



IoT Protocolo MQTT

Proyecto Final 2º ASIR

Juan Gil Estrada

ÍNDICE:

Capítulo 1. Introducción teórica.....	2
Capítulo 2. Planificación.....	8
Capítulo 3. Programas y herramientas.....	10
Capítulo 4. Instalación y configuración.....	12
Capítulo 5. Funcionamiento.....	19
Capítulo 6. Programación.....	23
Capítulo 7. Autoformación.....	28
Capítulo 8. Posibles ampliaciones y/o mejoras.....	29
Capítulo 9. Reflexión y conclusiones.....	30
Capítulo 10. Bibliografía.....	31
Manual de administración.....	32

Capítulo 1. Introducción teórica

1. ¿Qué es IoT?

Internet de las cosas (en inglés, *Internet of Things*, abreviado *IoT*) es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet. Alternativamente, Internet de las cosas es la conexión de Internet con más “cosas u objetos” que personas. También se suele conocer como Internet de *todas* las cosas o Internet *en* las cosas. Si objetos de la vida cotidiana tuvieran incorporadas etiquetas de radio, podrían ser identificados y gestionados por otros equipos, de la misma manera que si lo fuesen por seres humanos.

El concepto de Internet de las cosas fue propuesto por Kevin Ashton en el Auto-ID Center del MIT en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y tecnologías de sensores.

Por ejemplo, si los libros, termostatos, refrigeradores, la paquetería, lámparas, botiquines, partes automotrices, entre otros estuvieran conectados a Internet y equipados con dispositivos de identificación, no existirían, en teoría, artículos fuera de stock o medicinas caducadas; sabríamos exactamente la ubicación, cómo se consumen en el mundo; el extravío sería cosa del pasado y sabríamos qué está encendido o apagado en todo momento.

Con la próxima generación de aplicaciones de Internet (protocolo IPv6) se podrían identificar todos los objetos, algo que no se podía hacer con IPv4. Este sistema sería capaz de identificar instantáneamente por medio de un código a cualquier tipo de objeto.

El concepto de que los dispositivos se conectan a la red a través de señales de radio de baja potencia es el campo de estudio más activo del Internet de las cosas. Este hecho explica por qué las señales de este tipo no necesitan Wi-Fi ni Bluetooth. Sin embargo, se están investigando distintas alternativas que necesitan menos energía y que resultan más económicas, bajo el nombre de “Cheap Networks”.

Actualmente, el término Internet de las cosas se usa con una denotación de conexión avanzada de dispositivos, sistemas y servicios que va más allá del tradicional M2M (*máquina a máquina*) y cubre una amplia variedad de protocolos, dominios y aplicaciones.

1.1. Aplicaciones de IoT

Las aplicaciones para dispositivos conectados a internet son amplias. Múltiples categorizaciones han sido sugeridas, la mayoría están de acuerdo en separar las aplicaciones en tres principales ramas de aplicación: consumidores, empresarial, e infraestructura.

1.1.1. Aplicaciones de consumo

Un porcentaje creciente de los dispositivos IoT son creados para el consumo. Algunos ejemplos de aplicaciones de consumo incluyen automóviles conectados, entretenimiento, automatización del hogar, tecnología vestible, salud conectada y electrodomésticos como lavadoras, secadoras, aspiradoras robóticas, purificadores de aire, hornos, refrigeradores que utilizan Wi-Fi para monitoreo remoto.

1.1.2. Empresarial

El término "IoT empresarial" (EIoT, por sus siglas en inglés) es usado para referirse a todos los dispositivos en el ambiente de los negocios y corporativo.

1.1.3. Administración de infraestructura

El monitoreo y control de operaciones de infraestructura urbana y rural como puentes, vías férreas y parques eólicos es una aplicación clave de IoT. La infraestructura de IoT puede utilizarse para monitorear cualquier evento o cambio en las condiciones estructurales que puedan comprometer la seguridad e incrementar el riesgo. También puede utilizarse para planificar actividades de reparación y mantenimiento de manera eficiente, coordinando tareas entre diferentes proveedores de servicios y los usuarios de las instalaciones. Otra aplicación de los dispositivos de IoT es el control de infraestructura crítica, como puentes para permitir el pasaje de embarcaciones. El uso de dispositivos de IoT para el monitoreo y operación de infraestructura puede mejorar el manejo de incidentes, la coordinación de la respuesta en situaciones de emergencia, la calidad y disponibilidad de los servicios, además de reducir los costos de operación en todas las áreas relacionadas a la infraestructura. Incluso áreas como el manejo de desperdicios puede beneficiarse de la automatización y optimización que traería la aplicación de IoT.

1.1.4. Otros campos de aplicación

- ❖ Medicina y salud.
- ❖ Transporte.

2. Protocolo MQTT

Para dispositivos de Internet de las Cosas (IoT) es necesaria la conexión a Internet. La conexión a Internet permite que los dispositivos trabajen entre ellos y con servicios backend. El protocolo de red subyacente de internet es TCP/IP. MQTT (Message Queue Telemetry Transport), que está construido sobre la pila de TCP/IP, se ha convertido en el estándar para las comunicaciones de IoT.

Originariamente, MQTT fue inventado y desarrollado por IBM a finales de los 90. Su aplicación original era conectar sensores de los oleoductos con satélites. Tal como sugiere su nombre, es un protocolo de mensajería que soporta la comunicación asíncrona entre las partes. Un protocolo de mensajería asíncrona disocia al emisor y al receptor de los mensajes tanto en espacio como en tiempo, y, por lo tanto, es escalable en entornos de red no confiables. A pesar de su nombre, no tiene nada que ver con las colas de mensajería, y, en cambio, utiliza un modelo de publicación y suscripción. A finales del 2014, se convirtió oficialmente en un estándar abierto de OASIS, y se soporta en lenguajes de programación populares mediante la utilización de varias implementaciones de código abierto.

2.1. Por qué MQTT

MQTT es un protocolo de red liviano y flexible que logra el equilibrio adecuado para los desarrolladores de IoT:

- ❖ El protocolo liviano le permite implementarse en hardware de dispositivos altamente limitados y en redes con ancho de banda de alta latencia/limitado.
- ❖ Su flexibilidad hace que pueda soportar varios escenarios de aplicaciones para dispositivos y servicios de IoT.

Para entender por qué MQTT es tan adecuado para los desarrolladores de IoT, examinemos por qué otros protocolos de red populares han fallado con IoT.

La mayor parte de los desarrolladores ya están familiarizados con los servicios web de HTTP. Así, ¿por qué no hacer que los dispositivos de IoT se conecten con servicios web? El dispositivo podría enviar sus datos como una solicitud HTTP y recibir actualizaciones del sistema como la respuesta de HTTP. Este patrón de solicitud y respuesta tiene algunas limitaciones graves:

- ❖ HTTP es un protocolo sincronizado. El cliente espera a que el servidor responda. Los navegadores web tienen este requisito, que trae consigo el costo de una mala escalabilidad. En el mundo de IoT, el gran número de dispositivos y muy probablemente una red no confiable, o de alta latencia, han hecho que la comunicación sincronizada sea problemática. Un protocolo de mensajería asíncrona es mucho más adecuado para las aplicaciones de IoT. Los sensores pueden enviar lecturas y dejar que la red descubra la ruta y el momento óptimos para la entrega de dispositivos y servicios de destino.

- ❖ El HTTP tiene una dirección. El cliente debe iniciar una conexión. En una aplicación de IoT los dispositivos y los sensores son normalmente los clientes, lo que significa que no pueden recibir comandos de la red de forma pasiva.
- ❖ HTTP es un protocolo 1-1. El cliente hace una solicitud y el servidor responde. Transmitir un mensaje a todos los dispositivos de la red, lo que es un caso de uso habitual en las aplicaciones IoT, es algo difícil y caro.
- ❖ HTTP es un protocolo pesado con muchas cabeceras y reglas. No es adecuado para redes congestionadas.

Por las razones anteriores, la mayor parte de los sistemas escalables de alto rendimiento utilizan un bus de mensajería asíncrona, en vez de servicios web, para intercambiar datos internamente. De hecho, el protocolo de mensajería que se utiliza más habitualmente en los sistemas de middleware empresarial se llama AMQP (Advanced Message Queuing Protocol). Sin embargo, en entornos de alto rendimiento, el poder de la computación y la latencia de la red no son generalmente una preocupación. AMQP está diseñado para la confiabilidad y la interoperabilidad en aplicaciones empresariales. Tiene un amplio conjunto de funciones, pero no es adecuado para aplicaciones de IoT con recursos limitados.

Además de AMQP, hay otros protocolos de mensajería populares. Por ejemplo, el XMPP (Extensible Messaging and Presence Protocol) es un protocolo de mensajería instantánea (IM) peer-to-peer. Tiene muchas funciones que soportan casos de uso de IM, como los adjuntos de presencia y de medios de comunicación. Comparado con MQTT, requiere muchos más recursos tanto en el dispositivo como en la red.

Así que, ¿qué es lo que hace que MQTT sea tan liviano y flexible? Una de las principales funciones del protocolo MQTT es su modelo de publicación y suscripción. Como con todos los protocolos de mensajería, éste desacopla los datos del consumidor y del publicador.

2.2. El Modelo de publicación y suscripción

El protocolo MQTT define los tipos de entidades en la red: un intermediario de mensajes y un número de clientes. El intermediario es un servidor que recibe todos los mensajes de los clientes y luego los redirige a clientes de destinos relevantes. Un cliente es cualquier cosa que pueda interactuar con el intermediario para enviar y recibir mensajes. Un cliente puede ser un sensor de IoT en el campo o una aplicación del centro de datos que procesa datos de IoT.

- ❖ El cliente se conecta con el intermediario. Se puede suscribir a cualquier "topic/tema" de mensajes de intermediario. Esta conexión puede ser una conexión TCP/IP simple o una conexión TLS cifrada para mensajes confidenciales.
- ❖ El cliente publica el mensaje, sobre un topic/tema, enviando el mensaje y el topic/tema al intermediario.

- ❖ Después, el intermediario redirige el mensaje a todos los clientes que están suscritos a ese topic/tema.

Ya que los mensajes MQTT están organizados por topics/temas, el desarrollador de aplicaciones tiene la flexibilidad de especificar que ciertos clientes sólo puedan interactuar con determinados mensajes.

Al mismo tiempo, MQTT es liviano. Tiene una cabecera simple para especificar el tipo de mensaje, un tema basado en texto y, a continuación, una carga útil binaria y arbitraria. La aplicación puede utilizar cualquier formato de datos para la carga útil, como: JSON, XML, cifrado binario o Base64, siempre que los clientes destino puedan analizar la carga útil.

2.3. Aplicaciones en el mundo real

Hay varios proyectos que implementan MQTT. Los ejemplos son:

- ❖ Facebook Messenger. Facebook ha utilizado aspectos de MQTT en Facebook Messenger para chat en línea.
- ❖ IECC Scalable, la última versión de DeltaRail de su sistema de control de señalización IECC utiliza MQTT para comunicaciones dentro de las diversas partes del sistema y otros componentes del sistema de señalización.
- ❖ La plataforma EVERYTHING IoT usa MQTT como un protocolo M2M para millones de productos conectados.
- ❖ Amazon Web Services anunció Amazon IoT basado en MQTT en 2015
- ❖ La especificación estándar de API SensorThings del Consorcio geoespacial abierto tiene una extensión MQTT en el estándar como enlace de protocolo de mensaje adicional.
- ❖ Los servicios de OpenStack Upstream Infrastructure están conectados por un bus de mensajes unificados MQTT con Mosquitto como agente de MQTT.
- ❖ Adafruit lanzó un servicio en la nube gratuito MQTT para experimentadores y estudiantes de IoT llamado Adafruit IO en 2015.
- ❖ Microsoft Azure IoT Hub utiliza MQTT como su protocolo principal para mensajes de telemetría.
- ❖ XIM, Inc. lanzó un cliente MQTT llamado MQTT Buddy en 2017. Es una aplicación MQTT para Android e iOS.
- ❖ Node-RED admite nodos MQTT a partir de la versión 0.14, para configurar correctamente las conexiones TLS.

- ❖ La plataforma de automatización del hogar de software de código abierto Home Assistant está habilitada para MQTT y ofrece cuatro opciones para los corredores de MQTT.
- ❖ El marco de automatización del hogar Pimatic para Raspberry Pi y basado en Node.js ofrece el complemento MQTT que brinda soporte completo para el protocolo MQTT.
- ❖ McAfee OpenDXL está basado en MQTT con mejoras en los intermediarios de mensajería para que puedan comprender intrínsecamente el formato de mensaje DXL en apoyo de características avanzadas tales como servicios, mensajes de solicitud / respuesta (punto a punto), conmutación por error del servicio y servicio zonas.

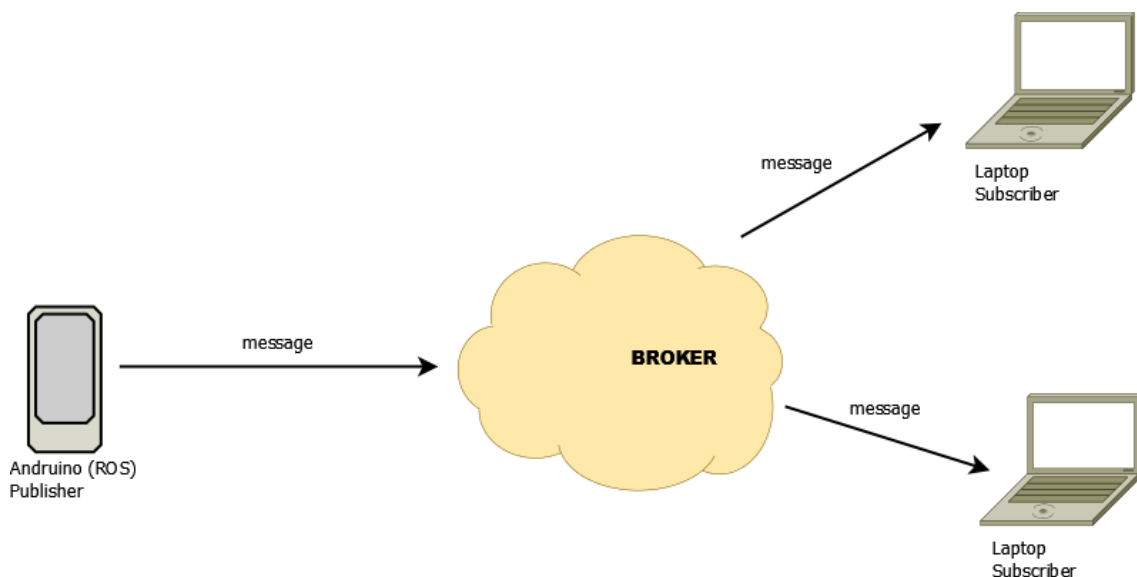
2.4. Comparación de Implementaciones MQTT

Name	Developed by	Language	Type	First release date	Last release	Last release date	License
Adafruit IO	Adafruit	Ruby on Rails, Node.js ^[31]	Client	?	2.0.0 ^[32]	?	?
M2Mqtt	eclipse	C#	Client	2017-05-20	4.3.0.0 ^[33]	2017-05-20	Eclipse Public License 1.0
Machine Head	ClojureWerkz Team	Clojure	Client	2013-11-03	1.0.0 ^[34]	2017-03-05	Creative Commons Attribution 3.0 Unported License
moquette	Selva, Andrea	Java	Broker	2015-07-08	0.10 ^[35]	2017-06-30	Apache License 2.0
Mosquitto	eclipse	C, Python	Broker and client	2009-12-03	1.4.15 ^[36]	2018-02-27	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD)
Paho MQTT	eclipse	C, C++, Java, Javascript, Python, Go	Client	2014-05-02	1.3.0 ^[37]	2017-06-28	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD) ^[38]
SharkMQTT	Real Time Logic	C	Client	2015-11-06	1.5 ^[39]	2017-10-08	Proprietary License
VerneMQ	VerneMQ	Erlang/OTP	Broker		1.3.2 ^[40]	2018-05-11	Apache License 2.0
wolfMQTT	wolfSSL	C	Client	2015-11-06	0.14 ^[41]	2017-11-22	GNU Public License, version 2
MQTTRoute	Bevywise Networks	C, Python	Broker	2017-04-25	1.0 ^[42]	2017-12-19	Proprietary License ^[43]
HiveMQ	dc-square GmbH	Java	Broker	2013-03-26	3.4.0 ^[44]	2017-05-09	Proprietary License
SwiftMQ	IIT Software GmbH	Java	Broker	2000-07-01	11.1.0 ^[45]	2018-06-06	Proprietary License

Capítulo 2. Planificación

1. Planificación inicial.

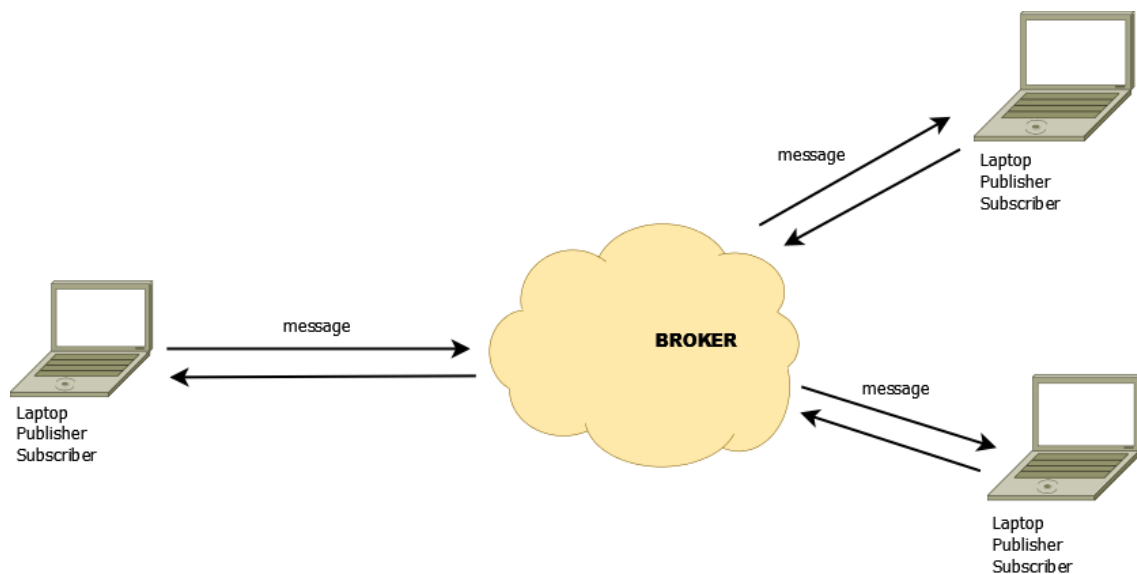
Instalar y configurar un servidor broker MQTT/Iot Hub (Kapua), de forma que varios dispositivos iot a través de un gateway puedan subscribirse y publicar datos a través de Internet. Además los datos recibidos por el broker se almacenarán en una base de datos, y se deberán definir jobs en función de los datos, y realizar análisis de los datos. En este proyecto los dispositivos conectados serán robots móviles (andruino) que se conectarán usando ROS a un pc, el cual que actúa de rosmaster en una red local, (esta parte está resuelta). Pero, para garantizar la conectividad de estos clientes con el lot Hub usando el protocolo MQTT a través del gateway, el gateway deberá traducir los mensajes ROS a mensajes MQTT y viceversa, lo cual deberá ser programado en Python por el alumno.



En el esquema podemos ver como el Andruino es el Publisher (en nuestro caso publicará los datos del acelerómetro del móvil) y los laptops son los subscribers que se encargarán de recibir los datos y guardarlos en una base de datos.

2. Planificación real ejecutada.

Instalar y configurar un servidor bróker MQTT/IoT Hub (Mosquitto), de forma que varios dispositivos IoT a través de un Gateway puedan subscribirse y publicar datos a través de Internet. Además los datos recibidos por el bróker se almacenaran en una base de datos, en un archivo txt y se representarán gráficamente. En este proyecto los dispositivos conectados serán Máquinas virtuales y equipos. Para garantizar la conectividad de estos clientes con el IoT Hub usando el protocolo MQTT a través de Gateway, el Gateway deberá traducir los mensajes a MQTT, lo cual deberá ser programado en Python por el alumno.



En el esquema podemos ver como los 3 portátiles actúan tanto como Publisher como Subscriber, los datos recibidos (Nombre, fecha y hora de la máquina que publica y un número aleatorio generado por el módulo de Python "random"), se almacenaran en una base de datos, en un archivo de texto y se representarán gráficamente.

Capítulo 3. Programas y herramientas

1. Eclipse Mosquitto

Eclipse Mosquitto es un intermediario de mensajes de código abierto (licencia EPL / EDL) que implementa el protocolo MQTT versiones 3.1 y 3.1.1. Mosquitto es liviano y es adecuado para usar en todos los dispositivos, desde computadoras de una sola placa de baja potencia hasta servidores completos.

El protocolo MQTT proporciona un método liviano para llevar a cabo mensajes utilizando un modelo de publicación / suscripción. Esto lo hace adecuado para mensajería de Internet of Things, como con sensores de baja potencia o dispositivos móviles como teléfonos, computadoras integradas o microcontroladores.

2. Paho MQTT

El proyecto Eclipse Paho proporciona implementaciones de cliente de código abierto de los protocolos de mensajería MQTT y MQTT-SN destinados a aplicaciones nuevas, existentes y emergentes para Internet de las cosas (IoT).

3. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

3.1. Módulos utilizados

- ❖ **Random:** este módulo implementa generadores de números pseudoaleatorios para diversas distribuciones.
- ❖ **Commands:** este módulo contiene funciones de contenedor para `os.popen ()` que toman un comando del sistema como una cadena y devuelven cualquier salida generada por el comando y, opcionalmente, el estado de salida.

- ❖ **MySQLdb:** es una interfaz compatible con subprocessos para el popular servidor de base de datos MySQL que proporciona la API de base de datos de Python.
- ❖ **Matplotlib.pyplot:** es una biblioteca de trazado 2D de Python que produce figuras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas.

4. UFW

Uncomplicated Firewall (ufw) es un cortafuegos diseñado para ser de fácil uso desarrollado por Ubuntu. Utiliza la línea de comandos para configurar las iptables usando un pequeño número de comandos simples. Ufw está escrito en python y es un programa para GNU/Linux.

5. VirtualBox

Oracle VM VirtualBox es un software de virtualización para arquitecturas x86/amd64. Actualmente es desarrollado por Oracle Corporation como parte de su familia de productos de virtualización. Por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como «sistemas invitados», dentro de otro sistema operativo «anfitrión», cada uno con su propio ambiente virtual. Entre los sistemas operativos soportados (en modo anfitrión) se encuentran GNU/Linux, Mac OS X, OS/2 Warp, Microsoft Windows, y Solaris/OpenSolaris, y dentro de ellos es posible virtualizar los sistemas operativos FreeBSD, GNU/Linux, OpenBSD, OS/2 Warp, Windows, Solaris, MS-DOS y muchos otros.

6. Open Nebula

OpenNebula es una plataforma para computación en la nube orientado a centros de datos distribuidos y heterogéneos, proporcionando la infraestructura virtual para construir nubes privadas, públicas, e implementaciones híbridas de infraestructura como servicio (IaaS). OpenNebula es software de fuente abierta amparado en la Licencia apache 2.

7. Dia

Dia es una aplicación informática de propósito general para la creación de diagramas, creada originalmente por Alexander Larsson, y desarrollada como parte del proyecto GNOME . Está concebido de forma modular, con diferentes paquetes de formas para diferentes necesidades.

Capítulo 4. Instalación y configuración

1. Instalación Servidor

Crearemos en OpenNebula una máquina Ubuntu

Primero instalaremos Mosquitto en el servidor

```
# apt-get install mosquitto
```

```
root@ubuntu:~# apt-get install mosquitto_
```

Comprobamos que el servicio corre perfectamente

```
# systemctl status mosquitto
```

```
root@ubuntu:~# systemctl status mosquitto
* mosquitto.service - LSB: mosquitto MQTT v3.1 message broker
   Loaded: loaded (/etc/init.d/mosquitto; bad; vendor preset: enabled)
   Active: active (running) since Tue 2018-05-29 19:46:14 UTC; 1 weeks 4 days ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 1
   Memory: 1.7M
         CPU: 6min 35.447s
   CGroup: /system.slice/mosquitto.service
           └─1344 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 29 19:46:14 ubuntu systemd[1]: Starting LSB: mosquitto MQTT v3.1 message bro
May 29 19:46:14 ubuntu mosquitto[1324]: * Starting network daemon: mosquitto
May 29 19:46:14 ubuntu mosquitto[1324]: ...done.
May 29 19:46:14 ubuntu systemd[1]: Started LSB: mosquitto MQTT v3.1 message brok
```

Ahora pasaremos a usar UFW para cerrar puertos, comprobamos que puerto/puertos usa Mosquitto

```
# netstat -lp --inet
```

```
root@ubuntu:~# netstat -lp --inet
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 *:1883                  *:*                     LISTEN
1344/mosquitto
```

Al ver que sólo usa el 1883, denegamos el acceso de entrada de todos los puertos exepto del 1883 con UFW

ufw enable (activamos ufw)

ufw default deny incoming (denegamos todo el tráfico entrante)

ufw allow 1883 (damos acceso a todo el tráfico del puerto 1883)

ufw status (vemos las reglas que tenemos activas)

```
root@ubuntu:~# ufw enable
Firewall is active and enabled on system startup
root@ubuntu:~# ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
root@ubuntu:~# ufw allow 1883
Skipping adding existing rule
Skipping adding existing rule (v6)
root@ubuntu:~# ufw status
Status: active

To Action From
--
1883 ALLOW Anywhere
1883 (v6) ALLOW Anywhere (v6)
```

Por último cambiamos la password del usuario root por seguridad

passwd

```
root@ubuntu:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

2. Instalación Máquinas

Este procedimiento lo haremos en nuestras 3 máquinas.

Primero instalamos mosquitto

apt-get install mosquitto

```
root@hulfllys-Desktop:/home/hulfllys# apt-get install mosquitto
```

Comprobamos que el servicio funciona correctamente

systemctl status mosquitto

```
root@hulflys-Desktop:/home/hulflys# systemctl status mosquitto
● mosquitto.service - LSB: mosquitto MQTT v3.1 message broker
   Loaded: loaded (/etc/init.d/mosquitto; bad; vendor preset: enabled)
   Active: active (running) since dom 2018-06-10 13:00:15 CEST; 32s ago
     Docs: man:systemd-sysv-generator(8)
    CGroup: /system.slice/mosquitto.service
            └─3286 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

jun 10 13:00:15 hulflys-Desktop systemd[1]: Starting LSB: mosquitto MQTT v3.1 message broker...
jun 10 13:00:15 hulflys-Desktop mosquitto[3274]: * Starting network daemon: mosquitto
jun 10 13:00:15 hulflys-Desktop mosquitto[3274]: ...done.
jun 10 13:00:15 hulflys-Desktop systemd[1]: Started LSB: mosquitto MQTT v3.1 message broker.
```

Instalamos python3-pip

apt-get install python3-pip

```
root@hulflys-Desktop:/home/hulflys# apt-get install python3-pip
```

Luego instalamos nuestro cliente Paho-MQTT, nos pide actualizar la versión de pip.

pip3 install paho-mqtt

```
root@hulflys-Desktop:/home/hulflys# pip3 install paho-mqtt
Collecting paho-mqtt
  Downloading https://files.pythonhosted.org/packages/2a/5f/cf14b8f9f0ed1891cda893a2a7d1d6fa23de2a9fb4832f05cef02b79d01f/paho-mqtt-1.3.1.tar.gz (80kB)
    100% |#####| 81kB 684kB/s
Building wheels for collected packages: paho-mqtt
  Running setup.py bdist_wheel for paho-mqtt ... done
  Stored in directory: /root/.cache/pip/wheels/38/ca/67/86c7e4acc659ce5ab74cbb8cc38de50c90ed4f827133e36994
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.3.1
You are using pip version 8.1.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Al intentar actualizar, nos manda el siguiente mensaje

pip install --upgrade pip

```
root@hulflys-Desktop:/home/hulflys# pip install --upgrade pip
The program 'pip' is currently not installed. You can install it by typing:
apt install python-pip
```

Instalamos Python-pip

apt install python-pip

```
root@hulflys-Desktop:/home/hulflys# apt install python-pip
```

Ahora si nos deja actualizar pip correctamente

pip install --upgrade pip

```
root@hulflys-Desktop:/home/hulflys# pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/0f/74/ecdl3431bcc456ed390b44c8a6e917c1820365cbebc6a8974d1cd045ab4/pip-10.0.1-py2.py3-none-any.whl (1.3MB)
    100% |#####| 1.3MB 605kB/s
Installing collected packages: pip
  Found existing installation: pip 8.1.1
    Not uninstalling pip at /usr/lib/python2.7/dist-packages, outside environment /usr
Successfully installed pip-10.0.1
```

Volvemos a instalar Paho-MQTT para que lo haga con la última versión

```
# pip install paho-mqtt
```

```
root@hulflys-Desktop:/home/hulflys# pip install paho-mqtt
Collecting paho-mqtt
  Using cached https://files.pythonhosted.org/packages/2a/5f/cf14b8f9f8ed1891cda893a2a7d1d6fa23de2a9fb4832f05cef02b79d01f/paho-mqtt-1.3.1.tar.gz
Building wheels for collected packages: paho-mqtt
  Running setup.py bdist_wheel for paho-mqtt ... done
  Stored in directory: /root/.cache/pip/wheels/38/ca/67/86c7e4acc659ce5ab74cbb8cc38de50c90ed4f827133e36994
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.3.1
```

Ahora le damos permisos a todos los archivos Python que vamos a utilizar

```
# chmod +x pub.py
```

```
# chmod +x mysql_sub.py
```

```
# chmod +x txt_sub.py
```

```
# chmod +x txt_sub.py
```

```
# chmod +x graf_sub.py
```

```
root@hulflys-Desktop:/home/hulflys# cd Desktop/
root@hulflys-Desktop:/home/hulflys/Desktop# chmod +x pub.py
root@hulflys-Desktop:/home/hulflys/Desktop# chmod +x mysql_sub.py
root@hulflys-Desktop:/home/hulflys/Desktop# chmod +x txt_sub.py
root@hulflys-Desktop:/home/hulflys/Desktop# chmod +x file.txt
root@hulflys-Desktop:/home/hulflys/Desktop# chmod +x graf_sub.py
```

Al intentar iniciar mysql_sub.py vemos que nos hace falta instalar un módulo de Python para poder ejecutarlo

```
# python mysql_sub.py
```

```
root@hulflys-Desktop:/home/hulflys/Desktop# python mysql_sub.py
Traceback (most recent call last):
  File "mysql_sub.py", line 4, in <module>
    import MySQLdb
ImportError: No module named MySQLdb
```

Para

ello:

```
# apt-get install python-pip python-dev libmysqlclient-dev
```

```
# pip install mysqlclient
```

```
# pip install MySQL-python
```

```
root@hulflys-Desktop:/home/hulflys/Desktop# apt-get install python-pip python-dev libmysqlclient-dev
root@hulflys-Desktop:/home/hulflys/Desktop# pip install mysqlclient
Collecting mysqlclient
  Using cached https://files.pythonhosted.org/packages/6f/86/bad31f1c1bb0cc99e88ca2adb7cb5c71f7a6540c1bb001480513de76a931/mysqlclient-1.3.12.tar.gz
Building wheels for collected packages: mysqlclient
  Running setup.py bdist_wheel for mysqlclient ... done
  Stored in directory: /root/.cache/pip/wheels/50/c7/31/81a516762c8e9324f2b1d1fffc1e84b9f07224fe3707956f6e1
Successfully built mysqlclient
Installing collected packages: mysqlclient
Successfully installed mysqlclient-1.3.12
root@hulflys-Desktop:/home/hulflys/Desktop# pip install MySQL-python
Collecting MySQL-python
  Using cached https://files.pythonhosted.org/packages/a5/e9/51b544da85a36a68debe7a7091f068d802fc515a3a202652828c73453cad/MySQL-python-1.2.5.zip
Building wheels for collected packages: MySQL-python
  Running setup.py bdist_wheel for MySQL-python ... done
  Stored in directory: /root/.cache/pip/wheels/07/d2/5f/314860e4cb53a44bf0ee0d051d4b34465e4b4f9e9de6d42f42
Successfully built MySQL-python
Installing collected packages: MySQL-python
Successfully installed MySQL-python-1.2.5
```


Ahora ya ejecuta correctamente mysql_sub.py

```
# python mysql_sub.py
```

```
root@hulflys-Desktop:/home/hulflys/Desktop# python mysql_sub.py
Connected with result code 0
```

Vemos que ejecuta correctamente txt_sub.py

```
root@hulflys-Desktop:/home/hulflys/Desktop# python txt_sub.py
Connected with result code 0
```

Al ejecutar graf_sub.py vemos que no tenemos el módulo matplotlib.pyplot

```
# python graf_sub.py
```

```
root@hulflys-Desktop:/home/hulflys/Desktop# python graf_sub.py
Traceback (most recent call last):
  File "graf_sub.py", line 4, in <module>
    import matplotlib.pyplot as plt
ImportError: No module named matplotlib.pyplot
```

Instalamos

el

módulo:

```
# apt-get install python-matplotlib
```

```
root@hulflys-Desktop:/home/hulflys/Desktop# apt-get install python-matplotlib
```

Al ejecutar graf_sub.py vemos que funciona correctamente

```
# Python graf_sub.py
```

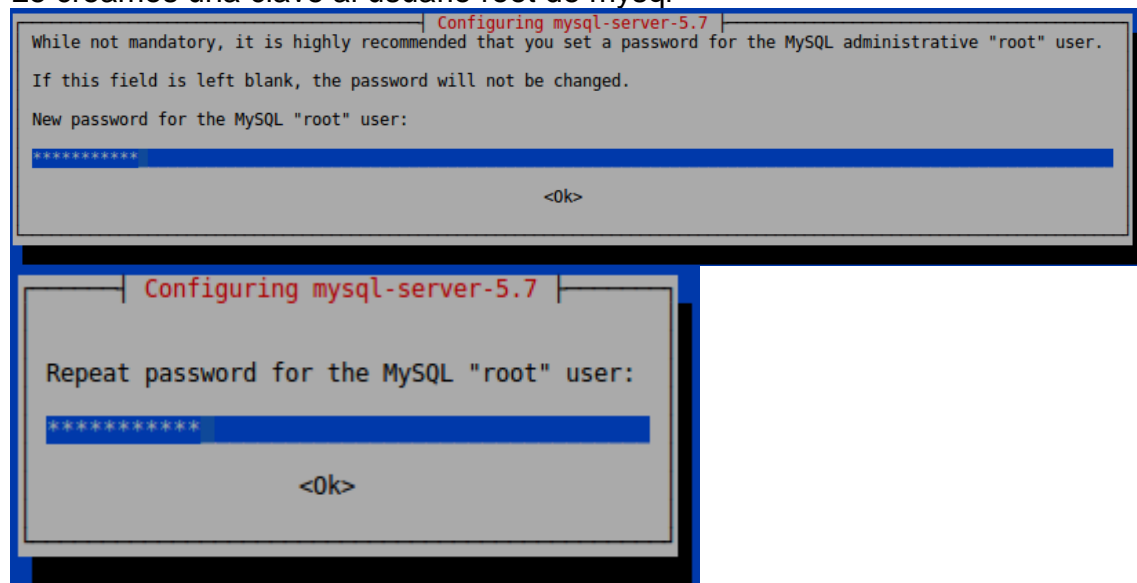
```
root@hulflys-Desktop:/home/hulflys/Desktop# python graf_sub.py
Connected with result code 0
```

Instalamos mysql

```
# apt-get install mysql-server
```

```
root@hulflys-Desktop:/home/hulflys# apt-get install mysql-server
```

Le creamos una clave al usuario root de mysql



Comprobamos que el servicio funciona correctamente

systemctl status mysql.service

```
root@hulfllys-Desktop:/home/hulfllys# systemctl status mysql.service
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
   Active: active (running) since dom 2018-06-10 14:01:12 CEST; 51s ago
     Main PID: 3130 (mysqld)
       CGroup: /system.slice/mysql.service
               └─3130 /usr/sbin/mysqld

jun 10 14:01:11 hulfllys-Desktop systemd[1]: Starting MySQL Community Server...
jun 10 14:01:12 hulfllys-Desktop systemd[1]: Started MySQL Community Server.
```

3. Configuración MySQL

Entramos en mysql como root

mysql -u root -p

```
root@hulfllys-Desktop:/home/hulfllys# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

Creamos un usuario llamado "user"

CREATE USER 'user'@'localhost' IDENTIFIED BY '2asirtriana';

```
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY '2asirtriana';
Query OK, 0 rows affected (0,00 sec)
```

Le damos todos los permisos al usuario que acabamos de crear

GRANT ALL PRIVILEGES ON * . * TO 'user'@'localhost';

```
mysql> GRANT ALL PRIVILEGES ON * . * TO 'user'@'localhost';
Query OK, 0 rows affected (0,00 sec)
```

Refrescamos los privilegios

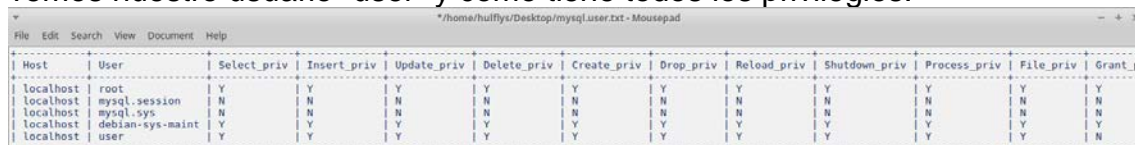
```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,00 sec)
```

Comprobamos que el usuario efectivamente tiene todos los privilegios

SELECT * FROM mysql.user;

```
mysql> SELECT * FROM mysql.user;
```

Lo más cómodo es copiarlo todo en un archivo de texto para poder leerlo, aquí vemos nuestro usuario "user" y como tiene todos los privilegios.



Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv	Grant_priv
localhost	root	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	mysql.session	N	N	N	N	N	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N	N	N	N	N	N
localhost	debian-sys-maint	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	user	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Entramos en mysql con nuestro usuario "user"

mysql -u user -p

```
root@hulfllys-Desktop:/home/hulfllys# mysql -u user -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.22-0ubuntu0.16.04.1 (Ubuntu)
```

Creamos una nueva base de datos "mqtt"

```
# CREATE DATABASE mqtt;
```

```
mysql> CREATE DATABASE mqtt;  
Query OK, 1 row affected (0,00 sec)
```

Cambiamos a la base de datos "mqtt" y le creamos la tabla data, podemos ver como se creo una columna "value" de tipo "VARCHAR"

```
# USE mqtt;
```

```
# CREATE TABLE data (value VARCHAR(800));
```

```
# DESCRIBE data;
```

```
mysql> USE mqtt;  
Database changed  
mysql> CREATE TABLE data (value VARCHAR(800));  
Query OK, 0 rows affected (0,02 sec)
```

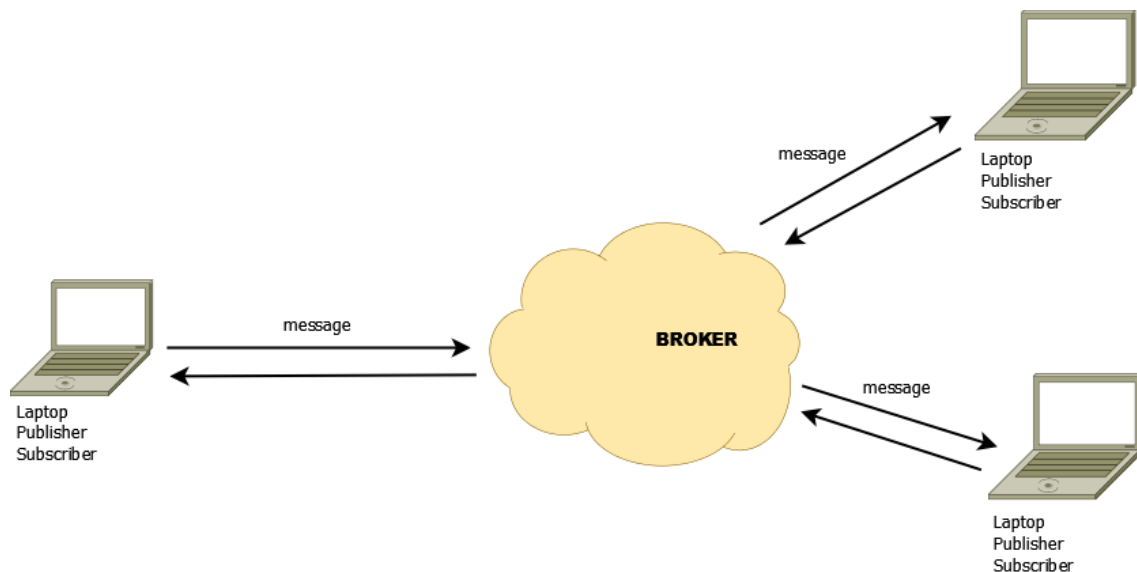
```
mysql> DESCRIBE data;
```

Field	Type	Null	Key	Default	Extra
value	varchar(800)	YES		NULL	

```
1 row in set (0,00 sec)
```

Capítulo 5. Funcionamiento

1. Breve explicación



Como Broker usaremos Eclipse Mosquitto.

Para publicar y subscribirnos a topics utilizaremos 3 máquinas.

Al publicar, usaremos el programa **pub.py**

Publicaremos en:

- ❖ **topic/num** -> Un número generado aleatoriamente.
- ❖ **topic/name** -> Nombre de la máquina que publica.
- ❖ **topic/date** -> Fecha y hora de la máquina que publica.

Al subscribirnos, usaremos 3 programas diferentes:

- ❖ **mysql_sub.py**: Este programa se encargará de guardar los datos recibidos en una base de datos.
- ❖ **mxt_sub.py**: Este programa se encargará de guardar los datos recibidos en un archivo de texto llamado "file.txt"
- ❖ **graf_sub.py**: Este programa se encargará de representar gráficamente los datos recibidos.

2. Demostración

Usamos como subscriber la máquina hulfllys-Desktop y usamos el programa mysql_sub.py

```
# python mysql_sub.py
```

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python mysql_sub.py
Connected with result code 0
```

Publicamos desde nuestras 3 máquinas

```
root@hulfllys-Laptop:/home/hulfllys/Escritorio# python pub.py
```

```
root@hulfllys-VirtualBox:/home/hulfllys/Desktop# python pub.py
```

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python pub.py
```

Vemos como llega toda la info al subscriber

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python mysql_sub.py
Connected with result code 0
Se inserto: 165
Se inserto: hulfllys-Desktop
Se inserto: dom jun 10 16:38:48 CEST 2018
Se inserto: 50
Se inserto: hulfllys-Laptop
Se inserto: Sun 10 Jun 16:41:33 CEST 2018
Se inserto: 195
Se inserto: hulfllys-VirtualBox
Se inserto: Sun 10 Jun 16:44:20 CEST 2018
```

Ahora si vamos a mysql, podremos comprobar como se guarda la información también en nuestra base de datos

```
# select * from data;
```

```
mysql> select * from data;
+-----+
| value |
+-----+
| 63    |
| hulfllys-Desktop |
| dom jun 10 16:18:42 CEST 2018 |
| 165   |
| hulfllys-Desktop |
| dom jun 10 16:38:48 CEST 2018 |
| 50    |
| hulfllys-Laptop  |
| Sun 10 Jun 16:41:33 CEST 2018 |
| 195   |
| hulfllys-VirtualBox |
| Sun 10 Jun 16:44:20 CEST 2018 |
+-----+
12 rows in set (0,00 sec)
```

Usamos de nuevo como subscriber la máquina hulfllys-Desktop y usamos el programa txt_sub.py

```
# python txt_sub.py
```

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python txt_sub.py
Connected with result code 0
```

Volvemos a publicar desde nuestras 3 máquinas

```
root@hulfllys-Laptop:/home/hulfllys/Escritorio# python pub.py
```

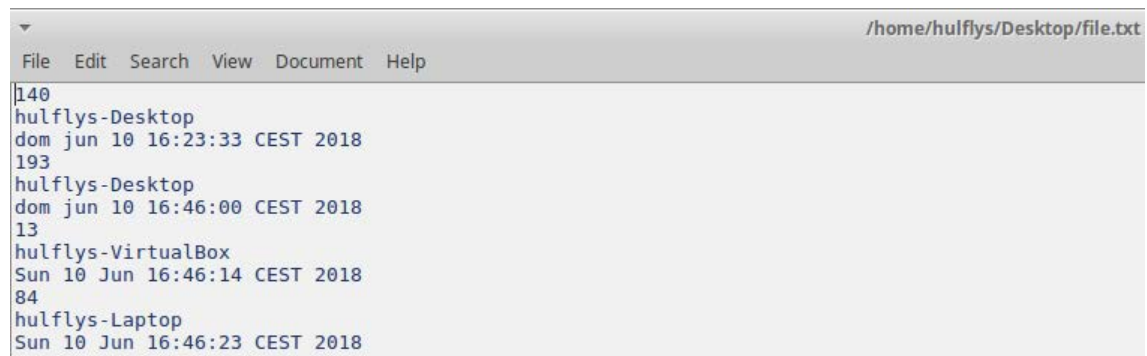
```
root@hulfllys-VirtualBox:/home/hulfllys/Desktop# python pub.py
```

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python pub.py
```

Vemos como llega toda la info al subscriber

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python txt_sub.py
Connected with result code 0
Se inserto: 193
Se inserto: hulfllys-Desktop
Se inserto: dom jun 10 16:46:00 CEST 2018
Se inserto: 13
Se inserto: hulfllys-VirtualBox
Se inserto: Sun 10 Jun 16:46:14 CEST 2018
Se inserto: 84
Se inserto: hulfllys-Laptop
Se inserto: Sun 10 Jun 16:46:23 CEST 2018
```

Ahora si vamos a nuestro archivo file.txt, podremos comprobar cómo se guarda la información en él



```
140
hulfllys-Desktop
dom jun 10 16:23:33 CEST 2018
193
hulfllys-Desktop
dom jun 10 16:46:00 CEST 2018
13
hulfllys-VirtualBox
Sun 10 Jun 16:46:14 CEST 2018
84
hulfllys-Laptop
Sun 10 Jun 16:46:23 CEST 2018
```

Por último usamos como subscriber la máquina hulfllys-Desktop con el programa graf_sub.py

```
# python graf_sub.py
```

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python graf_sub.py
Connected with result code 0
```

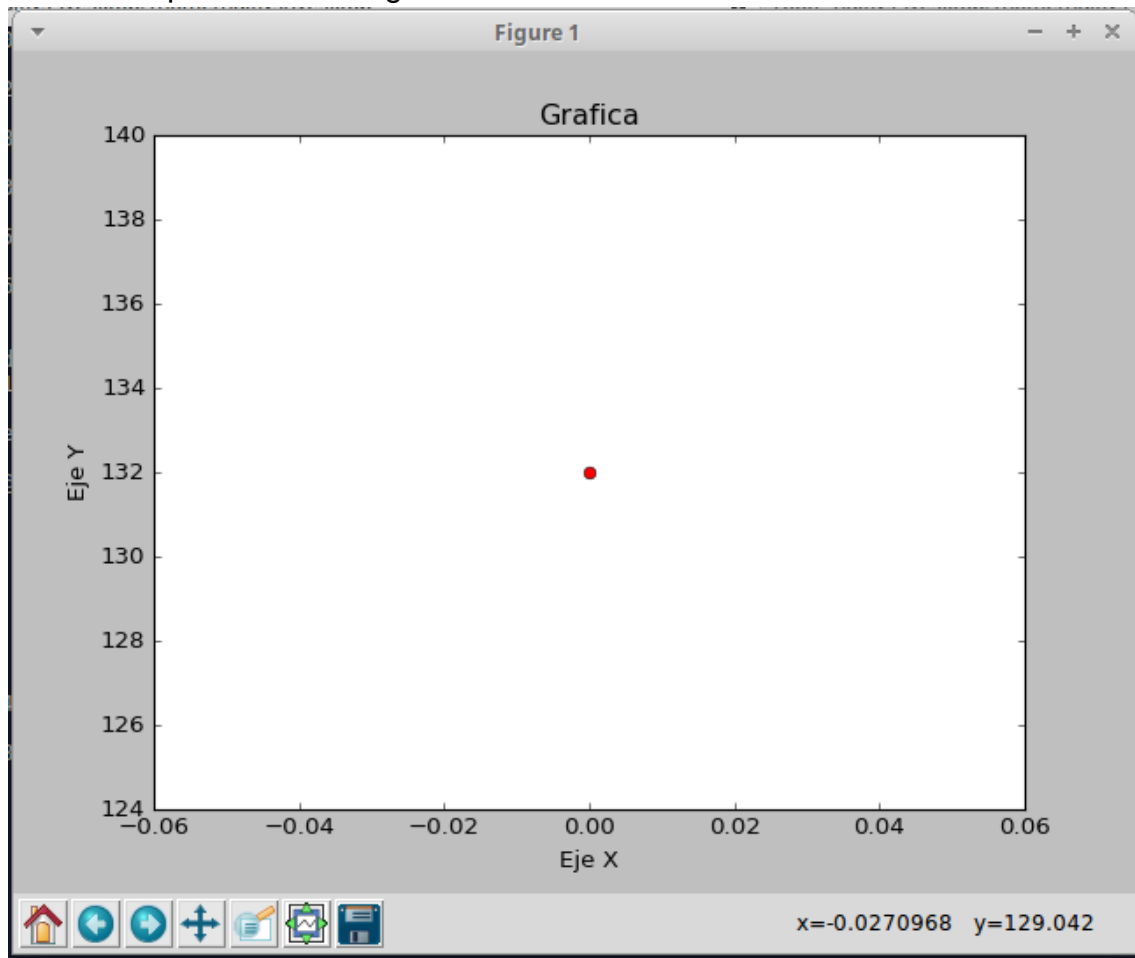
Publicamos únicamente con hulfllys-Desktop

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python pub.py
```

Vemos el valor que captó el subscriber

```
root@hulfllys-Desktop:/home/hulfllys/Desktop# python graf_sub.py
Connected with result code 0
Se inserto: 132
```

El valor se pública en una gráfica



Capítulo 6. Programación

1. Publisher

Programa sub.py

```
#!/usr/bin/env python3
#Indicamos que queremos que se ejecute con python3

import paho.mqtt.client as mqtt
#Importamos el módulo paho.mqtt.client y lo nombramos como mqtt
import random
#Importamos el módulo random
import commands
#Importamos el módulo commands

client = mqtt.Client()
client.connect("85.137.205.6",1883,60)
#Conexión con el broker
client.publish("topic/num", random.randint(0, 200));
#Guardamos en topic/num el valor de la derecha
client.publish("topic/name", commands.getoutput('hostname'));
#Guardamos en topic/name el valor de la derecha
client.publish("topic/date", commands.getoutput('date'));
#Guardamos en topic/date el valor de la derecha
client.disconnect();
#Desconectamos el cliente
```


2. Subscribers

Programa txt_sub.py

```
#!/usr/bin/env python3
#Indicamos que queremos que se ejecute con python3

import paho.mqtt.client as mqtt
#Importamos el módulo paho.mqtt.client y lo nombramos como mqtt

def on_connect(client, userdata, flags, rc):
    #Definimos la conexión

    print("Connected with result code "+str(rc))
    #Mostramos que conecto correctamente
    client.subscribe("topic/num")
    #Indicamos que el cliente se subscribe a "topic/num"

    client.subscribe("topic/name")
    #Indicamos que el cliente se subscribe a "topic/name"
    client.subscribe("topic/date")
    #Indicamos que el cliente se subscribe a "topic/date"

def on_message(client, userdata, msg):
    #Definimos el mensaje

    values = msg.payload.decode()
    #Guardamos el mensaje en la variable values

    file = open("file.txt", "a")
    #Abrimos un archivo llamado "file.txt", con "a" no se sobrescribe
    file.write(" "+values+"\n")
    #Escribimos en el archivo el valor del mensaje
    file.close()
    #Cerramos el archivo de texto

    print("Se inserto: "+values)
    #Indicamos por pantalla el mensaje que llegó

client = mqtt.Client()
client.connect("85.137.205.6",1883,60)
#Conexión con el broker

client.on_connect = on_connect
client.on_message = on_message

client.loop_forever()
```

Programa mysql_sub.py

```
#!/usr/bin/env python3
#Indicamos que queremos que se ejecute con python3

import paho.mqtt.client as mqtt
#Importamos el módulo paho.mqtt.client y lo nombramos como mqtt
import MySQLdb
#Importamos el módulo MySQLdb

def on_connect(client, userdata, flags, rc):
#Definimos la conexión

    print("Connected with result code "+str(rc))
    #Mostramos que conecto correctamente

    client.subscribe("topic/num")
    #Indicamos que el cliente se subscribe a "topic/num"
    client.subscribe("topic/name")
    #Indicamos que el cliente se subscribe a "topic/num"
    client.subscribe("topic/date")
    #Indicamos que el cliente se subscribe a "topic/num"

def on_message(client, userdata, msg):
#Definimos el mensaje

    values = msg.payload.decode()
    #Guardamos el mensaje en la variable values
    print("Se inserto: "+values)
    #Indicamos por pantalla el mensaje que llegó
    db=MySQLdb.connect(host='localhost',user='user',passwd='2asirtriana',db='mqtt')
    #Definimos la conexión con la base de datos y la guardamos en db
    cursor = db.cursor()
    #Definimos un cursor y lo guardamos en cursor
    query = "INSERT INTO data VALUES ('"+values+"');"
    #Realizamos una consulta y la guardamos en query
    cursor.execute(query)
    #Ejecutamos el cursor
    db.commit()
    #Confirmamos
    cursor.close()
    #Cerramos el cursor

client = mqtt.Client()
client.connect("85.137.205.6",1883,60)
#Conexión con el broker
```

```
client.on_connect = on_connect  
client.on_message = on_message  
  
client.loop_forever()
```

Programa graf_sub.py

```
#!/usr/bin/env python3
#Indicamos que queremos que se ejecute con python3

import paho.mqtt.client as mqtt
#Importamos el módulo paho.mqtt.client y lo nombramos como mqtt
import matplotlib.pyplot as plt
#Importamos el módulo matplotlib.pyplot y lo nombramos como plt
from pylab import *
#Lo importamos todo de pylab

def on_connect(client, userdata, flags, rc):
    #Definimos la conexión

    print("Connected with result code "+str(rc))
    #Mostramos que conecto correctamente
    client.subscribe("topic/num")
    #Indicamos que el cliente se subscribe a "topic/num"

def on_message(client, userdata, msg):
    #Definimos el mensaje

    values = msg.payload.decode()
    #Guardamos el mensaje en la variable values
    print("Se inserto: "+values)
    #Indicamos por pantalla el mensaje que llegó
    title('Grafica')
    #Titulo de la gráfica
    xlabel('Eje X')
    #Titulo del eje X
    ylabel('Eje Y')
    #Título del eje Y
    plt.plot(values, 'ro')
    #Representa el valor del mensaje con un punto rojo en la gráfica
    print(plt.show())
    #muestra la gráfica

client = mqtt.Client()
client.connect("85.137.205.6",1883,60)
#Conexión con el bróker

client.on_connect = on_connect
client.on_message = on_message

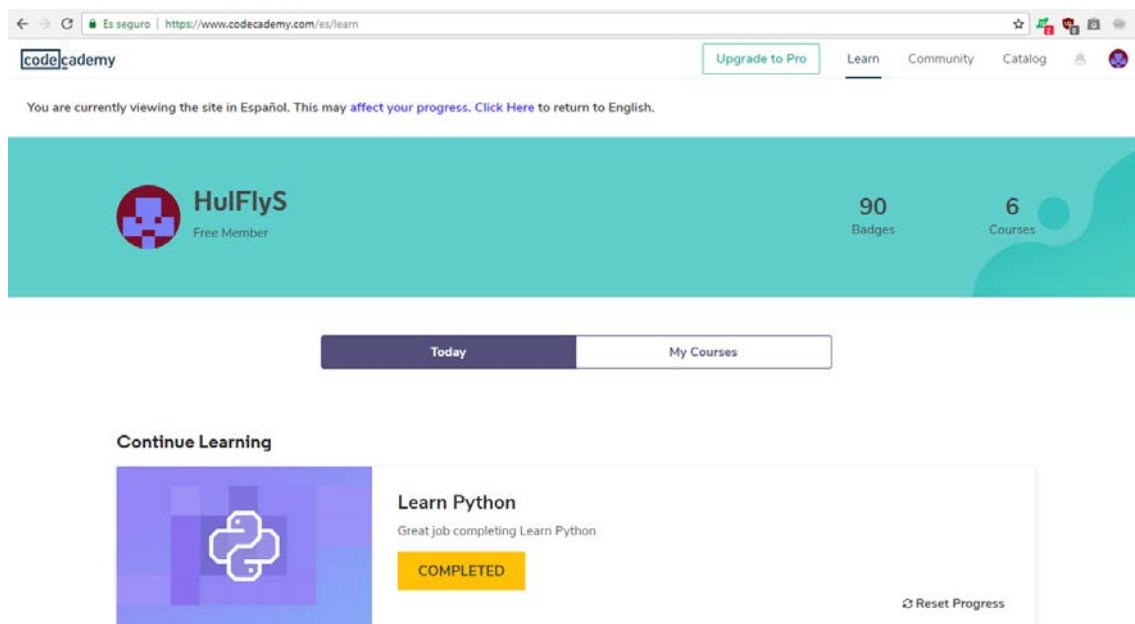
client.loop_forever()
```

Capítulo 7. Autoformación

Para poder llevar a cabo el proyecto he tenido que realizar un curso de Python en <https://www.codecademy.com/es/>, porque, este lenguaje no lo hemos utilizado en los dos años de ciclo formativo y toda la programación del proyecto se lleva a cabo con el mismo.

El curso es largo, por lo que me tomó unas 12 horas realizarlo entero.

Aquí adjunto una foto del curso de Python terminado al 100%



Capítulo 8. Posibles ampliaciones y/o mejoras

Respecto a lo que tenemos hecho podrían aplicarse las siguientes ampliaciones y/o mejoras:

- ❖ Utilizar como Publisher uno o dos Andruinos (ROS) como se planificó al principio del proyecto y utilizar las máquinas únicamente como subscribers a esos publishers.
- ❖ Mejorar el programa que representa gráficamente, ya que es muy interesante el módulo matplotlib que nos permite usar Python como si fuera Matlab y tiene múltiples opciones de representación de datos.
- ❖ Crear una interfaz Web sencilla para publicar datos. Hacer una lectura de la tabla data de la base de datos y ver los valores que contiene, lo mismo se podría hacer con el archivo file.txt y por último en esta interfaz web, realizar también la representación gráfica de los datos.
- ❖ Intentar utilizar como Broker y Cliente Eclipse Kapua en vez de Eclipse Mosquitto y Eclipse Paho, ya que se trata de un nuevo programa que contiene una interfaz web, pero debido a la poca documentación que existe ahora mismo en internet, decidí utilizar el Broker Eclipse Mosquitto y el Cliente Eclipse Paho, de los cuales hay muchísima documentación en internet y por lo tanto era más factible su uso.

Capítulo 9. Reflexión y conclusiones

Tras la realización del proyecto final sobre el IoT, en concreto, el protocolo MQTT, y toda la información recopilada en internet, a causa de esto, me doy cuenta de que actualmente el IoT está muy presente en nuestra vida cotidiana. Aunque como he dicho, actualmente se está usando MQTT bastante para IoT, me llamó muchísimo la atención, que este protocolo lo desarrolló IBM en los años 90 y que hasta ahora, excepto en ciertos trabajos específicos, apenas se usaba. Esto denota una evolución de la tecnología que ha llevado hasta nuestros hogares y quehaceres cotidianos tecnologías que antes tenían fines únicamente científicos o industriales.

Con este proyecto he aprendido muchos conceptos nuevos, ya que muchas de las cosas que he hecho en él, por no decir la mayoría, no las he estudiado a lo largo de los 2 años de ciclo que hemos tenido.

Por esto, adquiriré nuevos conceptos, sé lo que es el protocolo MQTT y como funciona, también sé lo que es un Broker (cómo Eclipse Mosquitto o Eclipse Kapua) y he aprendido a utilizarlo, se lo que es un topic (tema), lo que hacen el Publisher y el Subscriber, etc.

Además, gracias a este proyecto, he “aprendido” a programar en Python, porque todos los programas los tuve que desarrollar en este lenguaje, al principio fue un problema, ya que como era un lenguaje que desconocía, tuve que dedicarme a hacer cursos y familiarizarme un poco con el, pero una vez pasado esto, pude utilizarlo para desarrollar las aplicaciones sin problemas.

Ahora finaliza este ciclo, o mejor dicho, esta etapa y comienza otra nueva, tras 2 años en los que he aprendido bastante, incluso me sorprende, todo lo que he aprendido realizando este proyecto, que la verdad, me costó mucho trabajo al principio, ya que tuve que modificar varias cosas a lo largo del proyecto a raíz de la información que fui recopilando por Internet, pero creo que esto es algo común en cualquier tipo de proyecto que se realice, uno se va encontrando problemas y va buscando soluciones.

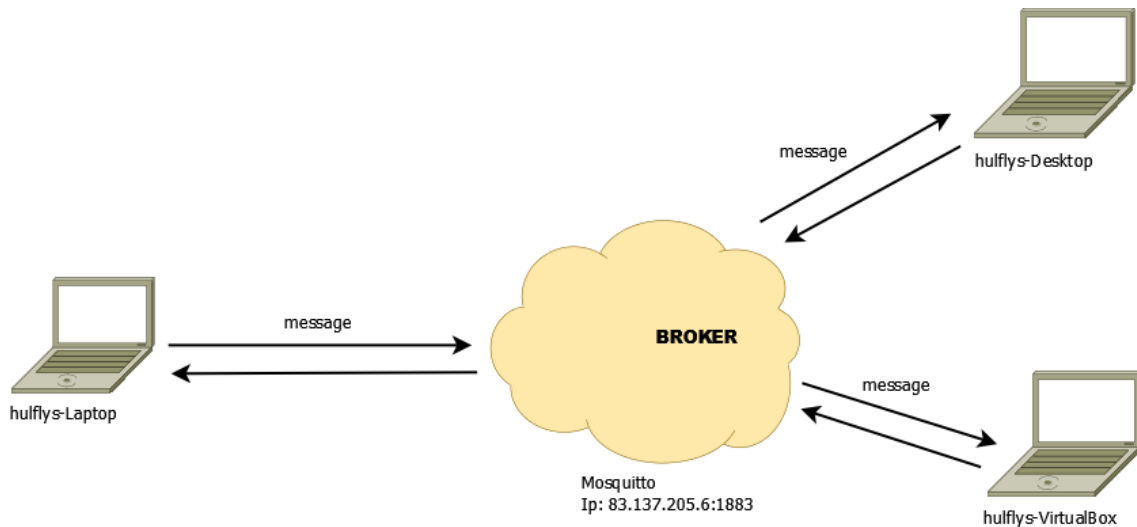
Finalmente creo que estoy satisfecho con el trabajo realizado y con todo lo aprendido a lo largo de estos casi 3 meses.

Capítulo 10. Bibliografía

<https://mosquitto.org/>
<https://test.mosquitto.org/>
<https://www.eclipse.org/paho/>
<http://mqtt.org/>
<https://www.python.org/>
<https://docs.python.org/3/>
<http://mysql-python.sourceforge.net>
<https://dev.mysql.com/>
<https://matplotlib.org/>
<https://stackoverflow.com/>
<https://www.w3schools.com/>
<https://www.digitalocean.com/>
<http://www.ev3dev.org>
<https://www.codecademy.com/es>

Manual de administración

Esquema de red



Servidor (Broker)

Eclipse Mosquitto en la nube (OpenNebula)

IP: 83.137.205.6:1883

Usuario: root

Clave: Jge0518jge

Máquinas

hulflys-Laptop (Publisher/Subscriber)

Ubuntu 16.04 LTS

Usuario: HulFlyS

Usuario admin: root

Clave: 2asirtriana

hulflys-Desktop (Publisher/Subscriber)

Máquina virtual (VirtualBox) Ubuntu 16.04 LTS

Usuario: HulFlyS

Usuario admin: root

Clave: 2asirtriana

hulflys-VirtualBox (Publisher/Subscriber)

Máquina virtual (VirtualBox) Ubuntu 16.04 LTS

Usuario: HulFlyS

Usuario admin: root

Clave: 2asirtriana

Base de datos

Mysql

Usuario: user

Clave: 2asirtriana

Base de datos: mqtt

Tabla: data (Consta de una columna llamada value)

Procedimientos

Todos los programas se encuentran en el escritorio de las máquinas, se ejecutan desde el terminal con python (ej: # python pub.py)

La base de datos se ejecutará desde el terminal y cualquier consulta que queramos hacer en ella se hará desde ahí.

Para acceder al bróker, deberemos acceder a OpenNebula <http://85.137.205.6:9869/>

Usuario: juangil

Clave: giljuan

Una vez dentro, ejecutar la única máquina que hay creada (Ubuntu).

En https://github.com/HuIFlyS/PROYECTO_GIL_ESTRADA_JUAN está toda la información del proyecto, código, memoria, manuales, etc.