UNIVERSIDAD POLITÉCNICA DE MADRID

Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid

SUSCRIPTOR MQTT

Introducción

En esta práctica[1][2] vamos a programar un suscriptor MQTT `subs.py` que recolecte las métricas publicadas por el cliente `client.py`. Para ello abriremos una suscripción con el broker y nos suscribiremos a los topics de reporte de métricas. Finalmente, haremos un detector de anomalías en el pulso para publicar alertas desde el `subs.py`. La figura mostrada a continuación resalta con fondo oscuro las componentes que modificaremos:

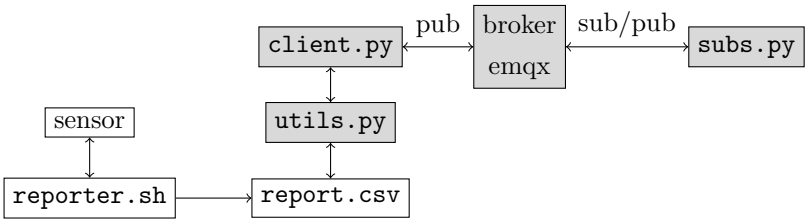

Figura 1: escenario de la práctica.

Arrancar broker EMQX

Antes de comenzar a programar el `subscriber.py` tenemos que arrancar el broker EMQX. Para ello siga los pasos de la práctica anterior. En caso de que el contenedor del broker EMQX ya esté disponible, ejecute el siguiente comando para arrancarlo de nuevo:

```
$ docker start ‘docker ps | grep "emqx:5.0.2"\
  | cut -d’ ’ -f1‘
```

[1] Todas las preguntas tienen el mismo valor/puntuación.

Puede comprobar que el broker se está ejecutando correctamente accediendo al panel de EMQX desde el navegador:

`localhost:18083`

usando el como nombre de usuario `admin` y como contraseña `public`.

Publicar métricas en topics

Tras arrancar el broker EMQX vamos a publicar cada métrica en un topic distinto. En concreto, publicaremos cada columna del `report.csv` en el topic `vitals/metrica`. Por ejemplo, el ritmo cardíaco `hr` se publica en el topic `vitals/hr`.

Crear diccionario de métricas

Para ello debe modificar crear una función en `utils.py` que reciba una línea de fichero y devuelva un diccionario python. Llame a dicha función `line_to_dict(fpath,line)` y prográmela para que devuelva el siguiente diccionario al recibir la primera línea del CSV de las constantes vitales:

```
{
  "idx": 0,
  "time": 0,
  "hr": 94,
  "resp": 21,
  "SpO2": 97,
  "temp": 36.2,
  "output": "Normal"
}
```

Para probar que la función está bien programada ejecútela en el entorno python3:

```
$ python3
>>> import utils
>>> utils.line_to_dict('report.csv',X+2)
```

donde X es el número del grupo al que pertenece.

Problema 1. Guarde el resultado en el campo `lectura` del JSON de respuestas.

Publicar en distintos topics

A continuación vamos a publicar cada métrica de las constantes vitales en un topic diferente. En concreto, vamos a publicar las constantes vitales `hr`, `resp`, `SpO2` y

`temp`. Para ello va a modificar el código de su `client.py` para que haga lo siguiente cada vez que lee una línea de constantes vitales:

1. obtener el diccionario de la línea usando `line_to_dict(fpath,line)`; y
2. publicar cada constante vital el un topic usando QoS 1.

Una vez haya modificado en `client.py`, ejecútelo para que reporte las constantes al broker EMQX[3], inicie una captura en Wireshark en la interfaz `lo`. Deje que se publiquen varias constantes vitales y guarde[4] en `publishes-grupoX.pcapng` la captura.

Problema 2. ¿Cuál es el máximo valor de `temp` reportado en la captura? Responda en el campo `maxtemp` del JSON de respuestas.

Suscripción a topic de temperatura

A continuación va a suscribirse al topic `vitals/temp` usando el script `subs.py` que se proporciona en la práctica. Este script lo usaremos para suscribirnos al topic de temperatura a través del broker EMQX.

Una vez haya modificado el script `subs.py`, inicie una captura wireshark. Después ejecute el `client.py` y `subs.py` para comprobar que se reciben las métricas de temperatura. Detenga la captura cuando se hayan recibido un par de métricas de temperatura y guarde la captura en el fichero `subs-temp-grupoX.pcapng`.

Fíjese en la captura de wireshark y responda a las siguientes preguntas:

Problema 3. ¿Cuál es el tipo de mensaje del `Subscribe request`?

Problema 4. ¿Con qué QoS se suscribe al topic `vitals/temp`?

Problema 5. ¿Cuál es el puerto utilizado por el cliente para publicar mensajes?

Problema 6. ¿Cuál es el puerto utilizado por el suscriptor para recibir mensajes?

Responda a las preguntas en los siguientes campos del JSON de respuestas: `reqtype`, `subsqos`, `pubport` y `subport`; respetivamente

Suscripción a varios topics

Ahora vamos a suscribirnos a todos los topics de constantes vitales que reporte nuestro `client.py`. Para ello, use un wildcard en el topic al que se suscribe el

[3]Recuerde identificar el puerto y dirección usados por EMQX para las comunicaciones MQTT.
[4]Recuerde que todas las capturas de la práctica deben contener *solo* los paquetes MQTT.

`subs.py` y compruebe que el suscriptor recibe todas las métricas reportadas por el `client.py`.

Inicie una captura wireshark y ejecute el `client.py` y `subs.py`. Deje que se publiquen unos veinte `PUBLISH` y guarde en `subs-wildcard-grupoX.pcapng` la captura de wireshark.

Problema 7. ¿A cuántas métricas se suscribe el `subs.py`?

Problema 8. ¿Cuál es el topic del `PUBACK` con `Message identifier` $10 + X$?

Responda a las preguntas en los siguientes campos del JSON de respuestas `nummetricas` y `pubacktopic`.

Alerta por anomalías

Para terminar vamos a generar alertas cuando la métrica `hr` exceda en un $\frac{X}{100}$ por ciento el valor medio de 89 BPM. Para ello debe modificar el script `subs.py` para que:

1. compruebe que recibe un mensaje del topic `vitals/hr`; y
2. envíe un `PUBLISH` en caso de que se cumpla la condición para generar la alerta.

Cuando publique la alerta, hágalo en el topic `alerts/hr` y envíe como mensaje cuáles son los BPM.

Una vez tenga funcionando el script `subs.py` debe iniciar una captura wireshark en la interfaz `lo`. Después inicie el `client.py` y el `subs.py`. Espere a que se publiquen un par de alertas antes de parar la captura, guárdela en el archivo `alertas-grupoX.pcapng`, y responda a las siguientes preguntas:

Problema 9. ¿Cuántas alertas se envían?

Problema 10. ¿Cuál es la QoS con la que envía las alertas?

Responda a las preguntas en los siguientes campos del JSON de respuestas: `numalerts` y `qosalerts`.

Entrega

Se subirá a moodle un archivo `subscriberX.zip` (con `X` el número de grupo) que contenga:

1. el cliente `client.py`;

2. el cliente `subs.py`;
3. el archivo `utils.py`;
4. el JSON de respuestas `respuestas-X.json`; y
5. las trazas de Wireskark `publishes-grupoX.pcapng`, `subs-temp-grupoX.pcapng`, `subs-wildcard-grupoX.pcapng`, `alertas-grupoX.pcapng`

Atención I: las capturas deben contener *solamente* tráfico MQTT.
Atención II: una entrega sin los archivos especificados, o con archivos sin formato especificado tendrá un 0 en los **Problemas** correspondientes.