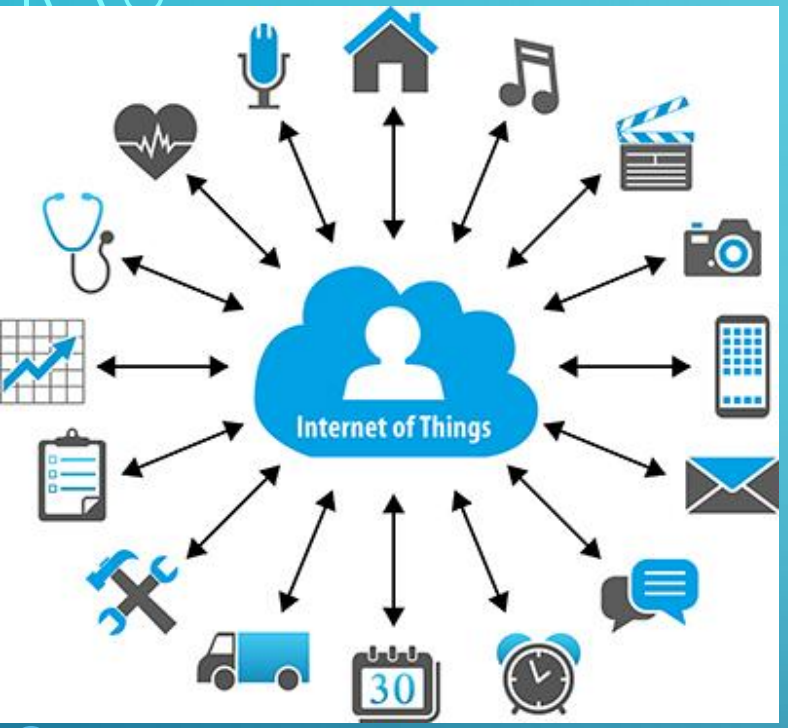
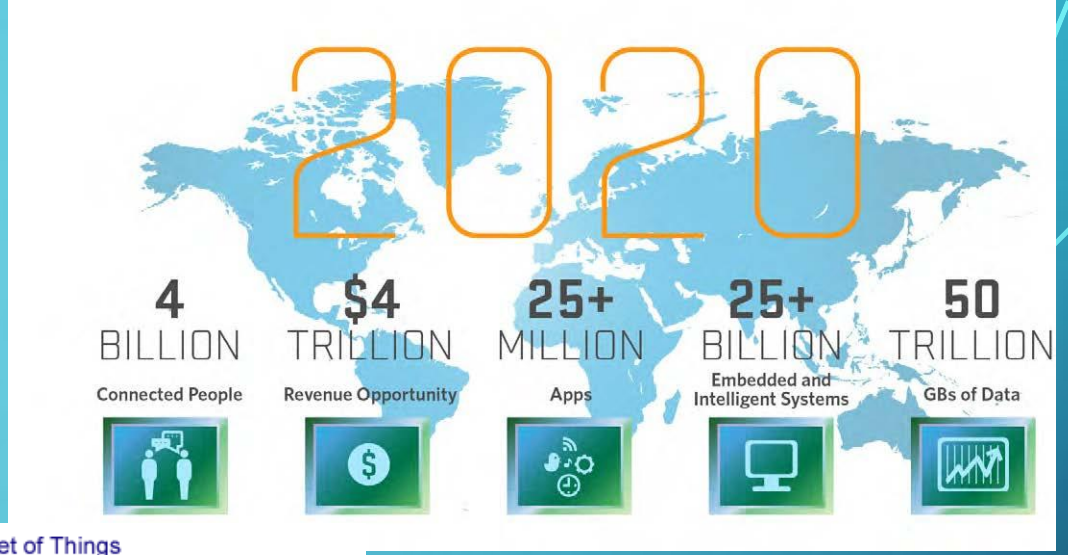
SERVER FOR IOT DEVICES AND MOBILE DEVICES USING WIFI NETWORK

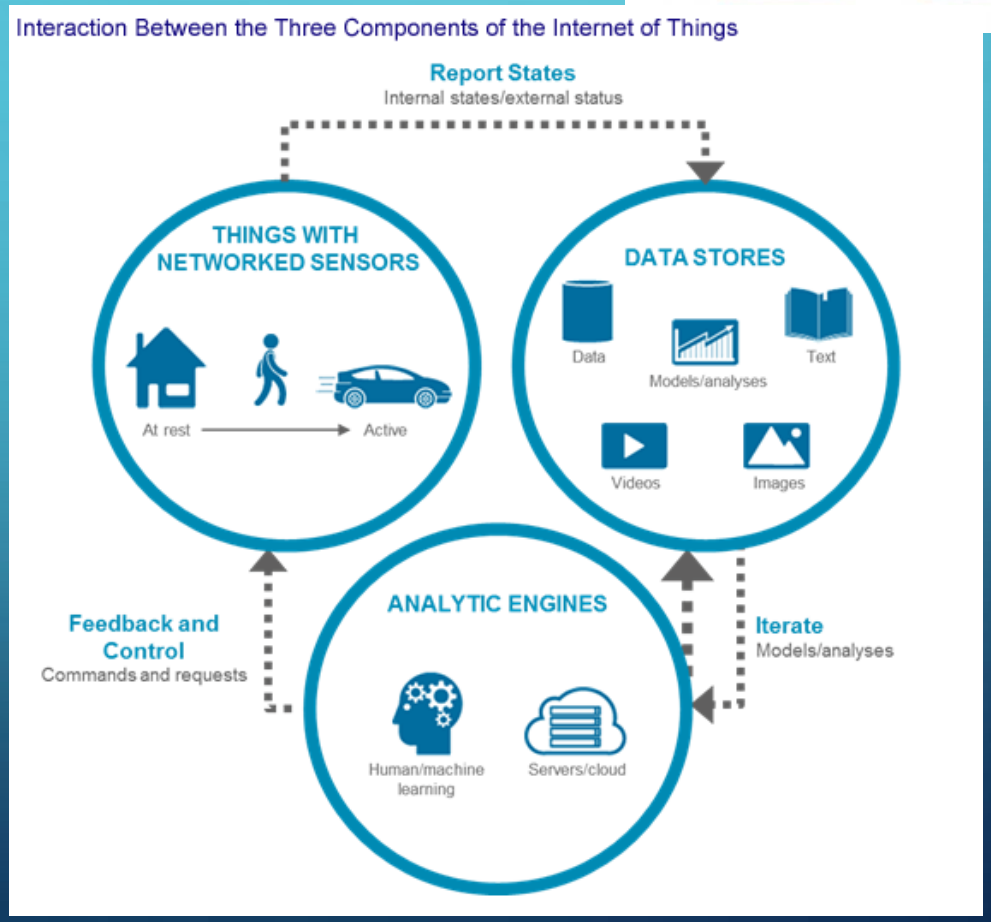
JOAQUIN BERENGUER

30 MINUTES

IMAGES OF IOT







CONTENT

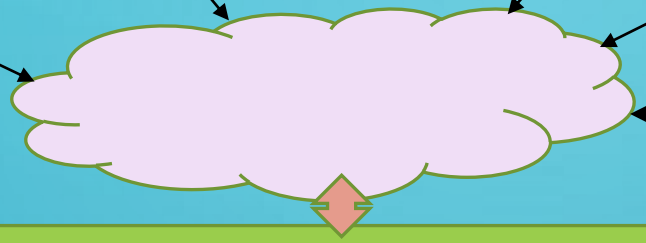
- Environment
- Architecture
- Server Functionality
- Security
- IoT Devices
- Mobile and Desktop Devices
- Events and Alarms
- Sensors, Actuators
- Future

PYTHON ENVIRONMENT : WINDOWS OR LINUX

- Python 3.5.1
- Standard Libs : os, sys, tkinter, hashlib, serial, time, datetime, signal, socket, random, threading, queue, sched, socket.io-client, flask, flask-socket.io, logger
- MySQL Lib : mysql.connector (mysql-connector-python-2.1.3.zip) (**Platform Independent (Architecture Independent), ZIP Archive, ver 2.1.3) to use Python 3.5.1, only 3.4 available as per today.) (AT : http://dev.mysql.com/downloads/connector/python/)**
- Pure Python Personal Libs : MyFunc16 (mysql functions), MyConfig (system config), ChimoFunc16 (general functions), USBFunc16 (USB Messaging).

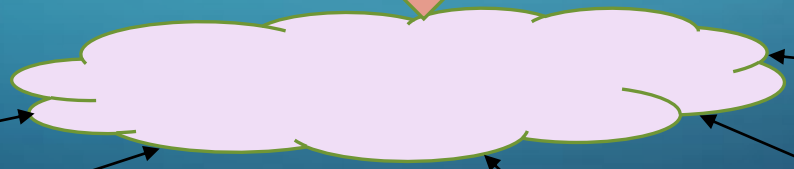
ARQUITECTURE

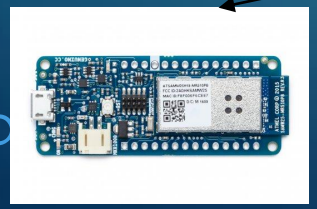
This is not a PaaS
Like Xively (LogMeIn
or ThingSpeak (Mathworks)!!!

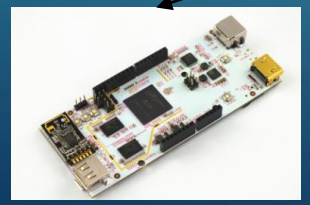


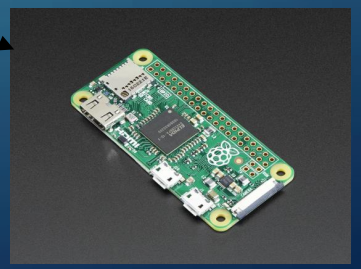
Flask + SocketIO

IoT Server









FLASK + SOCKET-IO APP.

- FrontEnd App. To show the Server Functionality
- Any Browser or OS, that support Websockets is valid.
- Flask App. Acts as several (load depending) modules connected to IoT Server
- Events from Modules are sent to the Flask App. Using SocketIO_Client Python Library.
- Messages to/from Modules are executed using http://cdn.socket.io/socket.io-1.4.5.js

```
$(document).ready(function() {
        var url = "http://" + document.domain + ":" + location.port;
        var socket = io.connect(url+"/serveriot");
        socket.on('connect', function() {
            socket.emit('msgcon', {data: 'Estoy Conectado!'});});
        $('#app-form').submit(function() {
            socket.emit('msgcli', {data: $("#modulos").val()+"&"+$("#funciones").val();});
        socket.on('msgserv', function(msg) {
            $("#12").prepend(msg.data+'\n'); });
        socket.on('eventserv', function(msg) {
            $("#13").prepend(msg.data+'\n'); });
});
```

```
socketio.emit('msgserv', {'data':resp2}, namespace="/serveriot")
```

```
@socketio.on('msgcon', namespace='/serveriot')
def RecMensServerIoT(msg):

@socketio.on('msgcli', namespace='/serveriot')
def RecMensCli(msg):

@socketio.on('srviot', namespace='/serveriot')
def RecMensServerIoT(msg):
socketio.emit('eventserv', {'data':msg.get('data')}, namespace="/serveriot")
```

IOT SERVER ARCHITECTURE

Thread Socket + Thread Queue = Device Connected

Threads are blocked waiting for a message

Server Process, waiting for TCP connections , starting threads x N

MySQL 5.7 Server or MariaDB 10.0.3

PARAMETERS TO DEFINE A DEVICE

- To register the device before being used by the end user, the IoT device is configured storing in THE DEVICE memory the following parameters, using his USB connection :
- Type of Device, Id of the Device, Serial Number, Server Name and Server Port, this 5 parameters are stored in flash memory(MKR1000) or eMMC(BBGW) or SD Card(Raspberry) of the device and in the database at the same time.
- The Serial Number NEVER is transmited between Server and Any device.
- Different Servers could be used if the number of devices is incremented, this will change the Server Port in the device.
- The architecture could have as many servers that could be needed and one database.
- Actually the server name is public using dynamic dns based on no-ip.

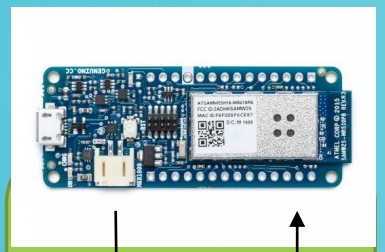
MESSAGE FORMAT

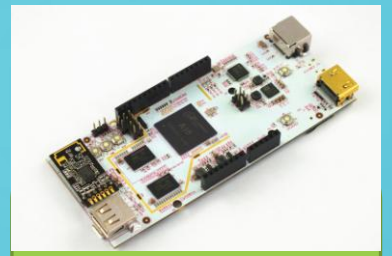
- The 3 first fields are mandatory : Device Origin, Device Destination and Message Type.
- The rest of the fields depend on the Type of Message.
- Each Field have variable length, and use a separator field.
- Each Message is finished by ASCII_EOT to indicate end of message.

SECURITY

- Used SHA256 in both parts of the communication to identify the devices registered.
- The database store in a table the devices registered with 3 columns that is needed for the connection : Type of Device, Id and Serial Number. Serial Number never is sent in any trasmission.
- In the first step Server will receive Type of Device and Id, getting the Serial Number from the Database.
- A Random Number is generated in the IoT device using srand(value of timer) with a timer value in the moment of calculation and rand() function. A number of 5 digits is generated. In the case of using Python, random lib is used.
- The device will send the digest of Serial Number + Random Number, and the random number.
- The Server using the serial number stored in the database, and the random number provided by the device, will calculate the digest and if the result is correct, the device is registered to receive and send messages, if not, the thread is terminated.
- In Python the library hashlib.sha256 is used, in C the implementation of sha256 is made based on Atmel documentation. And the usage of Cryptographic devices like ATECC508.
- The Devices are grouped by customer, only devices of the same group could send and receive messages among them.

DEVICES COMMUNICATION PROCESS





1

8

5

4

Thread 1
Origin

Thread i
Destination

Thread i + 1

Thread n

6

7

2

3

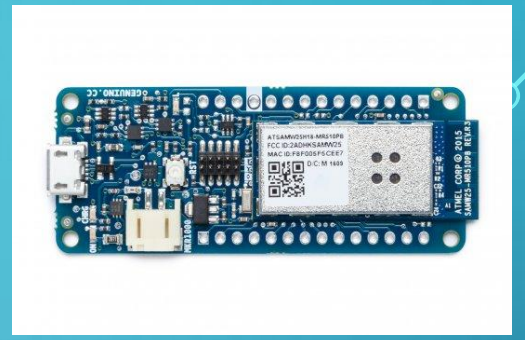
Queue1

Queue i

Queue i+1

Queue n

IOT DEVICE LOW CONSUMPTION



MKR1000

- Actually based on ATSAMD21J18 (64 pins) and ATSAMD21G18 (48 pins) from Atmel. Arm Cortex M0+, 32 bits with 256KB of Flash and 32K of RAM.
- Wifi Module : WINC1500 from Atmel.
- Crypto Device ATECC508, SHA204.
- Battery Management with MCP73831T, 1 cell Li-Po
- Possibility to add SPI Ram and Flash Memory.
- Any type of Sensor and Actuator could be used, each Project will have the own specific design.
- Atmel Studio 7 for Development, using C language. Atmel-ICE to debug and program the ATSAMD21 and Arduino 1.6.8 for MKR1000 in Basic Programs

WIFI RANGE

- In a normal laptop the power transmitted is about 15dBm (32mW), a normal receiver is using -70dBm (100pW).
- WINC1500 is transmitting 18,5dBm and receiving -98dBm, 802.11b, 1Mb/s
- With 20dBm more than 100 meters range could be made, at the worst case like at home. More than 1000 meters in open air.
- Tipical Calculation gives around 1Km for this devices in Open Air.
- Wireless Range Calculation : PathLoss=Pt-Pr-AntGain-Losses-FadeMargin, Fade Margin for a bad situation could be 25dBm, AntGain=2dBm -> PatLoss is about 90dBm
- Distance in Km = 10^((90-100,22)/20) for 2,45GHz -> about 300 meters

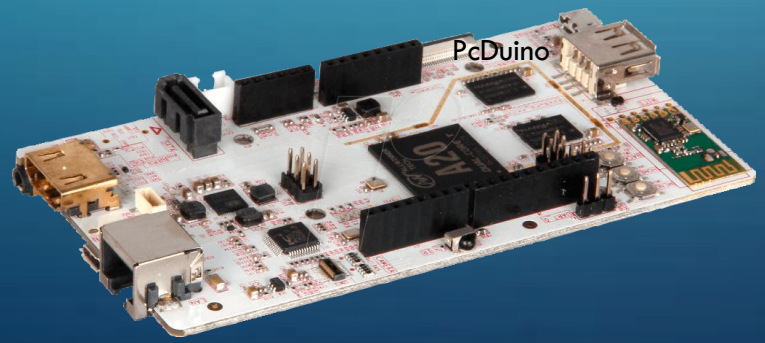
WIFI MODES

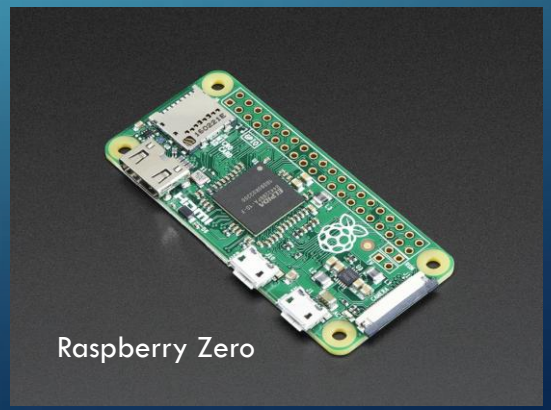
- Station (normal mode with the server), usually only this mode is used.
- Soft AP (used a server, or range extensión)
- Http Provisioning (WINC1500 only)
- Wifi Direct (WINC1500 only)

IOT ADVANCED MODULES

- Linux Embedded Boards like RaspberryPi 3, that integrate Wifi Module, could be an IoT Device for Advanced Applications (45€). Including Open CV Applications, using the camera that could be incorporated. And controlled from Python. New RaspBerry Zero + Wifi Dongle for 15€ is fantastic as well. Limited usage for Industrial, limited permissions, to be used for Media and Education
- Ubuntu Embedded Boards as PcDuinoWifi (30$) or PcDuino3 (50$), with Wifi modules, same Python usage.
- BeagleBone Green Wireless, 44$, should be used to build comercial products, specially for Industrial environment. Debian or Ubuntu OS


BeagleBone Green wireless


PcDuino


Raspberry Zero


RaspBerry3

MOBILE AND DESKTOP DEVICE

- Any web browser in any OS that support Websockets could be used.
- From any device and any OS, to use Flask + SocketIO, is the best way.
- For any type, Access to Server will have the same process as the IoT Device, and App. Will have the possibility to select which IoT Device is selected to send messages.

ALARMS AND EVENTS

- SocketIO is being used for start-stop alarms, and to define events on the modules, using the Flask Application.
- Alarms are defined by sending messages to each device or group of devices. The messages are sent by the Server. End User could implement alarms sending those messages to the server.
- The device by interrupt will capture the alarm and send a message to the Server, who will resend the message to the devices registered in the Server for that Alarm and store the message in the DB .
- Events are Scheduled messages sent by the Device to show the Users the activity in that device, this events are registered in the DB.

SENSORS

- Any type of sensor that is connected to a Device could be read. By sending messages to the corresponding device, from any mobile or desktop app. That is registered in the Server and that could Access to that device.
- Sensors like : Temperature, Ambient Humidity, Pressure, Presence, Light, Water Level, Position, Distance, Giroscope, Gravity, Rotation, Acceleration, …
- Three types of Reading, send a message to read a sensor in a determined moment, receive events or register an alarm for a sensor, and receive the Reading when a determined value have been set.

ACTUATORS

- Any write from the device, is going to be named as Actuator, into this group we could have from a led or lamp. To Motor velocity or acceleration.
- Sensor Alarm, also could activate the actuator, to avoid sending messages to the server and comeback.
- The end user Application, need to be developed in the Device and configure the server to arrive to the solution required by the customer.

FUTURE

- Software is the most important part of the Solution.
- Many Applications could be developed around IoT, we are at the beginning of a new business sector.
- Products as Xively from LogMeIn, ThingSpeak from Mathworks, Azure from Microsoft are good examples.
- Hardware will continue to integrate more parts of the Application, decreasing the Price and size at the same time.
- Sophisticated Wearables will be part of the Solution in the near future.
- Industry will take advantage as well, substituting actual solutions.


Questions

- Joaquin Berenguer
- joaquin@berentec.com
- https://github.com/joaquinbb
- @joaquinbb