Grupo 5

Práctica

11769 – Conectivitat i Integració de Sistemes a IoT

**Carlos Pérez Cano**

**Miquel Àngel Román Colom**

Contenido

[Preparación del entorno 2](#_Toc57928178)

[Ejercicio 1 4](#_Toc57928179)

[Ejercicio 2 8](#_Toc57928180)

[Ejercicio 3 11](#_Toc57928181)

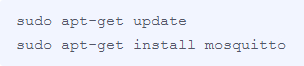
[Ejercicio 4 15](#_Toc57928182)

# Preparación del entorno

Todo el material generado y necesario utilizado en este documento se puede encontrar [aquí](https://github.com/NanoSpicer/MQTT-Over-TLS).

Para poder reproducir completamente este ejercicio necesitaremos instalar los paquetes siguientes de Ubuntu:

* **Mosquitto**: Un servidor que implementa el protocolo MQTT. Además, viene con un conjunto de herramientas útiles para su gestión.



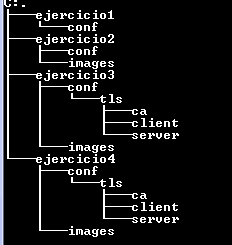
* **Mosquitto clients**: Conjunto de clientes para poder hacer operaciones como publicar y suscribirse en tópicos.



* **Tcpflow**: Una utilidad de la línea de comandos que nos permite leer el tráfico entrante y saliente de nuestra máquina. Útil para verificar si mosquitto y los clientes cifran los datos antes de enviarlos.



En el enlace proporcionado al principio de este apartado se puede observar una estructura de directorios del siguiente tipo:



En cada uno de los directorios existe un script **mosquitto.sh** cuya funcionalidad es simplificar el acceso a los ficheros locales de cada subdirectorio de configuración. Es decir, en el directorio ejercicio1 contiene su propio fichero **mosquitto.conf** dentro del directorio **conf** con los cambios necesarios para el ejercicio1 y el script **mosquitto.sh** se encarga de lanzar el servidor de mosquitto usando la configuración correspondiente al ejercicio1 y análogamente para el resto de ejercicios.

A parte de estos “wrappers” se pueden encontrar otros scripts que simplifican la simulación de las pruebas para verificar que solucionamos los ejercicios de manera satisfactoria.

# Ejercicio 1

**Ejercicio (1 punto)**: Configurar el broker para que pueda controlar a los usuarios no anónimos Crear varios usuarios con su contraseña y realizar diferentes pruebas verificando que el broker permite o restringe su acceso.

Así sin más, y con un fichero de configuración de mosquitto vacío podemos ejecutar el servidor como ilustran las Figuras 1, 2 y 3 para enviar y recibir mensajes sin ningún tipo de restricción.

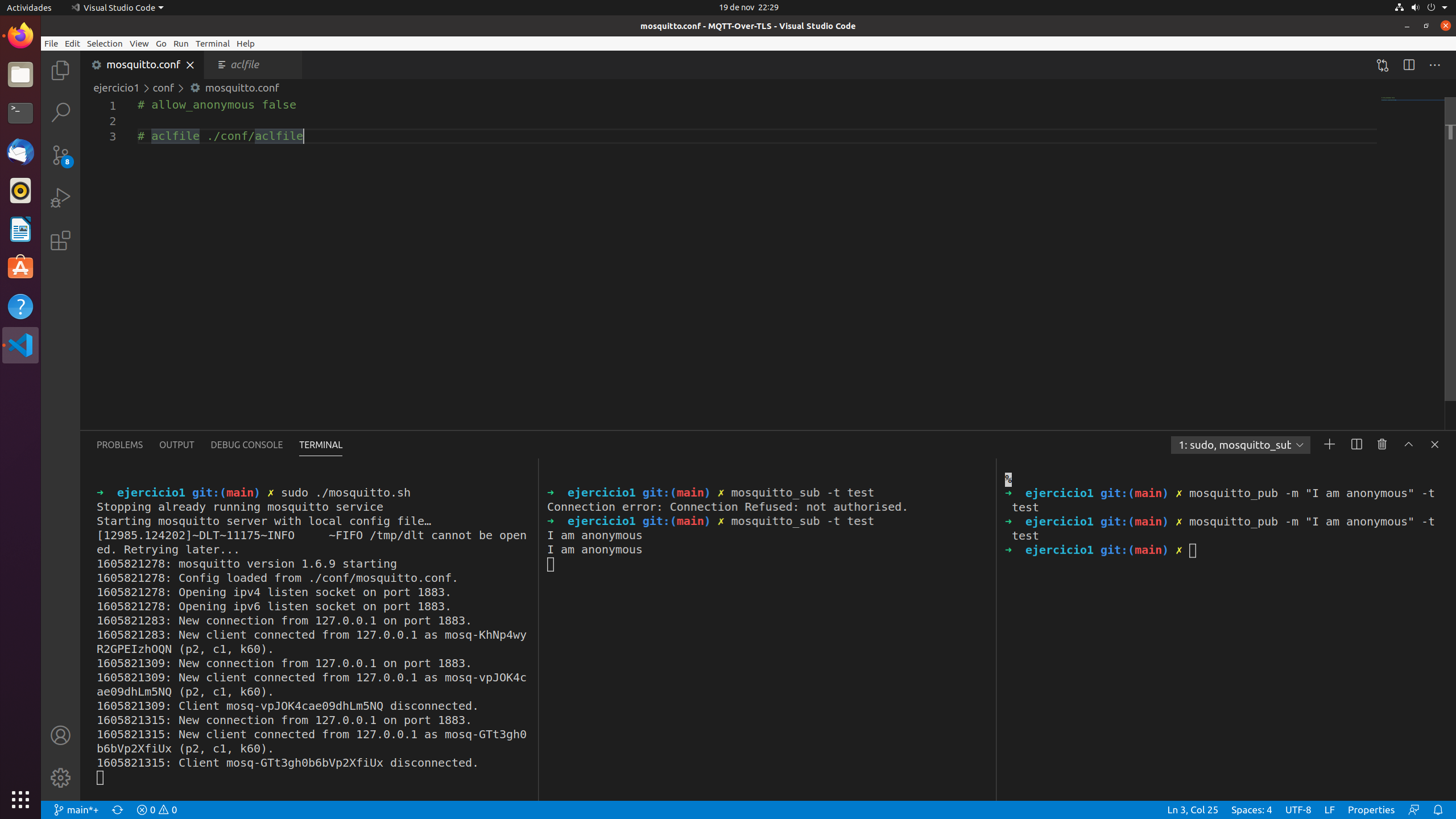


Figura 1: Output del mosquitto.sh del ejercicio 1

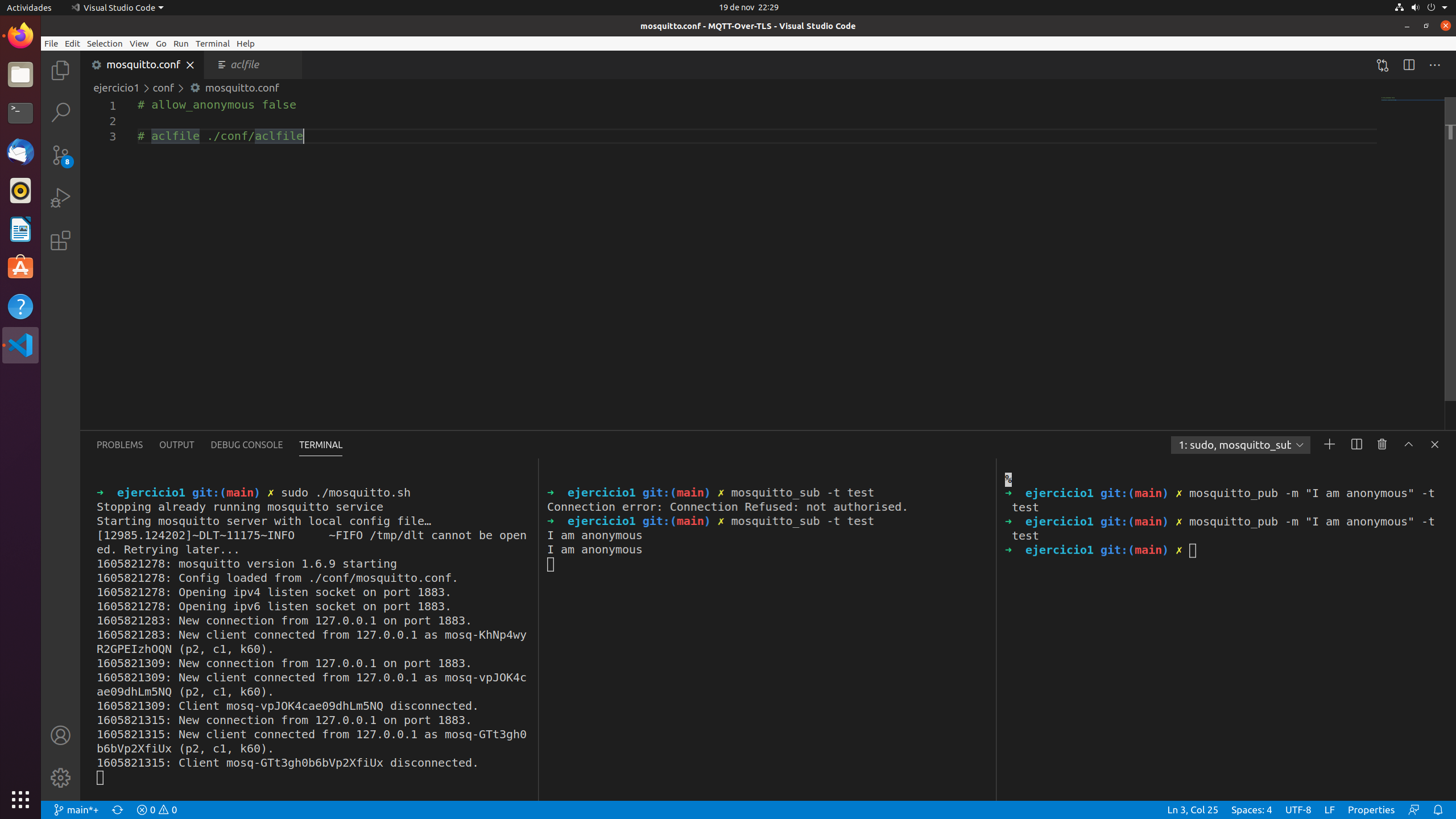


Figura 2 Suscriptor suscribiendose al topico "test" y recibiendo mensajes

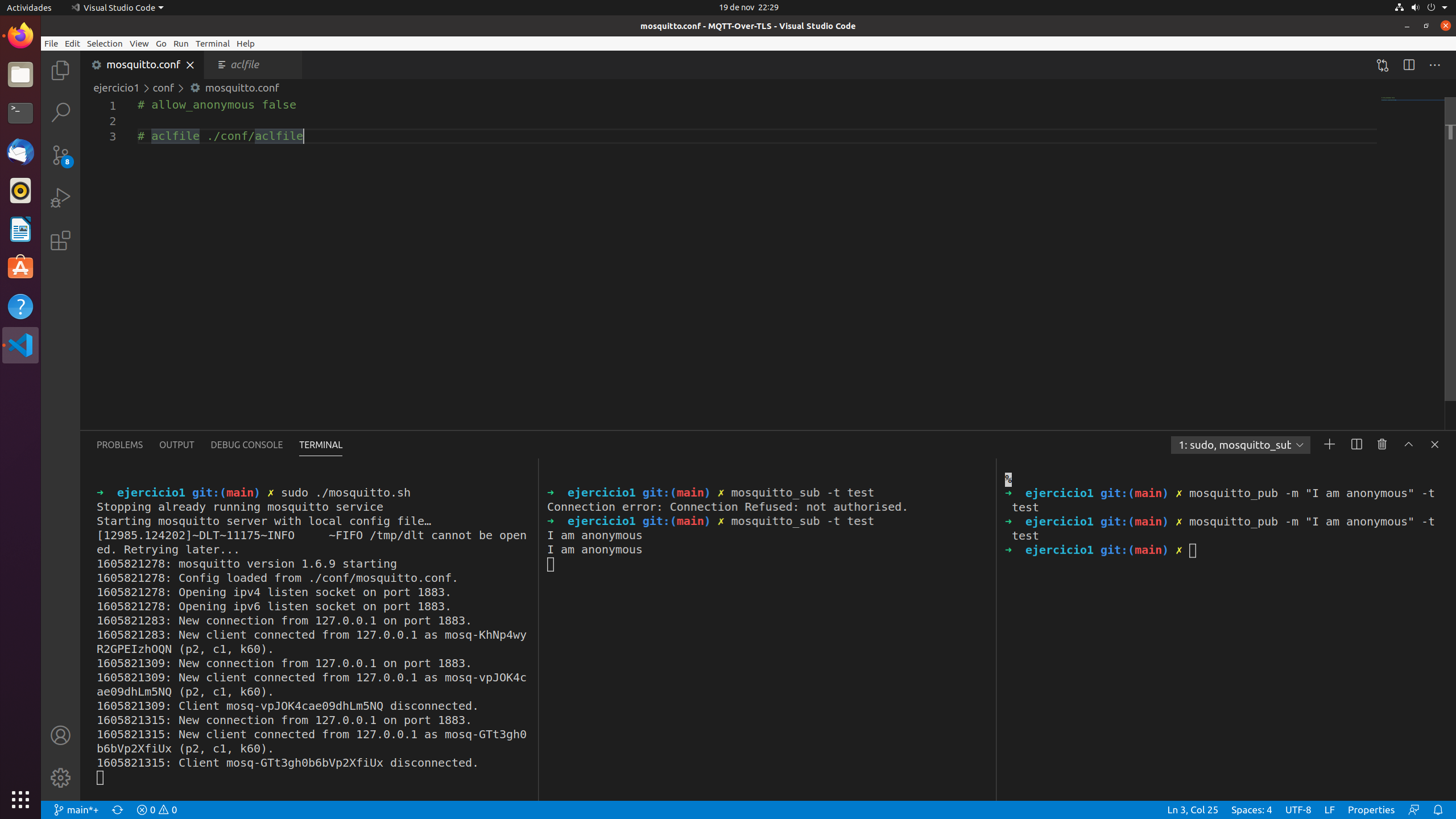


Figura 3 Publicador enviando mensajes en el topico "test"

A continuación, vamos a restringir el acceso de los usuarios no autenticados. Para ello vamos a poner la opción **allow\_anonymous** a false de tal manera que rechazamos todas aquellas conexiones sin credenciales de usuario.

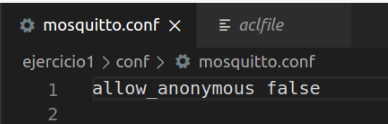


Figura 4 Linia del fichero mosquitto.conf que restringe el acceso a usuarios anónimos

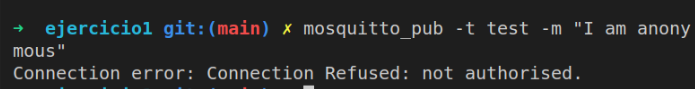
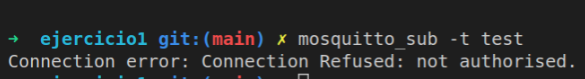


Figura 5 Clientes anónimos que no pueden conectarse al servidor

Para podernos conectar de nuevo al servidor vamos a dar de alta una serie de usuarios. Nos vamos a ayudar del script **créate\_users.sh** para darnos más prisa. El script define un par de usuarios con el mismo username y contraseña almacenándolos en el fichero **pwfile** dentro del directorio **conf**.

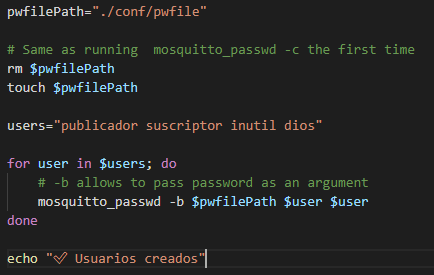


Figura 6 Código fuente del create\_users.sh

Ahora solamente nos hace falta referenciar el fichero **pwfile** desde el mosquitto.conf.

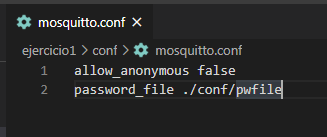


Figura 7 Versión final del mosquitto.conf del ejercicio1

Con esto hecho podemos reiniciar el servidor y lanzar los mismos comandos que antes, pero indicando las credenciales de los usuarios y deberíamos poder enviar y recibir mensajes.

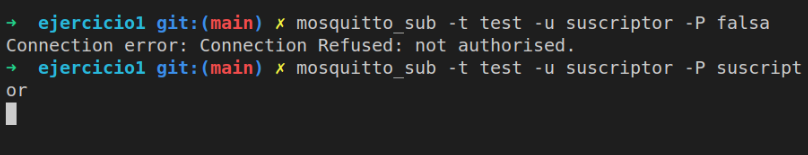


Figura 8 Enel primer comando intentamos suscribirnos con unas credenciales invalidas y el servidor nos desconecta. En el siguiente comando usamos las credenciales correctas y ya no recibimos el "Connection Refused: not authorised"

# Ejercicio 2

**Ejercicio: (2 puntos):** Configurar el broker para que pueda controlar el acceso (lectura-escritura-ambos) a los *topics* por parte de los clientes. Configura algunos casos de diferentes *topics* con diferentes permisos para que se pueda comprobar la funcionalidad de ACL del broker mosquitto.

Como se ha comentado en el ejercicio anterior, los suscriptores sólo reciben los mensajes de los temas o *topics* a los que están subscritos. **Mosquitto** nos permite implementar restricción de acceso a recursos (en este caso temas), es decir: nos permite gestionar qué usuarios pueden enviar o recibir mensajes de cada tema.

Estas reglas se configuran en el fichero ACL (*Access Control Lists*) o *acl file*. En nuestro caso definiremos las siguientes reglas:

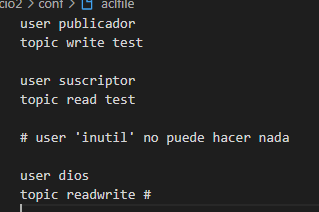


Figura 9 El usuario publicador puede publicar en test, dios puede hacer de todo en cualquier topic, suscriptor solo puede leer en test e inútil no puede hacer nada

Una vez configuradas las reglas debemos referenciar tal fichero desde **mosquitto.conf**:

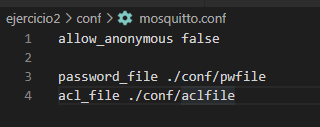


Figura 10 Versión final del mosquitto.conf del ejercicio2

Con todo esto hecho podemos verificar el correcto funcionamiento de nuestras configuraciones. Para ello debemos arrancar primero el servidor mosquitto y luego ejecutar el fichero **simulation.sh**.

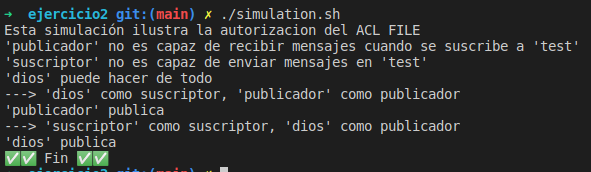


Figura 11 Output de la ejecución del fichero simulation.sh

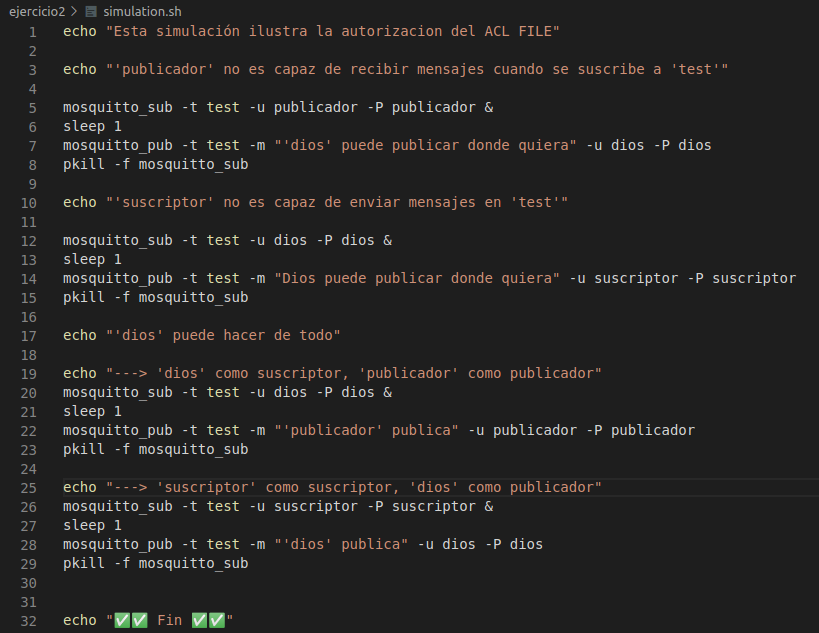


Figura 12 Código fuente el simulation.sh

Como esto puede ser poco ilustrativo vamos a revisar paso a paso las figuras 11 y 12 para demostrar que nuestra simulación se ha ejecutado acordemente.

Primero de todo lanzamos un proceso en segundo plano que se suscribe en el tópico “test” con el usuario “publicador”, que no tiene permisos. Esperamos un segundo antes de hacer que el usuario “dios” publique en el tópico “test” y vemos que no hay ningún output adicional. Las credenciales de “publicador” son válidas (y por eso el proceso no muere inmediatamente) pero no tiene derecho a recibir mensajes desde el tópico “test” y por ende no se imprime por **stdout** el mensaje que “dios” ha publicado.

A continuación, tenemos el caso inverso: lanzamos a “dios” en background para escuchar mensajes en el tópico “test” pero no tenemos output ya que el usuario que intenta publicar es “suscriptor” y no tiene permisos. Obtuvimos el mismo resultado que antes ya que “suscriptor” usa credenciales válidas, pero no tiene permisos y por ende “dios” no llega a escuchar nada.

Finalmente probamos los casos que “están bien”:

1. Lanzamos a “dios” como suscriptor y “publicador” publica: podemos ver como se imprime *‘publicador’ publica* por la consola de comandos justodespués del comando echo.
2. Lanzamos a “suscriptor” como suscriptor y “dios” publica: podemos ver como se imprime *‘dios’ publica* por la consola de comandos justodespués del comando echo.

Estos mensajes que aparecen en pantalla son los argumentos que reciben por el argumento

*-m* (message) dentro del script de simulación.

# Ejercicio 3

**Ejercicio: (3 puntos)**Seguir las indicaciones dadas para establecer una conexión TLS cifrada con autenticación del broker por parte del cliente.

Para establecer una comunicación bajo TLS lo primero que debemos hacer es generar una autoridad de certificación (*Certification Authority, CA*) de la cual tanto el servidor como el cliente deben fiarse. De tal manera que los certificados que generemos a posteriori del cliente y servidor estén firmados por esta CA. Cuando el cliente reciba un certificado por TLS del servidor podrá decidir si fiarse o no basándose en si ese certificado lo acredita una CA en la que confía. Lo que se pretende es que el cliente, antes de establecer la comunicación con el bróker, determine si otro “es quien dice ser”.

Para resolver el ejercicio 3, hemos creado una CA como se puede observar en las figuras 13 y 14.

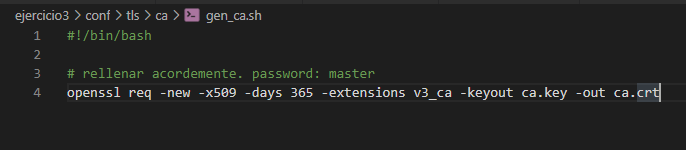


Figura 13 Script gen\_ca.sh que genera los archivos necesarios de la certification authority

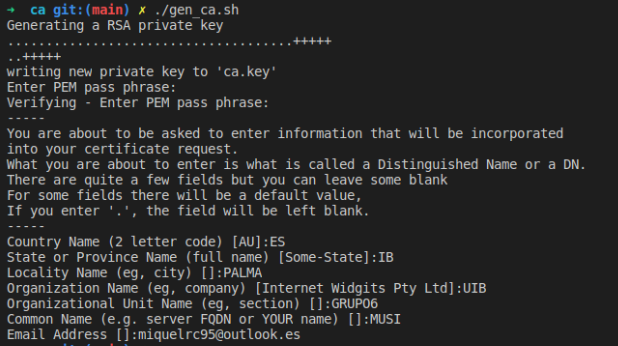


Figura 14 Rellenado del formulario para generar la CA

Posteriormente generamos las claves y certificados del bróker firmándolos por la CA como se puede ver en la figura 15.

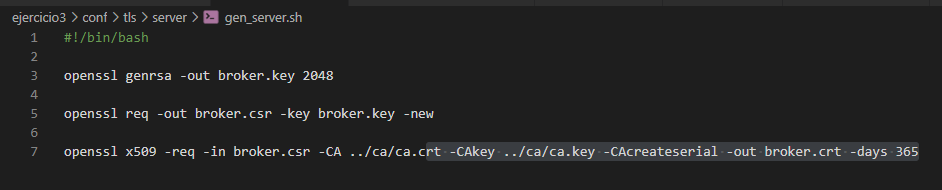


Figura 15 Script gen\_server.sh que genera los certificados y claves del servidor firmándolos por nuestra CA generada préviamente

Una vez generados los certificados y juegos de claves debemos referenciarlos desde el fichero **mosquitto.conf**. Además, cambiamos el puerto al 8883, que es donde suele correr el MQTT-over-TLS e indicamos que versión de TLS va a usar nuestro servidor. En nuestro caso será la versión 1.2.

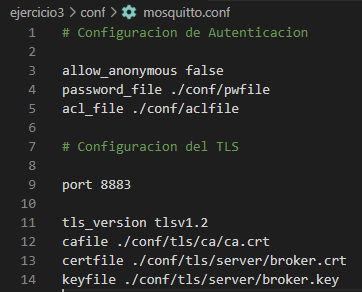


Figura 16 Configuracion final y correcta para el mosquitto en el ejercicio3

Para demostrar que todo esto funciona y no nos estamos tirando un farol hemos preparado dos simulaciones para saber si el tráfico va cifrado o no por la red. Los scripts **sniff1883.sh** y **sniff8883.sh** son utilidades que hemos creado para poder ver el tráfico que circula en susodichos puertos. Mientras que el script **unsafemosquitto.sh** arranca un servidor mosquitto referenciando al fichero **unsafemosquitto.conf** cuyo contenido se muestra en la figura 17.

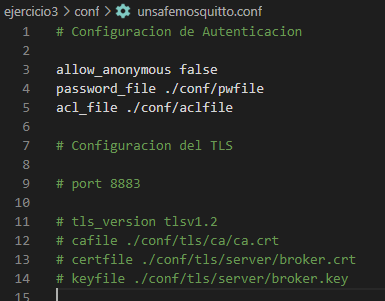


Figura 17 Contenido del unsafemosquitto.conf. No tiene ninguna opción de TLS configurada

A continuación, se presenta la ejecución del script **unsafe\_pub.sh** donde simplemente se publica un mensaje sin usar configuración de TLS alguna mientras nuestro script **sniff1883.sh** ya está corriendo.

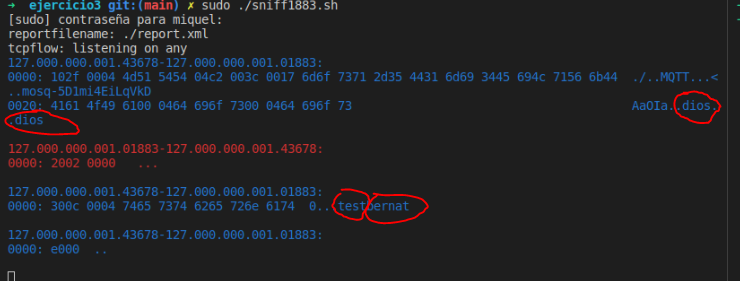


Figura 18 Podemos observar el payload del mensaje MQTT, en el las credenciales del usuario "dios" asi como el tópico "test" y el contenido del mensaje: "bernat". Claramente el tráfico no esta siendo cifrado.

Una vez hecho esto usamos la configuración de la figura 16 y repetimos el ejercicio: Ahora usando **sniff8883.sh** y el script **safe\_pub.sh**.

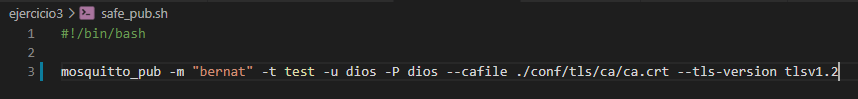


Figura 19 El mismo código que publica el mismo mensaje en el mismo tópico pero esta vez usando la configuración TLS

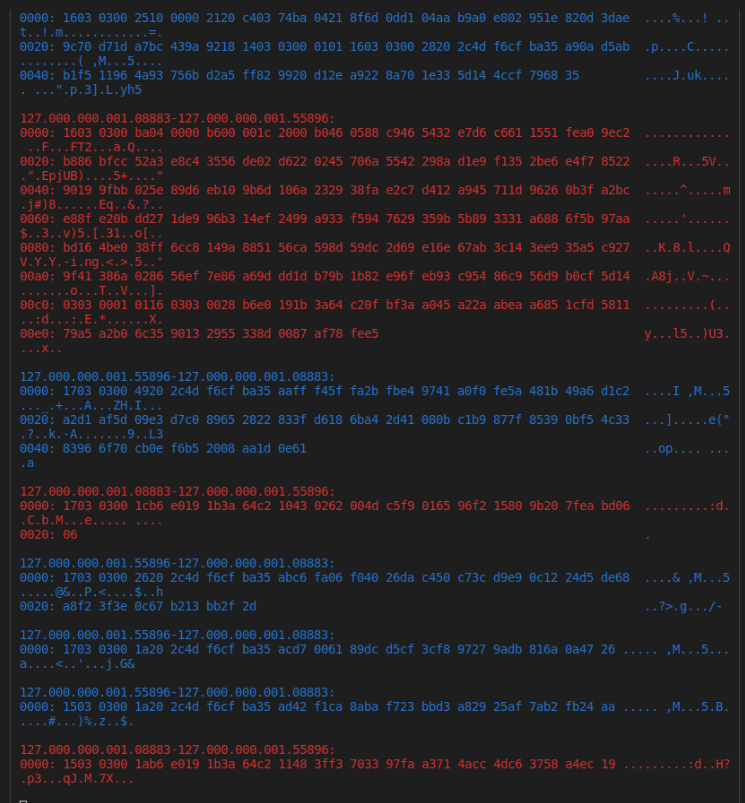


Figura 20 Output del "sniffer" de tráfico. No entendemos nada ya que todo está cifrado

# Ejercicio 4

**Ejercicio: (4 puntos)**Generar el material necesario y configurar cliente y servidor para establecer una conexión TLS cifrada con autenticación del broker por parte del cliente, y del cliente por parte del broker.

El bróker mosquitto tiene una opción de configuración para que el servidor valide también al cliente. Para ello hay que modificar el fichero **mosquitto.conf** y añadir lo siguiente:

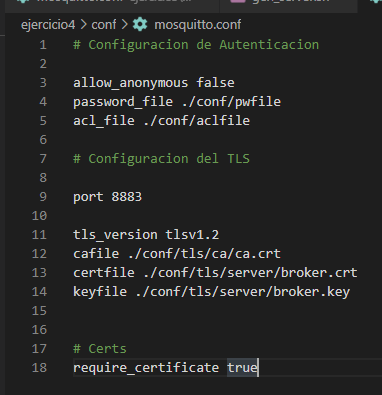


Figura 21. Añadimos la nueva opción: require\_certificate true para requerir que los clientes se autentiquen con sus certificados

Desde este momento, el servidor mosquitto requerirá un certificado a todos los clientes que se intenten conectar.

Por tanto, igual que se ha hecho anteriormente con el servidor, se crea un certificado y un par de claves para el cliente y se firman por la autoridad de certificación que creamos antes.

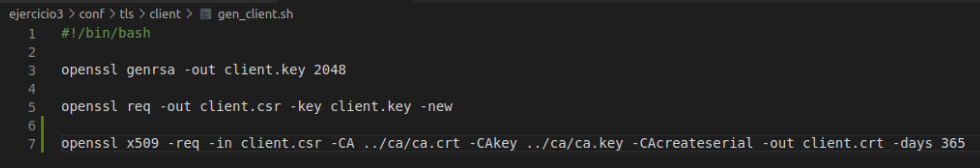


Figura 22: Código gen\_client.sh que genera el certificado a usar desde los clientes de mosquitto

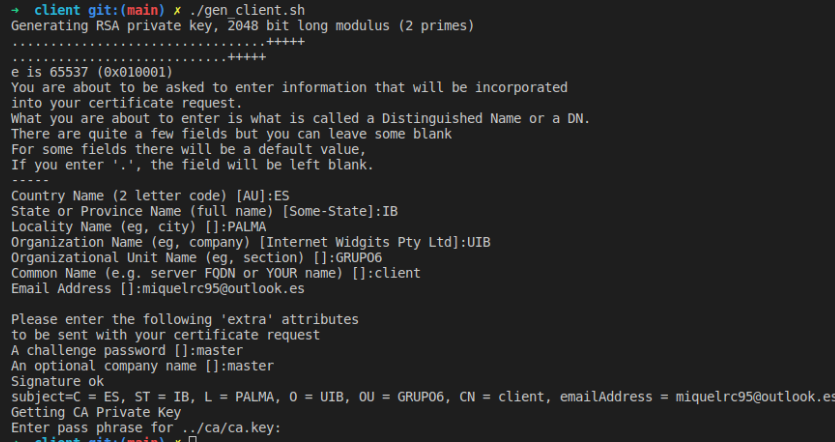


Figura 23. Rellenado del formulario de la generacion de los certificados de los clientes

Y finalmente hacemos una simulación de test: Probamos de publicar con una configuración en la que **PRIMERO NO** enviamos los datos de nuestro certificado como clientes y otra en la que **SÍ**. Esto se ve ilustrado por las figuras 24 y 25.

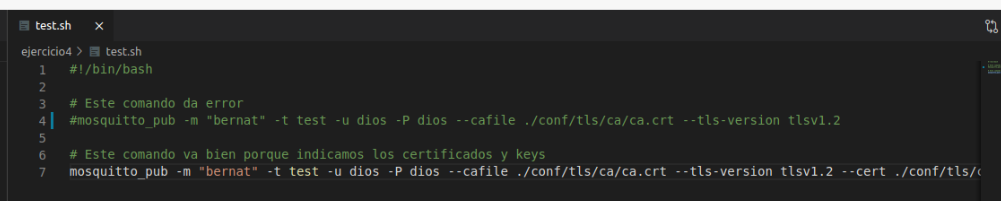


Figura 24: Código del script test.sh donde tenemos las dos instrucciones necesarias para poder verificar si nuestra configuración funciona

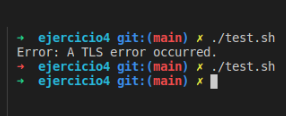


Figura 25: Ejecutar el script sin enviar nuestro certificado resulta en "Error: A TLS error occurred.". Mientras que la segunda vez el comando finaliza de manera satisfactoria