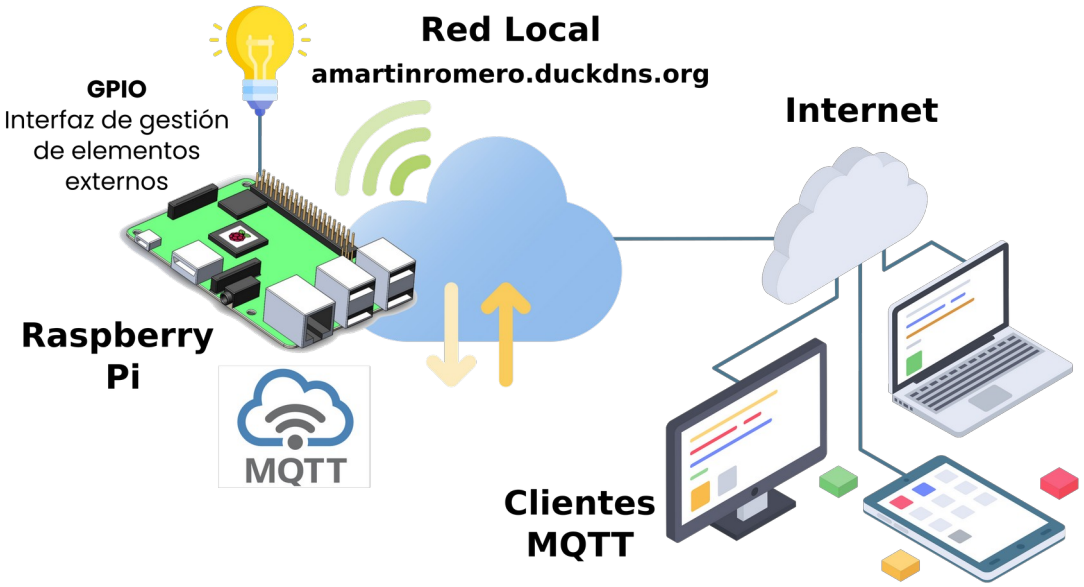
Tarea Curso Aularagón

RASPBERRY PI

Docker + MQTT + GPIO

VERSIÓN 24.01.18



CATEDU
Zaragoza – ESPAÑA
Arturo Martín Romero
amartinromero@gmail.com
18 enero 2024

Índice de Prácticas

Práctica nº1.- Servidor y cliente mosquitto

Implementación del servidor Mosquitto

Más info: Inhttps://github.com/sukesh-ak/setup-mosquitto-with-docker

Para configurar el servidor Mosquitto en la raspberry pi haremos uso de docker. Los pasos a seguir serán los siguientes:

0) Instalación de docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker ${USER}
sudo su - ${USER}
docker version
```

1) Creamos la estructura de directorios y fichero de configuración:

```
mkdir -p mqtt5/config # for storing mosquitto.conf and pwfile (for password)
touch mqtt5/config/pwfile mqtt5/config/mosquitto.conf
cd mqtt5

nano config/mosquitto.conf
allow_anonymous false
listener 1883
listener 9001
protocol websockets
persistence true
password_file /mosquitto/config/pwfile
persistence_file mosquitto.db
persistence_location /mosquitto/data/
```

2) Creamos el fichero de configuración de docker-compose:

```
nano docker-compose.yml
version: "3.7"
services:
  # mqtt5 eclipse-mosquitto
  mqtt5:
    image: eclipse-mosquitto
    container_name: mqtt5
    ports:
      - "1883:1883" #default mqtt port
      - "9001:9001" #default mqtt port for websockets
    volumes:
```

```
      - ./config:/mosquitto/config:rw
      - ./data:/mosquitto/data:rw
      - ./log:/mosquitto/log:rw

# volumes for mapping data,config and log
volumes:
  config:
  data:
  log:

networks:
  default:
    name: mqtt5-network
```

3) Ponemos en marcha el contenedor y obtenemos su id de contenedor:

```
docker-compose -p mqtt5 up -d
docker ps --all
```

4) Entramos al contenedor para personalizar la autenticación al servicio:

```
# login interactively into the mqtt container
sudo docker exec -it <container-id> sh

# Cambiamos permisos y propietario del fichero de passwords:
chmod 700 /mosquitto/config/pwfile
chown root:root /mosquitto/config/pwfile

# add user and it will prompt for password
mosquitto_passwd -c /mosquitto/config/pwfile user1

# delete user command format
mosquitto_passwd -D /mosquitto/config/pwfile <user-name-to-delete>

# type 'exit' to exit out of docker container prompt
```

5) Reiniciamos el contenedor:

```
docker restart <container-id>
```

Implementación del cliente de Mosquitto

En primer lugar será necesario instalar el cliente MQTT Mosquitto en todos aquellos dispositivos desde los cuales queramos interactuar con el servidor MQTT implementado, mediante la suscripción a topics y envío de mensajes. En concreto, en la raspberry instalaremos el cliente MQTT para poder estar pendientes a los mensajes que se envíen desde Internet y actuar ante estos sobre los elementos externos (actuadores) mediante su GPIO.

```
sudo apt install mosquitto-clients # Instalación del cliente Mosquitto en la Raspberry y PC
```

Para comprobar su funcionamiento, primero nos suscribiremos a un canal/topic (p.e. test/topic1) y luego publicaremos mensajes (les llegará un mensaje a todos lo que se hayan suscrito al topic al que se envía):

mosquitto_sub -v -L mqtt://arturo:aularagon@amartinromero.duckdns.org/test/topic1
mosquitto_pub -L mqtt://arturo:aularagon@amartinromero.duckdns.org/test/topic1 -m 'hola MQTT'

Recibiendo por la terminal de los que se han suscrito:

test/topic1 hola MQTT

En la rasperry, una vez suscritos, haremos un bucle y estar atentos de lo que vamos recibiendo, haciendo un estudio de casos para actuar ante el mensaje recibido. Si hacemos el script del bucle en bash podría ser algo así (en el siguiente apartado detallo un poco más el script):

```
#!/bin/bash
URL="mqtt://arturo:aularagon@amartinromero.duckdns.org"
TOPIC="/test/topic1"
while true; do
  echo "#> Nos suscribimos al canal ${TOPIC}"
  MENSAJE="$(mosquitto_sub -C 1 -v -L "${URL}${TOPIC}" | tee -a temp.txt)"
  echo "#> Mensaje recibido: \"${MENSAJE}\""
  case "${MENSAJE}" in
          *expresion1*)
                  comandos_ejecutar
                  ;;
          *expresion2*)
                  comandos_ejecutar
                  ;;
          *)
                  echo "#> El mensaje no concuerda con ninguna acción programada. Esperamos otra
orden."
                  ;;
  esac
done
```

En Android también existen aplicaciones cliente MQTT para poder enviar mensajes al canal/topic a través de Internet y que le lleguen al broker MQTT implementado en la raspberry, y así actuar esta en consecuencia sobre los elementos externos a través de su GPIO:

* Instalar un “MQTT Client”, por ejemplo, **MyMQTT** para enviar mensajes al broker mosquitto instalado en la raspberry pi y de esta forma gestionar el comportamiento del servicio que esta suscrito a escucha de los mensajes enviados.
* Instalar RaspController para tener información desde el móvil del estado de los puertos de salida del GPIO.

Ejemplo de escucha y actuación sobre los puertos del GPIO de la raspberry

Para el siguiente ejemplo, tenemos un led (o el dispositivo sobre el que queremos actuar) conectado al puerto 40 del GPIO, o también conocido como GPIO21 (Raspberry Pi 3), el cual se accionará (apagar/encender) en función del mensaje recibido vía MQTT. En concreto:

1) En la raspberry esta corriendo un servidor Mosquitto accesible desde Internet (DNS público): amartinromero.duckdns.org

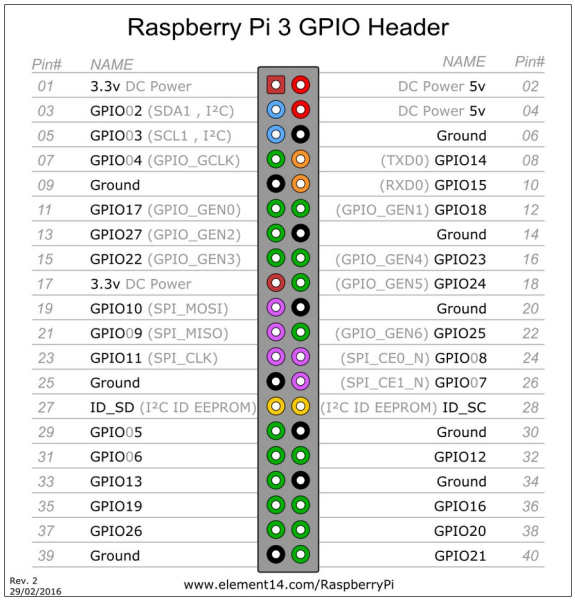
2) En la raspberry hay un cliente MQTT Mosquitto suscrito a un canal o topic denominado test/topic1 a la espera de mensajes de encendido o apagado. Para ello se hace uso de los siguientes scripts, uno para la suscripción y escucha de mensajes, y otro para actuar sobre el puerto del GPIO:

```
#!/bin/bash
URL="mqtt://arturo:aularagon@amartinromero.duckdns.org"
TOPIC="/test/topic1"
while true; do
  echo "#> Nos suscribimos al canal ${TOPIC}"
  MENSAJE="$(mosquitto_sub -C 1 -v -L "${URL}${TOPIC}" | tee -a temp.txt)"
  echo "#> Mensaje recibido: \"${MENSAJE}\""
  case "${MENSAJE}" in
        *encender_salon*)
                echo "#> Vamos a ENCENDER la luz del salón"
                sudo ./gpio-pinout.sh "21" "out" "1"
                ;;
        *apagar_salon*)
                echo "#> Vamos a APAGAR la luz del salón"
           sudo ./gpio-pinout.sh "21" "out" "0"
           ;;
        *)
                echo "#> El mensaje no concuerda con ninguna acción programada. Esperamos otra
orden."
                ;;
  esac
done
```

Donde el script gpio-pinout.sh ubicado en el mismo directorio que el anterior tiene el siguiente aspecto:

```
#!/bin/bash
NUM="${1:-"21"}"
MODO="${2:-"out"}"
ESTADO="${3:-"1"}"
DIRGPIO="/sys/class/gpio/gpio${NUM}"
[[ ! -d "${DIRGPIO}" ]] && \
echo "#> Activamos el PIN del GPIO: ${NUM}" && \
echo "${NUM}" > /sys/class/gpio/export
echo "${MODO}" > "${DIRGPIO}/direction"
```

echo "${ESTADO}" > "${DIRGPIO}/value"



3) Desde un cliente MQTT externo (PC, móvil, etc.) enviando los mensajes “encender_salon” o “apagar_salon” provocarán el encendido o apagado del led (o el dispositivo que hubiera) en el GPIO 21.

Es importante configurar los privilegios de sudo para poder ejecutar el script de gestión del GPIO como permisos del root. Lo más rápido y sencillo sería configurar el sudo de la siguiente forma (el usuario arturo pertenece al grupo docker y al grupo sudo):

id arturo
uid=1001(arturo) gid=1001(arturo) grupos=1001(arturo),27(sudo),118(lpadmin),995(docker)

sudo visudo
%sudo   ALL=(ALL:ALL) NOPASSWD: ALL

Erratas y críticas constructivas del curso

0) Echo en falta unos objetivos del curso, que se pretende conseguir y encaminarlo a ello. El curso es un conjunto de “islas” de cosas que se pueden hacer pero sin definirse, y aclararse en muchas de ellas el como hacerlo. Es decir, el curso parece más bien un glosario de cosas que se pueden hacer con una raspberry, que un curso con un hilo conductor que te conduce a realizar algo.

1) https://libros.catedu.es/books/raspberry-pi/page/31-portainer-gestion-de-contenedores

El contenido del fichero docker-compose.yml, los ports deben ir entre comillas dobles:

- 9000:9000 => - “9000:9000”

2) En el contenedor asociado al servicio DIUN, al lanzar el servicio crea un directorio llamado diun.yml, igual que el fichero de configuración que hay que crear. Hay que eliminar dicho directorio vacío para crear el fichero de configuración que se sugiere en la práctica.

3) En el filebrowser no se hace bien la redirección de puertos, no funciona correctamente. No es suficiente con poner comillas dobles. Yo he tenido que hacer uso de otro contenedor.

4) En la práctica de “duckdns” sería necesario aclarar que es necesario configurar el redireccionamiento de puertos en el router para que funcione todo correctamente. En el caso del servicio “duckdns” al ser mi raspberry la versión 3 B+, de arquitectura ARM32v7l, he tenido que recurrir a otro contenedor que admitiera dicha arquitectura:

```
version: "2.1"
services:
  duckdns:
    image: maksimstojkovic/duckdns
    container_name: duckdns
    environment:
      - DUCKDNS_DOMAIN="amartinromero.duckdns.org"
      - DUCKDNS_TOKEN="eb786147-c536-4504-8b95-22b5c5489234"
      - DUCKDNS_DELAY=60
    restart: unless-stopped
```

5) El contenedor para hacer de servidor de fotos requiere de 4GB de SWAP, no de RAM, ya que se puede implementar en una Raspberry Pi

6) El uso de ZigBee no queda claro, ni como usarlo, ni para que