas

```
const WebSocket = require('ws');
const wss = new WebSocket.Server({port: 8081}, function () {
console.log('Websocket server started');
});
wss.on('connection', function connection(ws) {
ws.on('message', function incoming(message) {
console.log('received: ', message);
});
// Send message to connected client
ws.send('hello, client');
});
```

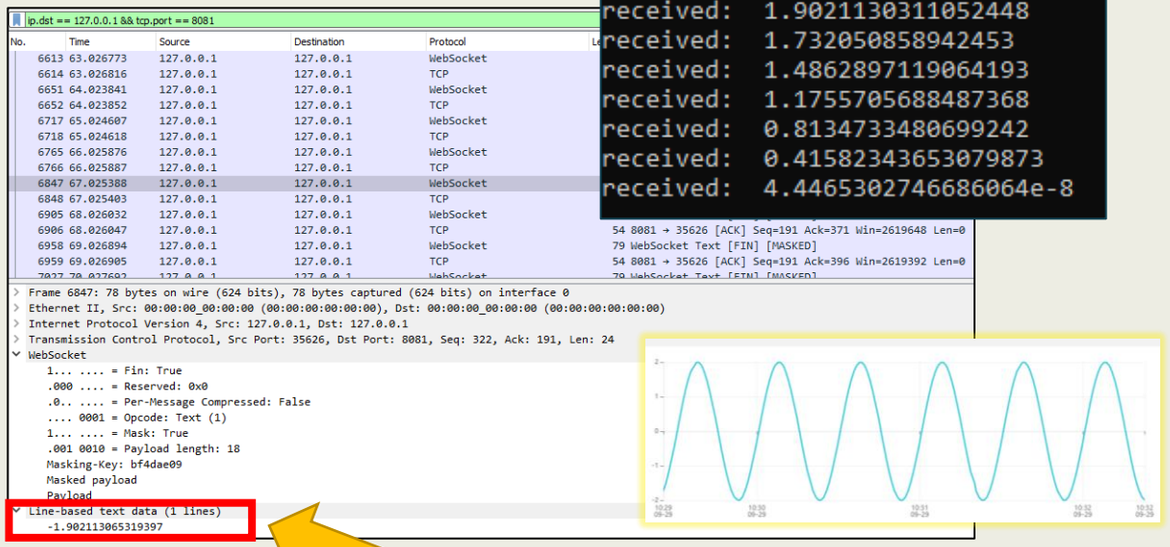
WebSocket Server Receiving data

En CMD ejecuto "node index.js" para iniciar el servidor y que vaya printando los datos recibidos (enviados por el Edge Gateway).

Símbolo del sistema - node index.js

```
C:\receiver>node index.js
Websocket server started
received:  1.9890437914526302
received:  1.9021130311052448
received:  1.732050858942453
received:  1.4862897119064193
received:  1.1755705688487368
received:  0.8134733480699242
received:  0.41582343653079873
received:  4.4465302746686064e-8
```

Analizado con Wireshark.



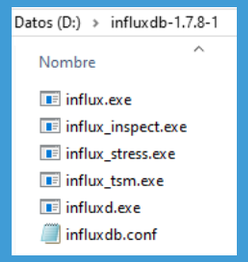
Payload o valor enviado/recibido

InfluxDB

Influx DB (database)

Lo primero es crear la base de datos. Me he descargado la versión "influxdb-1.7.8-1".

Para iniciarla, ir a la carpeta de ejecutables y en CMD poner "**influxd**".




```
D:\influxdb-1.7.8-1>influxd

 8888888          .d888 888                   8888888b.  888888b.
   888           d88P"  888                   888  "Y88b 888  "88b
   888           888    888                   888    888 888  .88P
   888   88888b.  888888 888 888  888 888  888 888    888 8888888K.
   888   888 "88b 888    888 888  888 888  888 888    888 888  "Y88b
   888   888  888 888    888 888  888 888  888 888    888 888    888
   888   888  888 888    888 Y88b 888 .d8""8b. 888  .d88P 888   d88P
 8888888 888  888 888    888  "Y88888 888  888 8888888P"  8888888P"

2019-10-01T09:06:38.777780Z     info    InfluxDB starting       {"log_id": "0IDdzpE0000", "version": "1.7.8", "branch":
"1.7", "commit": "ff383cdc0420217e3460dabe17db54f8557d95b6"}
2019-10-01T09:06:38.780778Z     info    Go runtime      {"log_id": "0IDdzpE0000", "version": "go1.11", "maxprocs": 12}
2019-10-01T09:06:38.888704Z     info    Using data dir  {"log_id": "0IDdzpE0000", "service": "store", "path": "C:\\User
\\r.moraleda\\.influxdb\\data"}
2019-10-01T09:06:38.888704Z     info    Compaction settings     {"log_id": "0IDdzpE0000", "service": "store", "max_conc
rrent_compactions": 6, "throughput_bytes_per_second": 50331648, "throughput_bytes_per_second_burst": 50331648}
2019-10-01T09:06:38.889675Z     info    Open store (start)      {"log_id": "0IDdzpE0000", "service": "store", "trace_id
: "0IDdzpfG000", "op_name": "tsdb_open", "op_event": "start"}
2019-10-01T09:06:38.943622Z     info    Reading file    {"log_id": "0IDdzpE0000", "engine": "tsm1", "service": "cacheld
der", "path": "C:\\Users\\r.moraleda\\.influxdb\\wal\\_internal\\monitor\\1\\_00001.wal", "size": 135859}
2019-10-01T09:06:38.956610Z     info    Opened shard    {"log_id": "0IDdzpE0000", "service": "store", "trace_id": "0IDd
pfG000", "op_name": "tsdb_open", "index_version": "inmem", "path": "C:\\Users\\r.moraleda\\.influxdb\\data\\_internal\\
onitor\\1", "duration": "24.933ms"}
2019-10-01T09:06:38.957611Z     info    Open store (end)        {"log_id": "0IDdzpE0000", "service": "store", "trace_id
: "0IDdzpfG000", "op_name": "tsdb_open", "op_event": "end", "op_elapsed": "67.818ms"}
2019-10-01T09:06:38.957611Z     info    Opened service  {"log_id": "0IDdzpE0000", "service": "subscriber"}
2019-10-01T09:06:38.958609Z     info    Starting monitor service        {"log_id": "0IDdzpE0000", "service": "monitor"}
2019-10-01T09:06:38.958609Z     info    Registered diagnostics client   {"log_id": "0IDdzpE0000", "service": "monitor",
"name": "build"}
2019-10-01T09:06:38.958609Z     info    Registered diagnostics client   {"log_id": "0IDdzpE0000", "service": "monitor",
"name": "runtime"}
2019-10-01T09:06:38.958609Z     info    Registered diagnostics client   {"log_id": "0IDdzpE0000", "service": "monitor",
"name": "network"}
```

Para abrir la consola (Shell) de influxdb poner "**influx**".

```
D:\influxdb-1.7.8-1>influx
Connected to http://localhost:8086 version 1.7.8
InfluxDB shell version: 1.7.8
> create database "mydb"
>
```

Influx DB (database)

Para crear la base de datos de ejemplo poner "**create database mydb**". mydb es el nombre de la base de datos.

Podemos usarla y luego hacer queries IQL similar a SQL para ver los resultados. Todo esto se puede hacer desde la web de administración de la BD que se llama **Chronograf** (puerto 8888).

Símbolo del sistema - influx

```
D:\influxdb-1.7.8-1>influx
Connected to http://localhost:8086 version 1.7.8
InfluxDB shell version: 1.7.8
> show databases
name: databases
name
----
_internal
mydb
> use mydb
Using database mydb
> SELECT COUNT(*), MEAN(*), MEDIAN(*), MODE(*), SPREAD(*), STDDEV(*), SUM(*), FIRST(*), LAST(*), MAX(*), MIN(*), PERCENT
ILE(*, 5) FROM "sensor1" WHERE time > now() - 10m
name: sensor1
time                count_value mean_value            median_value mode_value spread_value stddev_value       sum_value
           first_value last_value max_value min_value percentile_value
----                ----------- ----------            ------------ ---------- ------------ ------------       ---------
           ----------- ---------- --------- --------- ----------------
1569970805790855001 300         -2.2204460492503130e-18 0            0.42       3.98         1.4169541973276398 -6.661338
147750939e-16 1.9         1.49       1.99      -1.99     -1.99
> select time, value from sensor1 WHERE time > now() - 5m
name: sensor1
time                value
----                -----
1569971122434456500 -1.99
1569971124432848500 -1.9
1569971126433200400 -1.49
1569971128434003800 -0.81
1569971130433327000 0
```

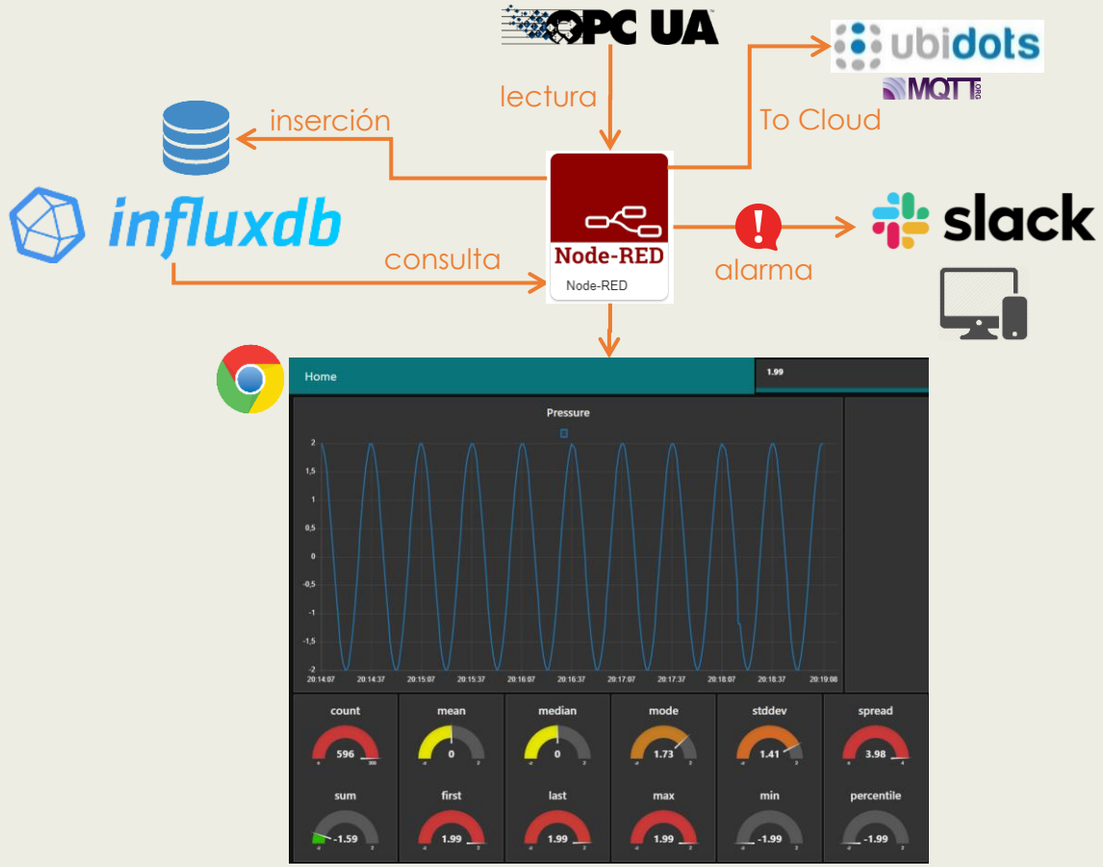
IIoT App 1

IIoT App ejemplo #1

Vamos a crear una aplicación para obtener datos de una señal de un OPC UA Server (M2M protocol), guardarlas en una base de datos de series temporales, mostrarlos en un dashboard web y enviar alarmas si supera un cierto valor.

1. Para ello vamos a utilizar **Node-Red.**
2. Los datos se envían directamente al Cloud por MQTT.
3. Los datos los almacenaremos en una base de datos óptima para series temporales como es **InfluxDB**. Tiene una precisión de nanosegundos. Es rápida y madura y permite operaciones analíticas.
4. Vamos a representar estos datos con los propios dashboards de **Node-Red**.
5. Enviaremos alarmas a **Slack** si el valor leído supera un umbral. Slack es una herramienta de mensajería en equipo. Utilizaremos, tanto la versión escritorio como la versión móvil.

Arquitectura #1



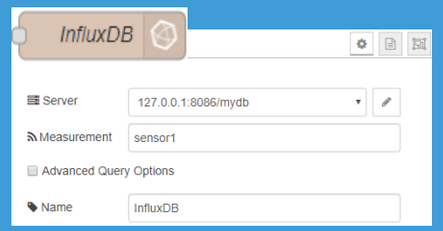
IIoT App 1

IIoT App ejemplo #1 (Node-Red)

Igual que en los ejemplos anteriores (MQTT, HTTP y WebSocket) leemos una señal del servidor OPC UA. Es una señal sinusoidal que va de -2 a 2.

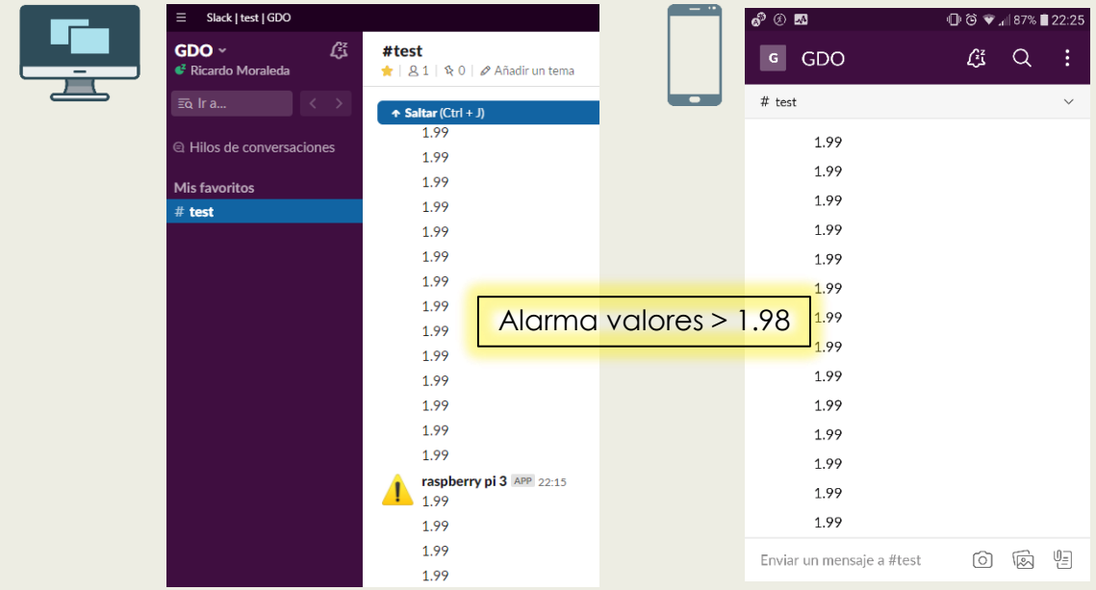
Primero acotamos a 2 decimales el valor leído y lo insertamos en la base de datos InfluxDB.

Además si el valor supera 1.98 mandamos una alarma a Slack, al canal #test de GDO. Consultable en desktop y smartphone.



IIoT App ejemplo #1 (Alarmas en Slack)

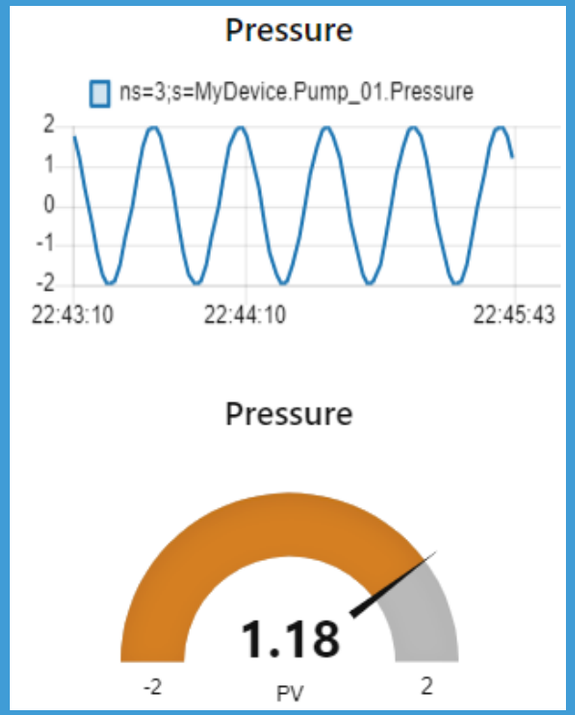
Para poder enviar estos mensajes a Slack necesitamos, a parte de tener la cuenta creada y el canal #test, una app llamada Incoming WebHooks asociada al canal #test. Este servicio es una URL del estilo: https://hooks.slack.com/services/TDDxxx/yyyyyy/zzzzzzzz

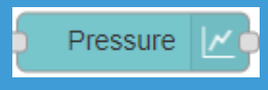


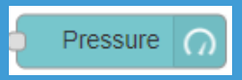
IIoT App 1

IIoT App ejemplo #1 (Node-Red)

Además, en este ejemplo, para comprobaciones, se grafica en tiempo real el valor en un gauge y una line chart (últimos 5 minutos) con la librería Dashboard de Node-Red.

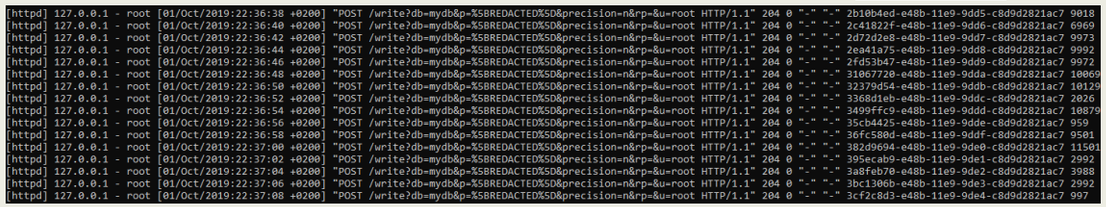






Datos en InfluxDB / InfluxQL / Chronograf (web admin IDB)

En el log de InfluxDB se puede ir viendo las inserciones mediante HTTP y método POST que va haciendo Node Red.

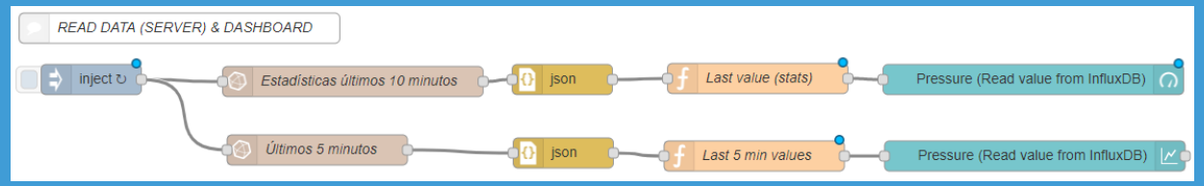


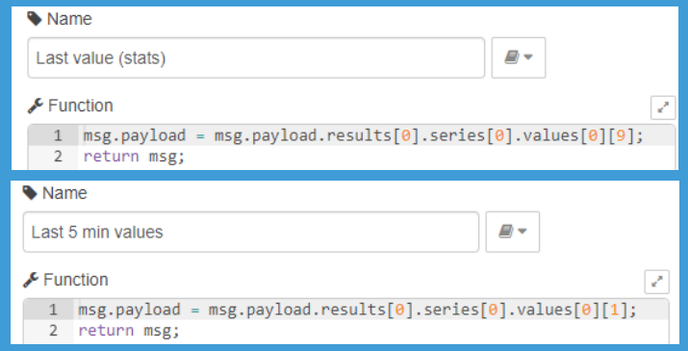
IIoT App 1

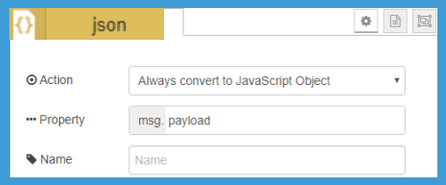
IIoT App ejemplo #1 (Node-Red)

Leemos (queries IQL) los datos que se van insertando en la base de datos (InfluxDB), y los vamos graficando. Hacemos un paso a formato JSON (objeto JavaScript) y nos quedamos con el dato adecuado usando funciones.

Las queries InfluxQL usadas son las mismas que ya hemos visto anteriormente (stats y last).



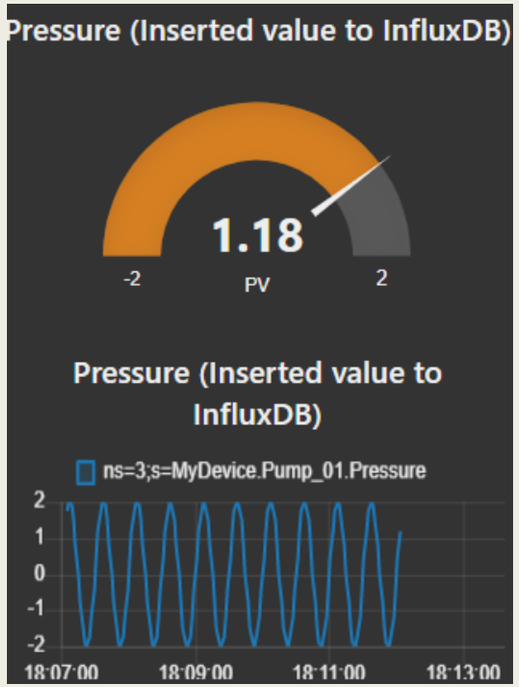


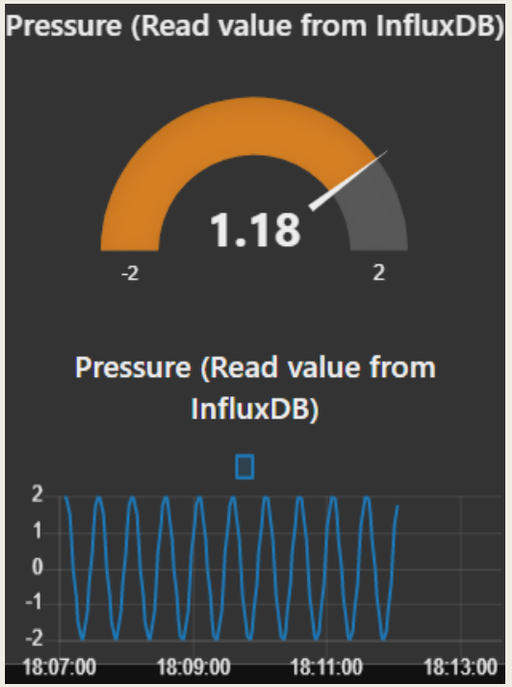


Visualización de los datos en tiempo real (Dashboard)

Si comparamos los gráficos de los valores en el momento de insertarse en InfluxDB y después, al leerlos de la BD, son iguales.

localhost:1880/ui

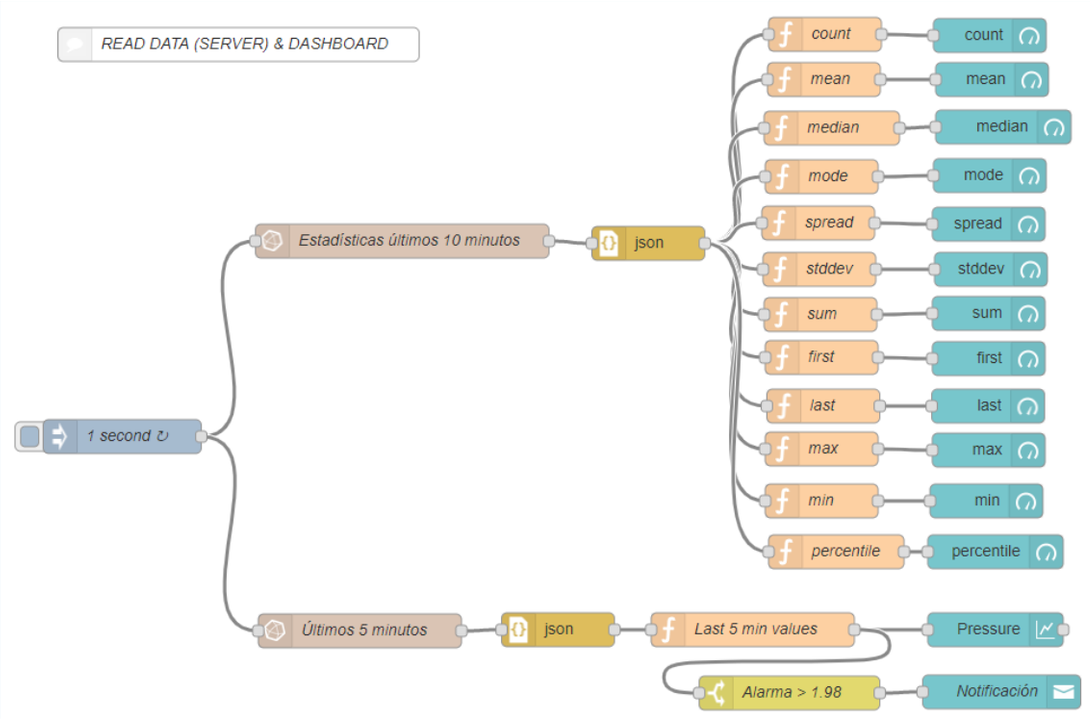




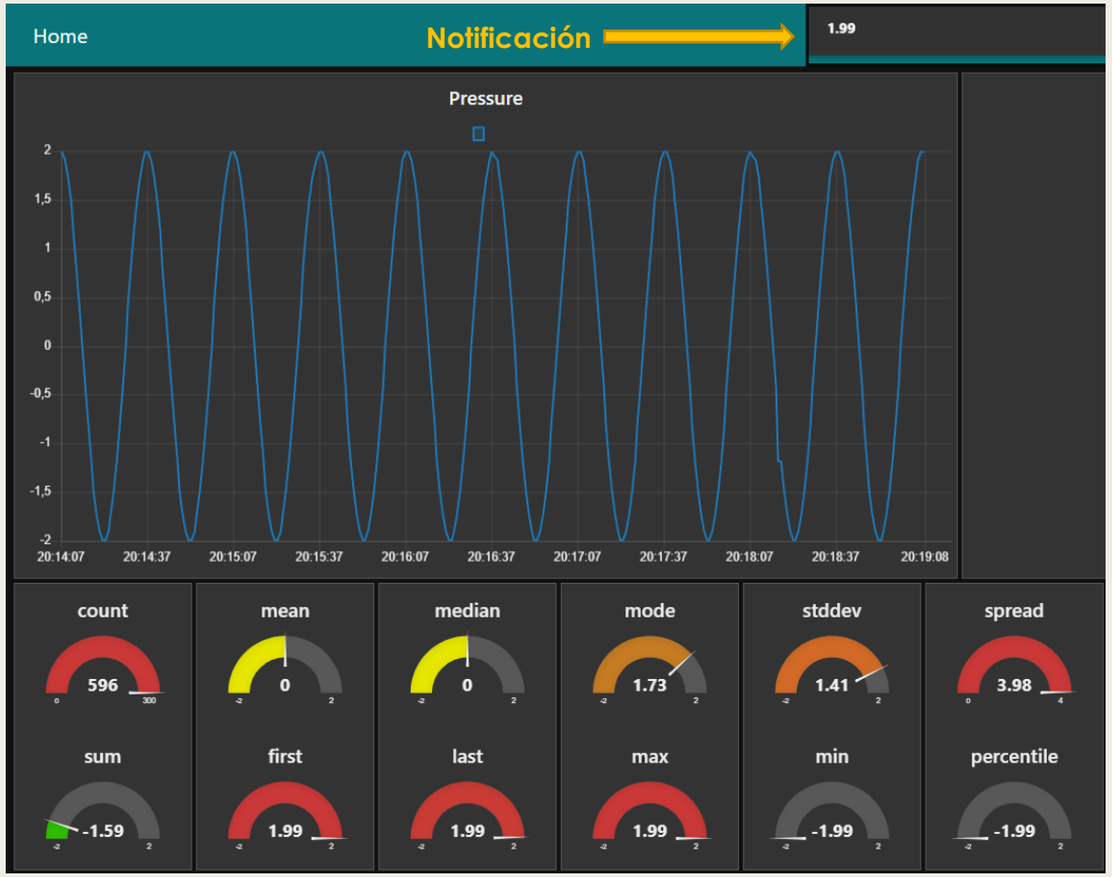
IIoT App 1

IIoT App ejemplo #1 (Node-Red)

Ahora podríamos hacer un Dashboard que consiste en una Line Chart con los valores de la señal de los últimos 5 minutos y 12 gauges, uno por cada estadística seleccionada.



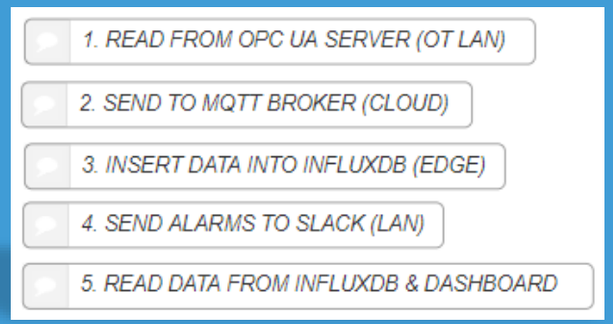
#1 Dashboard de Node-Red



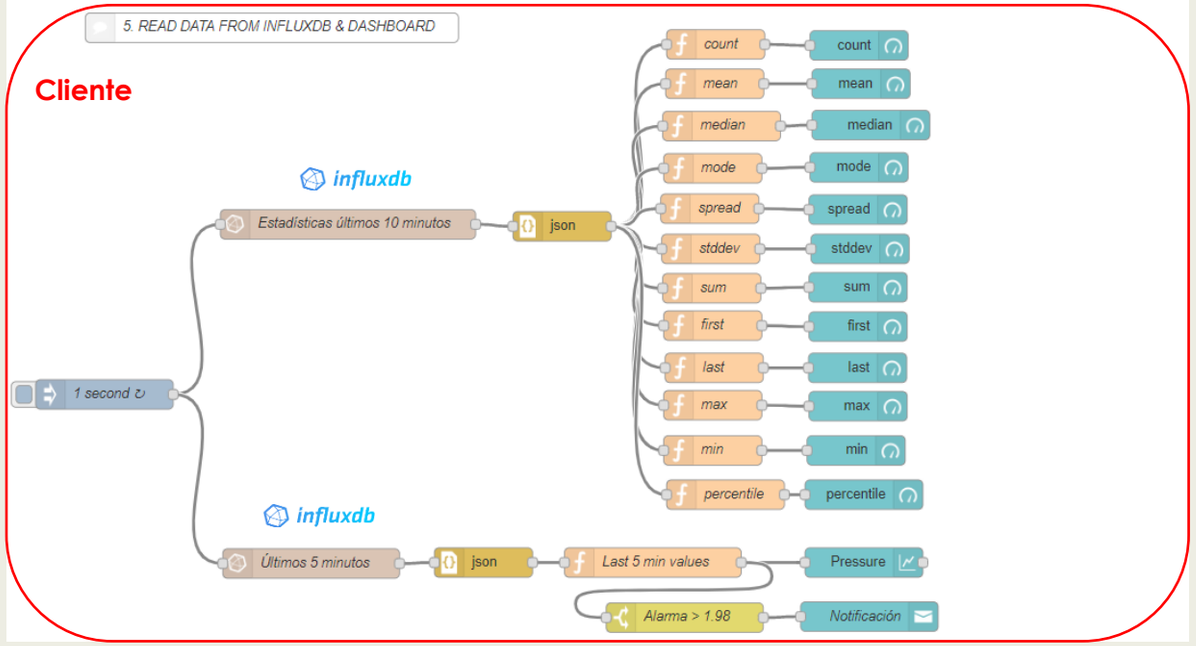
IIoT App 1

Resumen IIoT App ejemplo #1 (Node-Red)

- Como **Edge** y como ya explicamos, la arquitectura típica industrial, hace de enlace entre la planta y el cloud.
- Sus funciones de servidor son la de recoger los datos de planta, enviarlos a la nube (recomendado previo filtro), guardarlos en una base de datos (Edge y Cloud) y envío de alarmas si los valores pasan un cierto rango.
- Por otra parte, su función de cliente es servir los datos almacenados en la base de datos para representarlos en bonitos dashboards.



Servidor: Parte común entre este y los otros 2 ejemplos más que se van a mostrar (Node-Red, Node.js-Restify y Grafana). Lo que cambia entre ellos es la representación.



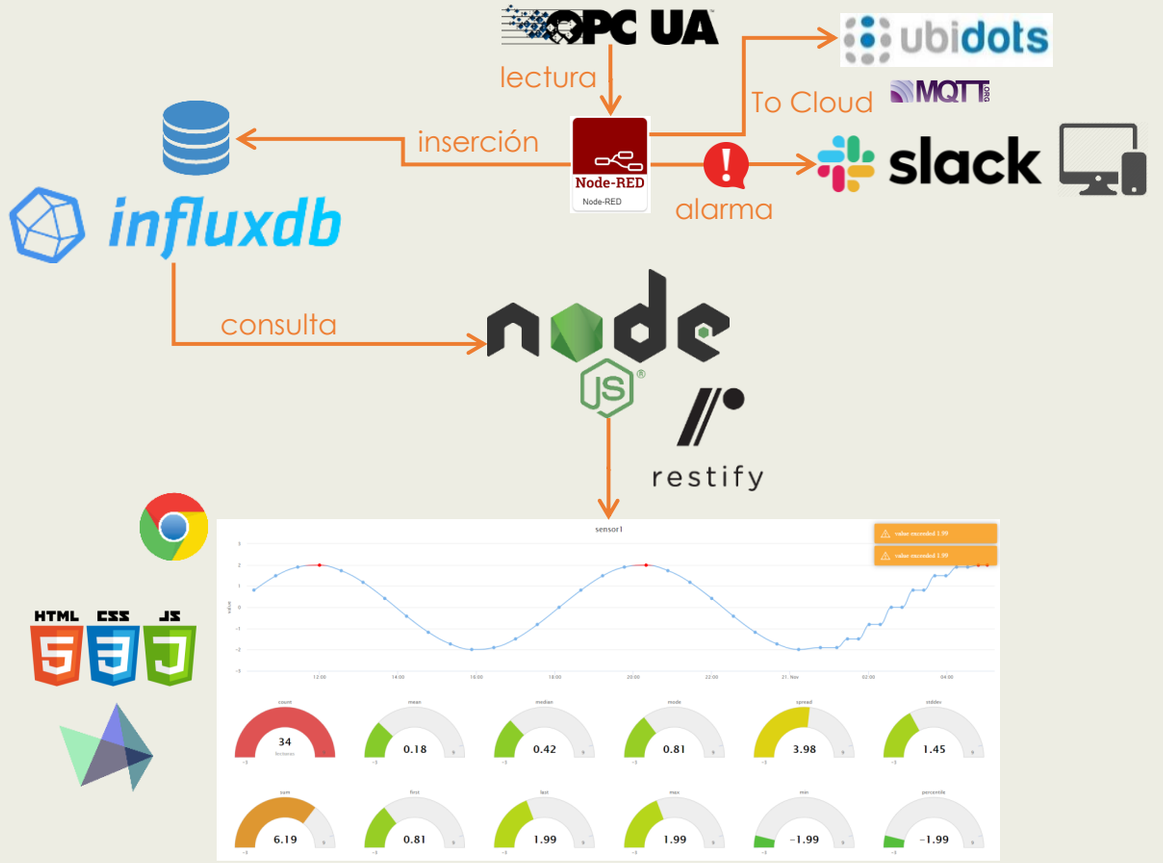
IIoT App 2

IIoT App ejemplo #2

Vamos a crear una aplicación para obtener datos de una señal de un OPC UA Server (M2M protocol), guardarlas en una base de datos de series temporales, mostrarlos en un dashboard web y enviar alarmas si supera un cierto valor.

1. Para ello vamos a utilizar **Node-Red y Node.js** con las librerías: restify, influx y slack-incoming-webhook.
2. Los datos se envían directamente al Cloud por MQTT.
3. Los datos los almacenaremos en una base de datos óptima para series temporales como es **InfluxDB**.
4. Vamos a representar estos datos con chart y gauges de **HighCharts** (JavaScript) en una página web accesible desde un navegador web.
5. Enviaremos alarmas a **Slack** si el valor leído supera un umbral. Slack es una herramienta de mensajería en equipo. Utilizaremos, tanto la versión escritorio como la versión móvil.

Arquitectura #2



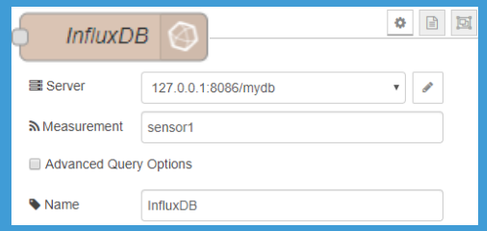
IIoT App 2

IIoT App ejemplo #2 (Node-Red)

Leemos una señal del servidor OPC UA. Es una señal sinusoidal que va de -2 a 2.

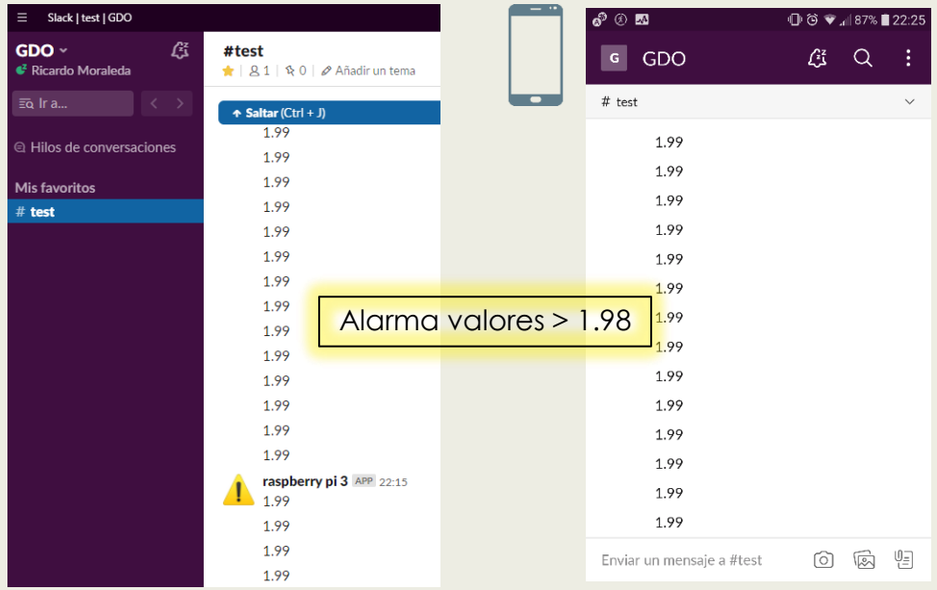
Primero acotamos a 2 decimales el valor leído y lo insertamos en la base de datos InfluxDB.

Además si el valor supera 1.98 mandamos una alarma a Slack, al canal #test de GDO. Consultable en desktop y smartphone.



IIoT App ejemplo #2 (Alarmas en Slack)

Para poder enviar estos mensajes a Slack necesitamos, a parte de tener la cuenta creada y el canal #test, una app llamada Incoming WebHooks asociada al canal #test. Este servicio es una URL del estilo: https://hooks.slack.com/services/TDDxxx/yyyyyy/zzzzzzzz



IIoT App 2

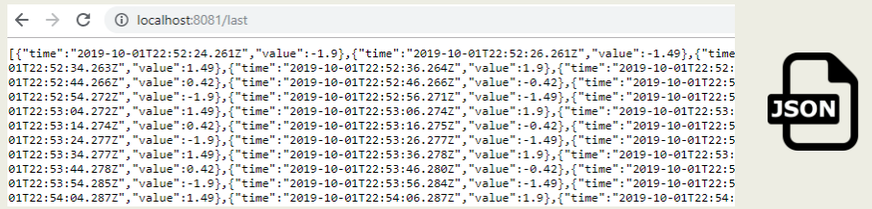
IIoT App ejemplo #2 (REST server)

Una vez que ya tengo un flujo de datos insertándose en la base de datos (InfluxDB), se trataría de leerlos y graficarlos en tiempo real. Para ello lo haremos con un REST (HTTP) Server (restify) en Node.js y una web app directamente con JavaScript y HTML (programación pura).

La aplicación JavaScript (REST-HTTP SERVER :8081) hace las queries a InfluxDB y publica en unas determinadas URLs ficheros JSON con los datos para que una web app los pueda consumir en tiempo real.

Publica los datos en ficheros JSON en URLs

Fichero JSON obtenidos con las queries InfluxQL (IQL).




```
server.get('/last', function search(req, res, next) {
  influx.query('select time, value from sensor1 WHERE time > now() - 5m').then(function (result) {
    res.json(result)
  }).catch(function (err) {
    res.status(500).send(err.stack)
  });
  return next();
});

server.get('/stats', function search(req, res, next) {
  influx.query('SELECT COUNT(*), MEAN(*), MEDIAN(*), MODE(*), SPREAD(*), STDDEV(*), SUM(*), FIRST(*), LAST(*), MAX(*), MIN(*), PERCENTILE(*, 5) FROM "sensor1" WHERE time > now() - 10m').then(function (result) {
    res.json(result)
  }).catch(function (err) {
    res.status(500).send(err.stack)
  });
  return next();
});
```

Datos de los últimos 5 minutos

Datos estadísticos de los últimos 10 minutos

IIoT App 2

IIoT App ejemplo #2 (Web App Client)

En la URL "http://localhost:8081/public/index.html" dejo una aplicación web que muestra los gráficos de los datos obtenidos del servidor REST. Está desarrollada en HTML, JavaScript (JS) y CSS.

Por un lado se grafica un Line Chart con los datos de los últimos 5 minutos (JSON) y 12 Gauges, uno por cada dato estadístico consultado (JSON). Las gráficas son de HighCharts.

Datos de los últimos 5 minutos

```
$.getJSON('http://localhost:8081/last', function (response) {
  var data = [];
  for (var i = 0; i < response.length; i++) {
    data.push({
      x: new Date(response[i].time).getTime() * 1000,
      y: response[i].value
    });
  }
```

JSON

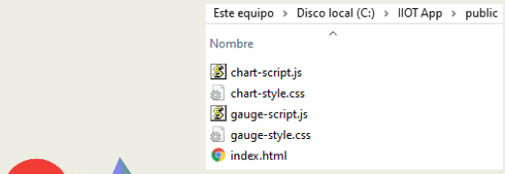
Datos estadísticos de los últimos 10 minutos

```
$.getJSON('http://localhost:8081/stats', function (response) {
  stats = response;
  for (var i = 0; i < gaugeTypes.length; i++) {
    var name = gaugeTypes[i].name;
    if (!gauges[name]) return;
    var newVal = stats[0][name + '_value'];
    if (!newVal) return;
    newVal = parseFloat(newVal.toFixed(2));
    var point = gauges[name].series[0].points[0];
    point.update(newVal);
  }
```

JSON

Visualización de los datos en tiempo real

Ficheros de la web app:



IIoT App 3

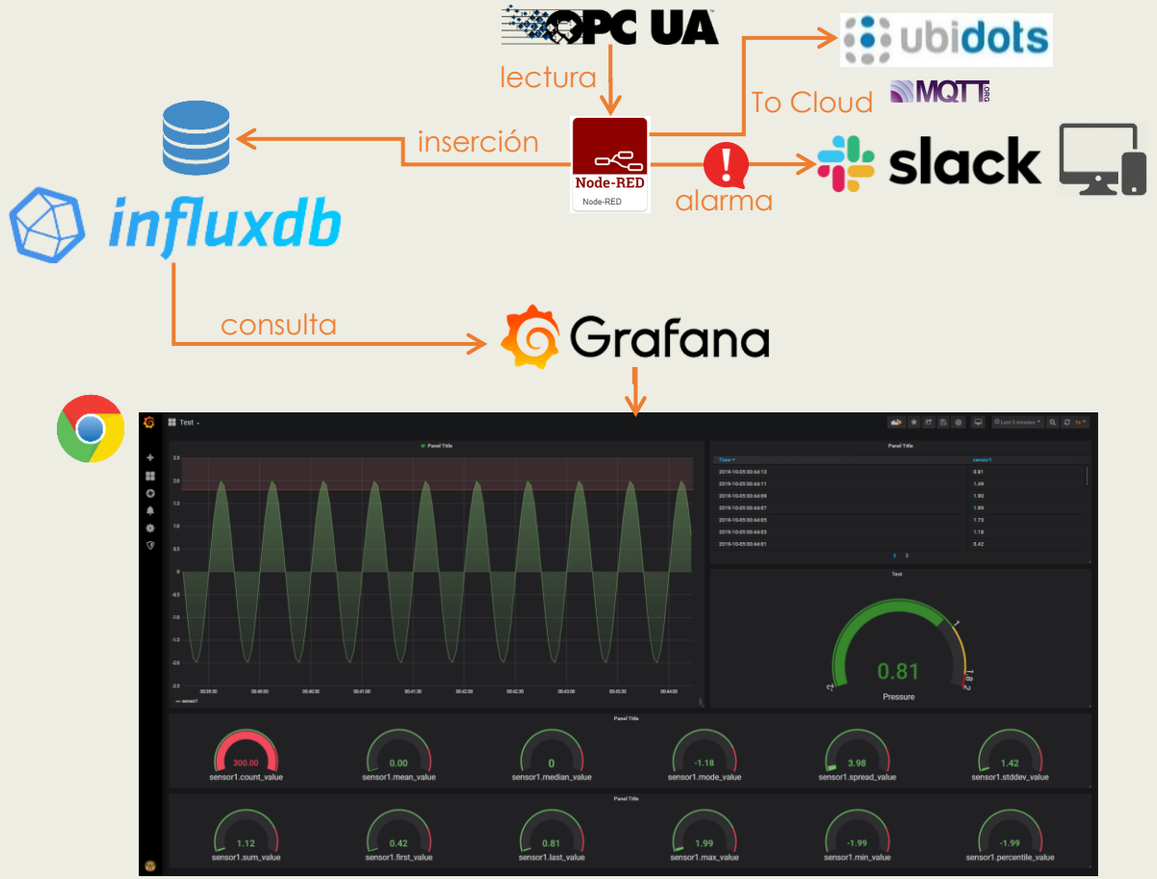
IIoT App ejemplo #3

Vamos a crear una aplicación para obtener datos de una señal de un OPC UA Server (M2M protocol), guardarlas en una base de datos de series temporales, mostrarlos en un dashboard web y enviar alarmas si supera un cierto valor.

1. Para ello vamos a utilizar **Node-Red**
2. Los datos se envían directamente al Cloud por MQTT.
3. Los datos los almacenaremos en una base de datos óptima para series temporales como es **InfluxDB**.
4. Vamos a representar estos datos con charts y gauges con la herramienta **Grafana**.
5. Enviaremos alarmas a **Slack** si el valor leído supera un umbral. Slack es una herramienta de mensajería en equipo. Utilizaremos, tanto la versión escritorio como la versión móvil.

Esta arquitectura es de las más típicas que podréis encontrar. La parte de Cloud puede cambiar a cualquiera de las que he nombrado al principio, o cualquier Broker MQTT que pueda existir.

Arquitectura #3



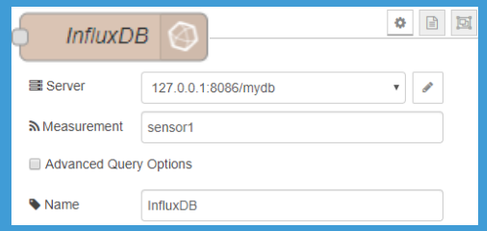
IIoT App 3

IIoT App ejemplo #3 (Node-Red)

Leemos una señal del servidor OPC UA. Es una señal sinusoidal que va de -2 a 2.

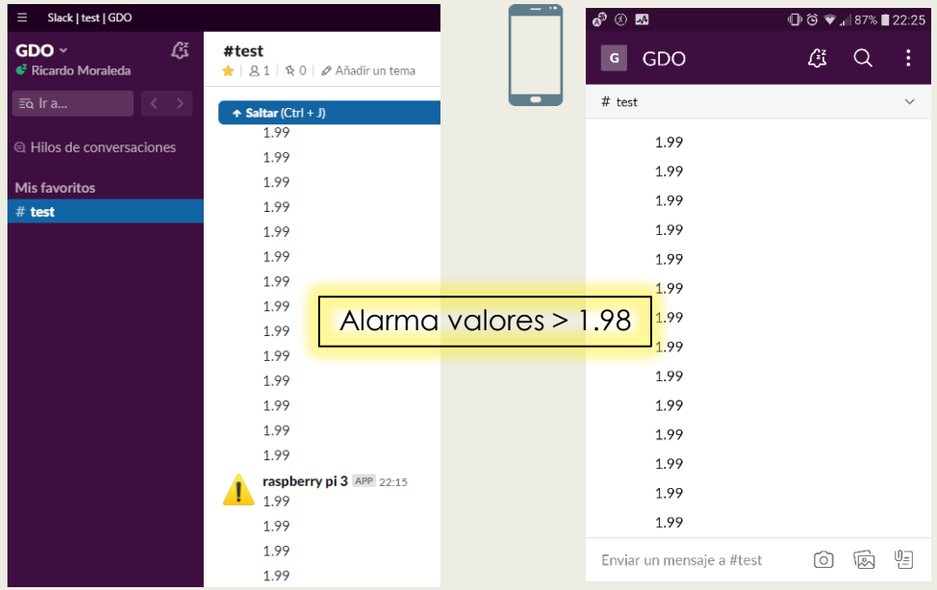
Primero acotamos a 2 decimales el valor leído y lo insertamos en la base de datos InfluxDB.

Además si el valor supera 1.98 mandamos una alarma a Slack, al canal #test de GDO. Consultable en desktop y smartphone.



IIoT App ejemplo #3 (Alarmas en Slack)

Para poder enviar estos mensajes a Slack necesitamos, a parte de tener la cuenta creada y el canal #test, una app llamada Incoming WebHooks asociada al canal #test. Este servicio es una URL del estilo: https://hooks.slack.com/services/TDDxxx/yyyyyy/zzzzzzzz



IIoT App 3

IIoT App ejemplo #3 (Grafana)

Grafana es una herramienta web open source que permite monitorizar y analizar, con gráficos espectaculares, los datos leídos de diferentes bases de datos (InfluxDB, MySQL, Graphite, etc.)

https://www.grafana.com

Una vez instalado (en mi caso en Raspberry pi 3) e iniciado se puede trabajar en la siguiente URL "http://localhost:3000".

Es super rápido montar un dashboard resultón.

Se añade el Datasource (en este caso, InfluxDB ya creada anteriormente). Se añaden varios paneles cada uno con su query (IQL) y su representación gráfica (line chart, table, gauge, etc.)

Se puede configurar el periodo de actualización (1 segundo) y además configurar alertas por si la señal supera un cierto valor.

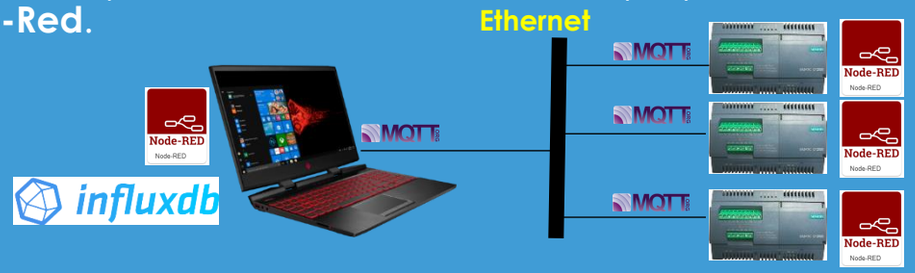
Visualización de los datos en tiempo real (Dashboard)

IIoT App 4

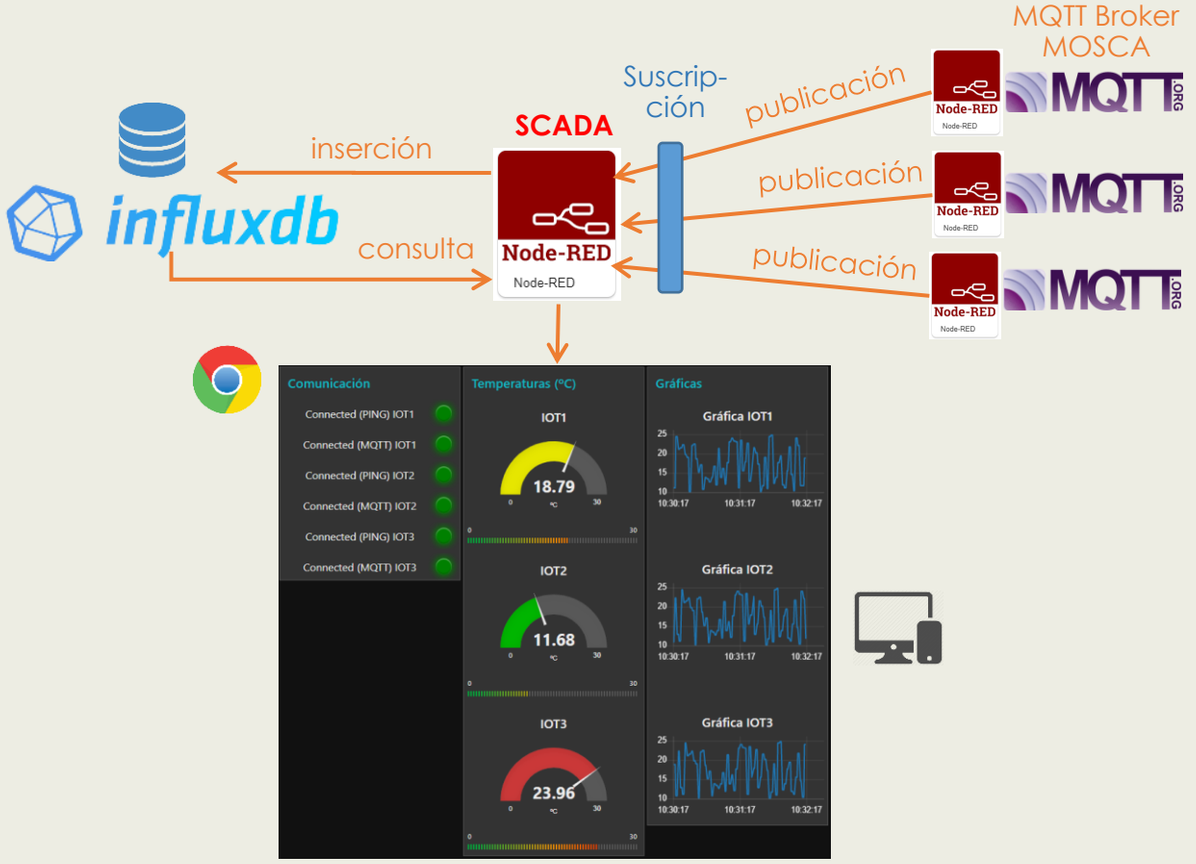
IIoT App ejemplo #4 (MQTT)

Dejando el Edge y Cloud, centrémonos en local. Finalicemos creando una aplicación centralizada, tipo SCADA (LAN) para obtener datos de 3 dispositivos embedded (IOT 2020 de Siemens) en la misma red mediante el protocolo MQTT (Message Queuing Telemetry Transport), historizarlos en una base de datos y mostrarlos mediante dashboards.

1. Para ello vamos a utilizar **Node-Red.**
2. Aplicación dispositivos embedded (Node-Red – Broker Mosca)
3. Aplicación centralizada SCADA (Node-Red – MQTT client)
4. Los datos los almacenaremos en una base de datos óptima para series temporales como es **InfluxDB**.
5. Vamos a representar estos datos con los propios dashboards de **Node-Red**.



Arquitectura #4

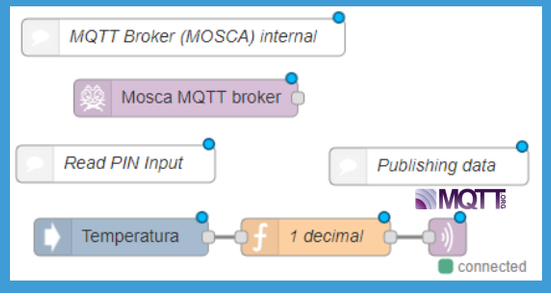


IIoT App 4

3 x Embedded devices

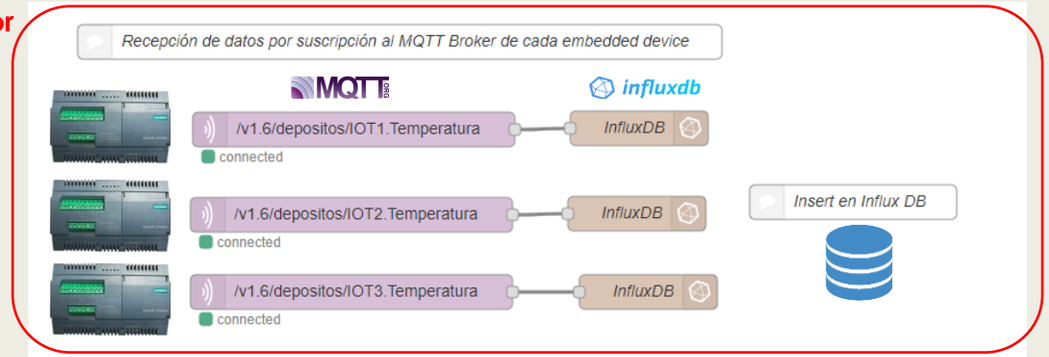
La finalidad de la aplicación en el dispositivo embedded es publicar el dato de temperatura leído de una GPIO (General Purpose Input/Output) en un MQTT Broker interno (**Mosca**) para que el PC SCADA se suscriba y pueda obtener los datos publicados.

Cada dispositivo tendrá su propia aplicación y, por tanto, publicará el valor en un topic distinto "/v1.6/depositos/IOT<*num*>.Temperatura", siendo *num* del 1 al 3.

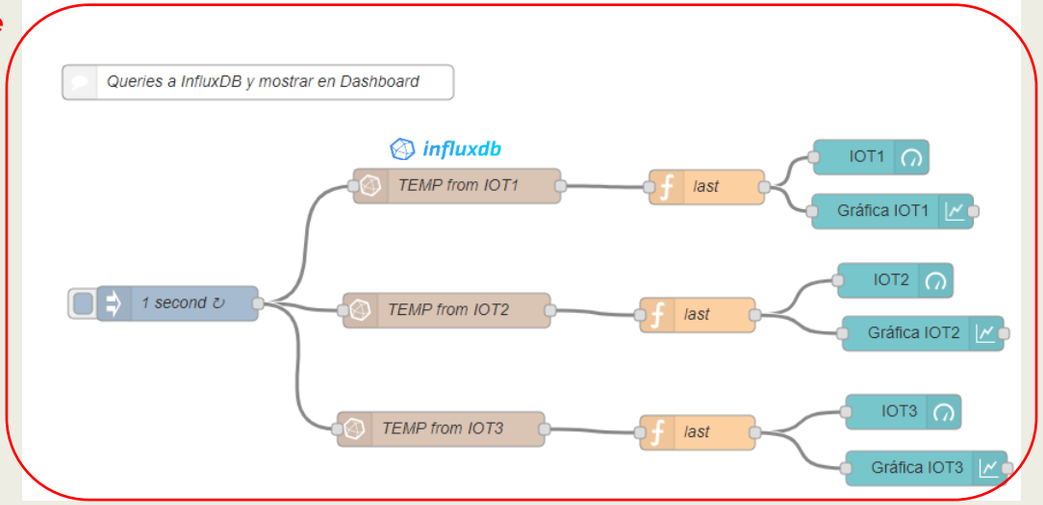


SCADA en PC

Servidor



Cliente



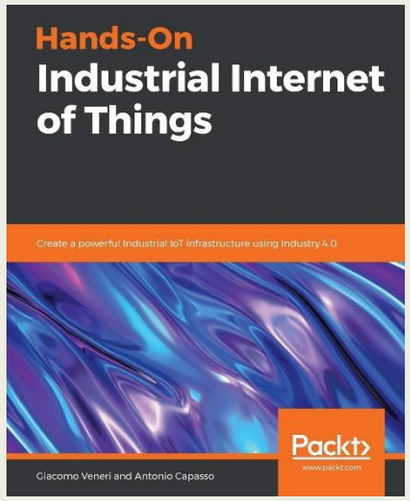
Industrial Internet Apps

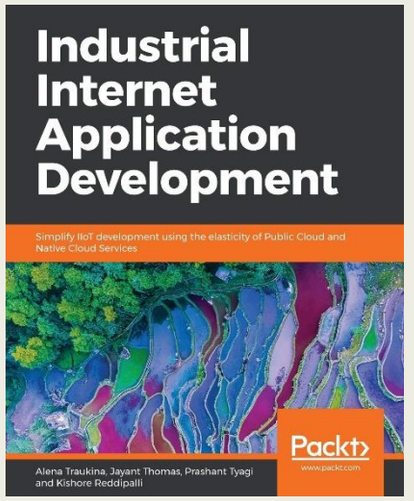
Resumen

- Industrial Internet of Things (IIoT), muchos sensores en dispositivos industriales que generan cantidades ingentes de datos.
- Es necesario tener clara la arquitectura IIoT a implantar y saber elegir lo que realmente se necesita del cloud (plataforma completa, unos servicios concretos, unas aplicaciones, repositorio de datos, etc.)
- Estos datos se tienen que transformar en información, para monitorizar, describir, predecir y prescribir a través de la analítica.
- Qué puntos de nuestra arquitectura son clave y susceptibles de securizar.
- Qué maremágnum de tecnologías son las más adecuadas para cada proyecto y qué protocolos debe incluir nuestra aplicación.
- Buenas prácticas para desarrollar, mantener y escalar las aplicaciones.
- La verdad que todo esto evoluciona vertiginosamente y ojo al modelo descentralizado, opuesto al centralizado actual, como es el basado en Blockchain, eliminando puntos únicos de fallo creando una red verdaderamente abierta, independiente y autogestionada.

Lecturas recomendadas

Recomiendo la lectura de estos 2 libros que te muestran en profundidad todo el abanico de arquitecturas, tecnologías y aplicaciones prácticas en el mundo de IIoT (Industrial Internet of Things). www.packtpub.com





Industrial Internet Apps
v.1.1 MARZO 2024

https://www.linkedin.com/in/ricardo-moraleda-gareta-9421099

https://www.linkedin.com/company/gdo-electric1996/

RICARDO MORALEDA GARETA

GDO SOFTWARE