Protocolo MQTT

|  |  |
| --- | --- |
| Fecha | 20210113 |
| Autor | Ramón Junquera |

**Índice**

[​ Introducción 1](#__RefHeading___Toc291_451637037)

[​ Instalación de mosquitto 3](#__RefHeading___Toc579_1953494034)

[​ Suscripción y publicación simple 3](#__RefHeading___Toc581_1953494034)

[​ Debug 4](#__RefHeading___Toc583_1953494034)

[​ Comandos generales 4](#__RefHeading___Toc585_1953494034)

[​ Brokers públicos 4](#__RefHeading___Toc587_1953494034)

[​ Archivo de usuarios 5](#__RefHeading___Toc589_1953494034)

[​ Permisos 6](#__RefHeading___Toc591_1953494034)

[​ Conexión segura 10](#__RefHeading___Toc859_3028098233)

[​ Conexión segura con usuarios y reglas 13](#__RefHeading___Toc896_3028098233)

[​ Certificados de cliente 15](#__RefHeading___Toc898_3028098233)

[​ Funcionalidades adicionales 17](#__RefHeading___Toc861_3028098233)

# Introducción

En el mundo IoT las comunicaciones son esenciales.

Habitualmente tenemos en los extremos del canal de comunicación un cliente y un servidor.

El cliente es un dispositivo que se dedica a recopilar información que envía a un servidor.

El servidor recibe la información y la procesa, ya sea generando eventos instantáneos o guardándola para analizarla posteriormente.

El caso más sencillo es el de un dispositivo que necesita transmitir cierta información a otro.

Podríamos conectar ambos con una simple conexión tcp, un websocket, o incluso una conexión udp si estamos dispuestos a perder mensajes. El dispositivo servidor está escuchando ciertos puertos ip y atiende las peticiones de conexión, las conexiones existentes y sus mensajes, dando respuesta a cada una de ellas.

El dispositivo cliente se limita a solicitar al servidor una conexión, transmitir su mensaje y esperar la respuesta, si corresponde.

No es raro que los dispositivos sean mixtos. No sólo envían mensajes sobre los eventos locales, sino que reciben órdenes de otros dispositivos para lanzar eventos.

Si complicamos la infraestructura con varios (o miles) de dispositivos, algunos mixtos, lanzando mensajes entre sí, tenemos un problema: el dimensionamiento del servicio de recepción de mensajes.

Un dispositivo tiene una potencia de cálculo limitada que no puede desperdiciar gestionando el servidor de comunicaciones. Sobre todo cuando podríamos tener gran cantidad de clientes.

Para resolver el problema aparecen protocolos de como MQTT, XMPP o AMQP.

Este documento se centrará en MQTT por ser uno de los más populares hoy en día.

MQTT (Message Queue Telemetry Transport) fue desarrollado por IBM y publicado con una licencia sin derechos.

La solución pasa por crear un servidor de mensajería intermedio que descarga la gestión de comunicaciones a los dispositivos.

Inicialmente, este servidor no almacenará información, simplemente la recibirá y la enviará. Es un broker.

Los mensajes tendrán una estructura definida por el protocolo.

El mensaje más sencillo tendrá dos propiedades:

* El propio mensaje. Lo que se desea comunicar.
* Un tópico o asunto que facilitará su clasificación.

Los tópicos estarán jerarquizados. Los distintos niveles se separarán con el carácter "/". Como las carpetas de un sistema de archivos. Ej: asignaturas/mates/polinomios

Los tópicos siempre se escriben completos, comenzando desde la raíz.

La primera acción es publicar un mensaje en un broker.

Crearemos un mensaje con su contenido y tópico y lo enviaremos al broker.

No nos preocupamos de quién recibe el mensaje, puesto que de eso se encargará el broker.

Para poder conocer/recibir los mensajes que recibe un broker sobre cierto tópico tenemos que suscribirnos al tópico.

La suscripción es indicarle al broker que estamos interesados en los mensajes de cierto tópico.

El broker toma nota de las preferencias y mantiene abierto el canal de comunicación.

Cada vez que se reciba un mensaje sobre cierto tópico, el broker comprobará el listado de comunicaciones que tiene abiertas que están interesadas en él y les enviará una copia.

Esto significa que desde el dispositivo también se mantiene abierta la comunicación.

Periódicamente un heartbeat que permite saber si la comunicación se ha perdido o aun se mantiene.

Por lo tanto, tenemos que un dispositivo con una potencia de cálculo reducida es capaz de enviar o recibir mensajes

desde un broker (servidor especializado en la distribución de mensajes) de manera relativamente cómoda.

Puesto que los tópicos están jerarquizados, tenemos posibilidad de utilizar caracteres comodín para suscribirnos a más de un tópico.

* #. Significa: y cualquier otro sub-tópico. Siembre debe ser utilizado el final.
* +. Significa: cualquier tópico.

Pondremos algún ejemplo.

Supongamos que tenemos la siguiente jerarquía de tópicos:

**asignaturas**

**mates**

**polinomios**

**radicales**

**exámenes**

**lengua**

**fonemas**

**exámenes**

Si nos suscribimos a asignaturas/mates/#, recibiremos los mensajes de los tópicos:

* asignaturas/mates/polinomios/polinomios
* asignaturas/mates/polinomios/radicales
* asignaturas/mates/polinomios/exámenes

Si nos suscribimos a asignaturas/+/exámenes, recibiremos los mensajes de los tópicos:

* asignaturas/mates/exámenes
* asignaturas/lengua/exámenes

Podemos utilizar los caracteres comodín combinados en la especificación del tópico de suscripción.

Los brokers sin seguridad son excelentes para hacer pruebas, pero en entornos productivos se les añade identificación y encriptación.

Gracias a ello, se pueden especificar reglas que impidan la suscripción de cualquiera a cierta rama de un tópico.

Los puertos tcp utilizados por defecto en cada caso son:

* 1883 sin seguridad
* 8883 con seguridad

# Instalación de mosquitto

Pasemos a la práctica.

Utilizaremos la aplicación Eclipse Mosquitto para hacer unas prácticas, puesto que es ligera, versátil y muy popular.

Las siguientes explicaciones corresponden a Linux como sistema operativo, aunque la aplicación existe también para Windows y Mac.

Instalación del broker y el cliente de Mosquitto:

sudo apt-get install mosquitto mosquitto-clients

Con el broker se instala la parte del servidor que se ejecuta en background a través de un servicio.

Una vez instalado, el servicio se pone en marcha automáticamente.

Con el cliente se instalan dos consolas, una para publicar (mosquitto\_pub) y otra para suscribirse (mosquitto\_sub).

# Suscripción y publicación simple

Solicitaremos al broker que nos informe de un tópico concreto...nos suscribiremos.

El tópico será "asignaturas/mates". El comando es:

mosquitto\_sub -t "asignaturas/mates"

Puesto que la conexión debe permanecer abierta, tras lanzar el comando no se finalizará la ejecución hasta que no se le obligue con CTRL+C. Mientras tanto se mostrarán en pantalla los mensajes recibidos.

Desde otra terminal publicaremos un mensaje a ese mismo tópico para comprobar si funciona, con el comando:

mosquitto\_pub -t "asignaturas/mates" -m "Hola!"

El mensaje se envía al broker, que lo recibe, lee el tópico, comprueba si tiene alguna conexión suscrita a ese tópico, la encuentra y le envía una copia del mensaje, que es recibida por el cliente suscrito y presentada en pantalla.

Si no hay suscriptores del tópico de un mensaje publicado, se pierde sin notificarlo a nadie.

Hemos utilizamos muy pocos parámetros porque hemos dado algunos por supuestos:

* El broker se encuentra en la misma máquina (localhost)
* El puerto utilizado es el estándar (1883)

Los comandos completos si detallamos estos datos serían...

**Suscripción**: mosquitto\_sub -h localhost -p 1883 -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -h localhost -p 1883 -t "asignaturas/mates" -m "Hola!"

# Debug

Ni la suscripción ni la publicación nos muestran detalles adicionales. Normalmente se utilizan así. Pero con el parámetro -d podemos solicitar que nos muestre todos los mensajes internos. Lo probaremos:

**Suscripción**: mosquitto\_sub -d -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -d -t "asignaturas/mates" -m "Hola!"

En una suscripción comprobamos como hay un proceso inicial de solicitud de conexión al broker y tras ser aceptada, se solicita la suscripción al tópico indicado, que es aceptada.

Además podemos ver como periódicamente el cliente envía un ping al broker para verificar que la comunicación se mantiene y el broker responde.

En una publicación pasa algo parecido. Se solicita una conexión al broker, que la acepta, solicita la publicación del mensaje y a continuación desconecta.

# Comandos generales

La configuración del broker se guarda en la carpeta etc/mosquitto

El archivo principal de configuración es etc/mosquitto/mosquitto.conf

En la misma carpeta tenemos algunos archivos más de ejemplo.

Se guarda una copia de esos mismos archivos en usr/share/doc/mosquitto/examples.

Esta es una lista de los comandos de gestión del servicio más comunes:

* Comprobación del estado del servicio: sudo systemctl status mosquitto
* Arrancar el servicio: sudo systemctl start mosquitto
* Parar el servicio: sudo systemctl stop mosquitto
* Parar y arrancar el servicio: sudo systemctl restart mosquitto
* Añadir el servicio al iniciar sistema: sudo systemctl enable mosquitto
* Eliminar el servicio del inicio de sistema: sudo systemctl disable mosquitto
* Comprobar el archivo de log: sudo gedit /var/log/mosquitto/mosquitto.log

# Brokers públicos

Existen brokers MQTT públicos para poder probar clientes. Nunca deberían ser utilizados en producción.

Uno de ellos es **broker.emqx.io**

Haremos la misma prueba con este servidor para comprobar que también funciona bien.

Omitimos el parámetro del puerto porque utilizaremos el estándar.

**Suscripción**: mosquitto\_sub -h broker.emqx.io -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -h broker.emqx.io -t "asignaturas/mates" -m "Hola!"

Probaremos los caracteres comodín...

**Suscripción**: mosquitto\_sub -h broker.emqx.io -t "asignaturas/#"

**Publicación**: mosquitto\_pub -h broker.emqx.io -t "asignaturas/mates" -m "Hola!"

# Archivo de usuarios

Volveremos al broker local para hacer algunas modificaciones en su configuración.

Añadiremos algo de seguridad. Crearemos algún usuario con su correspondiente contraseña.

Guardaremos la información de usuarios en el archivo **/etc/mosquitto/pwfile**

Podemos crear tantos usuarios como necesitemos.

Crearemos el primer usuario. Sus credenciales serán usuario **user1** y contraseña **Puser1**

Como al principio no existe el archivo pwfile utilizaremos el siguiente comando:

sudo mosquitto\_passwd -c /etc/mosquitto/pwfile user1

Nos pedirá que escribamos su contraseña y la verifiquemos.

El parámetro -c indica que queremos crear el archivo de usuarios.

Podemos comprobar que como el archivo se ha creado y contiene una sólo linea que comienza con el nombre de usuario y separado por **:** se añade la contraseña codificada (hash).

Es muy importante recordar que el parámetro -c sólo se debe utilizar para crear el archivo si no existe, porque si ya existe, lo sobreescribe perdiendo cualquier información que pueda contener.

Añadiremos un segundo usuario con credenciales **user2** y **Puser2**

Esta vez utilizaremos el mismo comando pero sin parámetros:

sudo mosquitto\_passwd /etc/mosquitto/pwfile user2

También nos solicita la contraseña que queremos asignar y que la verifiquemos.

A continuación muestra el actual contenido del archivo de usuarios y añade una nueva línea al final.

Por último añadiremos un tercer usuario con credenciales **user3** y **Puser3**

Pero aprovecharemos la posibilidad de incluir la contraseña en la línea de comandos con el parámetro -b:

sudo mosquitto\_passwd -b /etc/mosquitto/pwfile user3 Puser3

Directamente muestra e contenido actual del archivo y añade una nueva línea al final.

Borraremos el usuario **user2** con el comando:

sudo mosquitto\_passwd -D /etc/mosquitto/pwfile user2

Podemos borrar todos los usuarios de uno en uno. Si lo hacemos, el archivo no desaparecerá. Simplemente quedará vacío (tamaño 0).

Existe una última vía para crear usuarios. Se utiliza cuando queremos dar de alta muchos usuarios de una vez.

Se crea manualmente el archivo de usuarios (pwfile) y se completa con una línea por usuario.

Comienza con el nombre de usuario, seguido por **:** y finalmente la contraseña. Sin espacios.

Todo en texto plano.

Este archivo no puede ser utilizado por la aplicación directamente. Debemos escribir:

sudo mosquitto\_passwd -U /etc/mosquitto/pwfile

Esto calculará el hash de las contraseñas (las encriptará) y las sustituirá.

Ahora que sabemos crear un archivo de usuarios debemos indicar en la configuración que debe tenerlo en cuenta.

Editamos el archivo de configuración:

sudo gedit /etc/mosquitto/mosquitto.conf

Y añadimos una nueva línea al final:

password\_file /etc/mosquitto/pwfile

Guardamos y salimos.

Es necesario reiniciar el servicio para obligarle a leer de nuevo el archivo de configuración.

Esto ocurre cuando cambiamos la configuración o los usuarios.

sudo systemctl restart mosquitto

Cuando indicamos que trabajaremos con un archivo de usuarios se desactiva automáticamente la opción de aceptar las conexiones anónimas.

A partir de ahora no se permiten conexiones anónimas tanto para suscribirse como para publicar hay que identificarse.

Haremos una suscripción con user1 y una publicación con user3:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "asignaturas/mates" -m "Hola!"

Y si lo intentamos sin credenciales o con credenciales erróneas nos dará un error.

Habitualmente nos interesa que además de tener usuarios registrados, se acepten las conexiones de usuarios anónimos. Para ello necesitamos especificar la siguiente línea en el archivo de configuración:

allow\_anonymous true

Recordando que se debe reiniciar el servicio para que tenga en cuenta los cambios realizados en cualquiera de los archivos de configuración.

# Permisos

Tener usuarios registrados está bien porque podríamos impedir que nuestro broker quede abierto al público en general, pero el potencial está en poder definir reglas/permisos respecto a los tópicos que puede utilizar cada uno.

El listado de reglas se denomina ACL (Access Control List) y al igual que con los usuarios, se guarda en un archivo de configuración adicional. Normalmente **/etc/mosquitto/aclfile**

Tenemos un ejemplo de este archivo en la carpeta de configuración.

El archivo se estructura en 3 secciones que deben mantenerse en el siguiente orden:

* Reglas de usuarios anónimos. Líneas que comienzan con el comando **topic**.
* Reglas de usuarios registrados. Grupos de líneas que componen las reglas de un usuario. La primera línea del grupo comienza con el comando **user** y continúa con una o más líneas de comando **topic**.
* Reglas generales. Reglas grupales definidas por patrones (caracteres comodín). Líneas que comienzan con el comando **pattern**.

Las líneas vacías o que comienzan por # no se tendrán en cuenta.

En el archivo de configuración tendremos que hacer algún cambio para que se tenga en cuenta este archivo. Además aprovecharemos para incluir el parámetro de indica explícitamente si se permiten las conexiones anónimas…

Editamos el archivo de configuración:

sudo gedit /etc/mosquitto/mosquitto.conf

Incluimos las siguientes líneas al final:

allow\_anonymous true

acl\_file /etc/mosquitto/aclfile

Guardamos y salimos.

No reiniciaremos el servicio porque todavía no hemos creado ni el archivo ACL.

Presentaremos varios ejemplos y los comentaremos.

Para todos los ejemplo permitiremos las conexiones anónimas.

El procedimiento siempre es el mismo:

Abrimos el archivo ACL:

sudo gedit /etc/mosquitto/aclfile

Lo editamos incluyendo las reglas del ejemplo. Guardamos, salimos y reiniciamos el servicio:

sudo systemctl restart mosquitto

Comencemos con los ejemplos…

El archivo ACL contiene:

topic public/#

user user1

topic #

user user3

topic asignaturas/#

Las reglas son las siguientes:

* Los usuarios anónimos sólo pueden publicar y suscribirse bajo el tópico raíz **public**.
* El usuario **user1** tiene permisos globales. Puede publicar y suscribirse a cualquier tópico.
* El usuario **user3** sólo puede publicar y suscribirse a bajo el tópico **asignaturas**.

Comprobamos que una conexión anónima puede publicar y suscribirse bajo el tópico **public**:

**Suscripción**: mosquitto\_sub -t "public/#"

**Publicación**: mosquitto\_pub -t "public/test" -m "Hola!"

Comprobamos si una conexión anónima pueden publicar y suscribirse bajo otro tópico que no sea **public**:

**Suscripción**: mosquitto\_sub -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -t "asignaturas/mates" -m "Hola!"

Descubrimos que no aparece ningún error ni en la suscripción ni en la publicación...pero no funciona. Es correcto!

Comprobamos que el **user1** puede suscribirse y publicar en cualquier tópico.

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "asignaturas/mates"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "asignaturas/mates" -m "Hola!"

Incluso puede suscribirse a tópicos permitidos para usuarios anónimos:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "public/#"

**Publicación**: mosquitto\_pub -t "public/test" -m "Hola!"

O publicar en tópicos para los usuarios anónimos:

**Suscripción**: mosquitto\_sub -t "public/#"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "public" -m "Hola!"

Comprobamos que **user3** puede suscribirse a sub-tópicos de **asignaturas**:

**Suscripción**: mosquitto\_sub -u user3 -P Puser3 -t "asignaturas/#"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "asignaturas/mates" -m "Hola!"

Pero **user3** no tiene permisos para publicar en el tópico de los usuarios anónimos:

**Suscripción**: mosquitto\_sub -t "public/#"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "public/test" -m "Hola!"

Podemos asignar más de una regla a un usuario?

Sí. Simplemente añadiendo más líneas en la sección que corresponda.

Basado en el ejemplo anterior, queremos que **user3** pueda trabajar con los tópicos de los usuarios anónimos. Simplemente añadimos una nueva línea al grupo de permisos del **user3**:

topic public/#

user user1

topic #

user user3

topic asignaturas/#

topic public/#

Repetimos la última prueba para comprobar que ahora **user3** puede publicar en el tópico de los usuarios anónimos:

**Suscripción**: mosquitto\_sub -t "public/#"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "public/test" -m "Hola!"

Es obligatorio que los permisos a un tópico siempre sean de lectura/escritura?

No.

Hasta ahora hemos utilizado reglas del tipo **topic “asignaturas/mates”**, que dan permisos de lectura y escritura. Es lo mismo que escribir **topic readwrite “asignaturas/mates”**.

Pero podríamos especificar sólo **read** o sólo **write**.

Hagamos un ejemplo.

El contenido del archivo ACL es:

topic read public/#

user user1

topic #

Comprobamos que una conexión anónima se puede suscribir a sub-tópicos de **public**:

**Suscripción**: mosquitto\_sub -t "public/#"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "public/test" -m "Hola!"

Pero no puede publicar:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "public/#"

**Publicación**: mosquitto\_pub -t "public/test" -m "Hola!"

Hasta ahora hemos utilizado permisos para usuarios anónimos y registrados.

Haremos algún ejemplo simple de reglas generales.

El contenido del archivo ACL es:

user user1

topic #

pattern asignaturas/#

Las reglas son:

* El usuario **user1** tiene permisos globales. Puede trabajar con cualquier tópico.
* Cualquier usuario (anónimo o no) puede trabajar con sub-tópicos de **asignaturas**.
* Cualquier usuario anónimo o distinto de **user1**, no puede trabajar con ningún tópico que no esté bajo **asignaturas**.

Vemos que las reglas se procesan tal y como aparecen en el archivo ACL. En cuanto se encuentra una regla que incluya el tópico procesado, se tiene en cuenta y se deja de buscar más. Esto ocurre cuando **user1** intenta publicar en un tópico que no pertenece a asignaturas. Esta acción se procesa en la línea topic # que le otorga permisos globales. No continúa leyendo la siguiente línea.

Comprobamos que **user1** puede trabajar con tópicos distintos a **asignaturas**:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "public"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "public" -m "Hola!"

Comprobamos que **user1** puede trabajar con **asignaturas**:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "asignaturas/#"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "asignaturas/mates" -m "Hola!"

Comprobamos que una conexión anónima no recibe mensajes de una suscripción que no pertenezca a **asignaturas**:

**Suscripción**: mosquitto\_sub -t "public"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "public" -m "Hola!"

Comprobamos que una conexión anónima puede trabajar con **asignaturas**:

**Suscripción**: mosquitto\_sub -t "asignaturas/#"

**Publicación**: mosquitto\_pub -t "asignaturas/mates" -m "Hola!"

Comprobamos que una **user3** puede trabajar con **asignaturas**:

**Suscripción**: mosquitto\_sub -u user3 -P Puser3 -t "asignaturas/#"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "asignaturas/mates" -m "Hola!"

Comprobamos que **user3** no puede suscribirse a tópicos distintos a **asignaturas**:

**Suscripción**: mosquitto\_sub -u user3 -P Puser3 -t "public"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "public" -m "Hola!"

El comando **pattern** admite el comodín **%u** que representa el nombre de usuario.

Desarrollaremos un ejemplo que lo demuestre.

El contenido del archivo ACL será:

pattern write correo/#

pattern read correo

pattern read correo/%u

Las reglas son:

* Cualquiera puede escribir en el tópico **correo** y sus sub-tópicos, para así poder mandar mensajes a cualquiera.
* Cualquiera puede leer el tópico **correo**. Sólo éste, porque no hemos añadido **/#** al final.
* Un usuario podrá leer los mensajes del tópico **correo/<nombre de usuario>**. No tendrá permiso de lectura del resto de sub-tópicos y no le llegarán.

Comprobamos que un usuario puede leer sus propios mensajes:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/user1"

**Publicación**: mosquitto\_pub -u user1 -P Puser1 -t "correo/user1" -m "Hola!"

Comprobamos que un usuario puede enviar un mensaje a otro:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/user1"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "correo/user1" -m "Hola!"

Comprobamos que un usuario no puede leer los mensajes de otro:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/user3"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "correo/user3" -m "Hola!"

Comprobamos que un usuario puede enviar mensajes a todos:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/#"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "correo" -m "Hola!"

Comprobamos que un usuario anónimo puede enviar mensajes a uno registrado:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/#"

**Publicación**: mosquitto\_pub -t "correo/user1" -m "Hola!"

Comprobamos que un usuario anónimo puede enviar mensajes a todos:

**Suscripción**: mosquitto\_sub -u user1 -P Puser1 -t "correo/#"

**Publicación**: mosquitto\_pub -t "correo" -m "Hola!"

Un usuario anónimo puede recibir mensajes enviados a todos:

**Suscripción**: mosquitto\_sub -t "correo/#"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "correo" -m "Hola!"

Comprobamos que un usuario anónimo no puede recibir mensajes de un usuario registrado:

**Suscripción**: mosquitto\_sub -t "correo/user3"

**Publicación**: mosquitto\_pub -u user3 -P Puser3 -t "correo/user3" -m "Hola!"

# Conexión segura

En los mensajes sin seguridad (no encriptados), las contraseñas se transmiten en texto plano y pueden ser capturadas.

No es una buena idea utilizar un sistema de credenciales y permisos sin encriptación.

Cuando activamos la encriptación necesitamos utilizar certificados y claves como en cualquier servidor web.

Estos archivos habitualmente se guardan en la carpeta /etc/mosquitto/cert y tienen los siguientes nombres:

* ca.crt Certificado de la autoridad certificadora (Certificate Authority = CA).
* server.key Clave privada del servidor.
* server.crt Certificado del servidor.

Estos archivos corresponden al broker (servidor).

Para crear los certificados utilizaremos la aplicación openssl, que se puede instalar con:

sudo apt-get install openssl

En los siguientes párrafos generaremos los archivos necesarios.

Ejecutaremos todos los comandos en una terminal abierta en la carpeta **Downloads**, que es donde se crearán los archivos.

Puesto que no vamos a trabajar con ninguna entidad certificadora, no necesitaremos pagar ni por la generación del certificado ni por su alojamiento.

Seremos nosotros mismos la entidad certificadora.

Lo primero que haremos será conseguir el certificado CA.

Para ello debemos generar un par de claves complementarias.

openssl genrsa -des3 -out ca.key 2048

Nos pide la contraseña. Para este ejemplo utilizaremos **passCA**. No pide verificar la contraseña.

Finalmente genera el archivo **ca.key**.

Con este archivo crearemos el certificado CA.

Supondremos que lo vamos a utilizar durante un año (365 días si el año actual no es bisiesto). Lo normal es que el certificado de una entidad certificadores dure más tiempo (5 años), para no tener que renovarlo cada año. Pero para esta prueba con un año es más que suficiente.

openssl req -new -x509 -days 365 -key ca.key -out ca.crt

Nos pide la contraseña de la clave CA que era **passCA**.

Nos pide el código de 2 letras del país. Supondremos que es España = SP.

Nos pide la provincia. Supondremos que es **Barcelona**.

Nos pide la localidad. Supondremos que también es **Barcelona**.

Nos pide el nombre de nuestra organización. Supondremos que es **RoJo**.

Nos pide el nombre de la unidad organizativa. Supondremos que es **IT**.

Nos pide el nombre del contacto o del servidor (ip o hostname). Supondremos que es **Melchor**.

Nos pide el correo del contacto. Supondremos que es **melchor@reyesmagos.com**

Y después de responder a todo esto, se genera el archivo del certificado (**ca.crt**).

Por supuesto que para las pruebas que estamos haciendo no tiene demasiada importancia la respuesta que demos a las preguntas de generación de certificado.

Ahora generaremos un par de claves complementarias para el servidor/broker.

openssl genrsa -out server.key 2048

Crearemos una solicitud de certificado (archivo .csr):

openssl req -new -out server.csr -key server.key

Otra vez comienzan las preguntas.

Nos pide el código de 2 letras del país. Supondremos que es España = SP.

Nos pide la provincia. Supondremos que es **Barcelona**.

Nos pide la localidad. Supondremos que también es **Barcelona**.

Nos pide el nombre de nuestra organización. Supondremos que es **RoJo**.

Nos pide el nombre de la unidad organizativa. Supondremos que es **IT**.

Nos pide el nombre del contacto o del servidor (ip o hostname). Esta pregunta es importante. Deberíamos responder con la ip o el nombre del servidor donde pretendemos instalar el certificado. Supondremos que es **172.16.0.250**

Nos pide el correo del contacto. Supondremos que es [**baltasar@reyesmagos.com**](mailto:melchor@reyesmagos.com)

Nos pide la **password challenge**. No pondremos ninguna. Directamente pulsamos enter. Si lo hacemos tendremos problemas más adelante.

Nos pide un nombre de empresa opcional. Tampoco responderemos. Pulsamos enter.

Y con esto finaliza la generación de la solicitud del certificado (**server.csr**).

Puesto que la entidad certificadora somos nosotros mismos, utilizaremos la solicitud junto con el certificado CA y claves CA para generar el certificado del servidor.

El certificado tendrá una validez un año (365 días).

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365

Nos pide la contraseña de las claves CA, que es **passCA**.

Ya tenemos el certificado para el servidor en **server.crt**

De todos los archivos creados sólo nos interesan **ca.crt**, **server.crt** y **server.key**.

El certificado de la CA lo copiaremos a la carpeta **/etc/mosquitto/ca\_certificates** que ya existe:

sudo cp ~/Downloads/ca.crt /etc/mosquitto/ca\_certificates

Los otros dos archivos los copiamos a la carpeta **/etc/mosquitto/certs** que también está creada.

sudo cp ~/Downloads/server.crt /etc/mosquitto/certs

sudo cp ~/Downloads/server.key /etc/mosquitto/certs

El resto de archivos no los borramos por ahora. Tendremos que generar un certificado para el cliente con la misma CA. Y para ello necesitaremos utilizar el **ca.key**.

Mantenemos los archivos generados en la carpeta **Downloads**.

Este proceso de generación de claves y certificados es el mismo para cualquier tipo de servidor (un servidor web).

Ahora que ya tenemos generados los archivos necesarios y en las carpetas correctas, tenemos que indicarlo en el archivo de configuración. Lo abrimos:

sudo gedit /etc/mosquitto/mosquitto.conf

Sobreescribimos su contenido con las siguientes líneas:

pid\_file /var/run/mosquitto.pid

persistence true

persistence\_location /var/lib/mosquitto/

log\_dest file /var/log/mosquitto/mosquitto.log

log\_type all

include\_dir /etc/mosquitto/conf.d

#password\_file /etc/mosquitto/pwfile

allow\_anonymous true

#acl\_file /etc/mosquitto/aclfile

cafile /etc/mosquitto/ca\_certificates/ca.crt

keyfile /etc/mosquitto/certs/server.key

certfile /etc/mosquitto/certs/server.crt

tls\_version tlsv1.2

port 8883

Guardamos, salimos y reiniciamos el servicio:

sudo systemctl restart mosquitto

Ya tenemos el servidor preparado para trabajar con seguridad.

Explicamos un poco la nueva configuración.

Estamos intentando configurar la seguridad (encriptación) de los mensajes.

Para simplificar el ejemplo, desactivaremos el archivo de usuarios:

#password\_file /etc/mosquitto/pwfile

Y el de reglas (ACL):

#acl\_file /etc/mosquitto/aclfile

Quedará preparado para aceptar conexiones anónimas. Indicamos de manera explícita que así es (aunque no sea necesario):

allow\_anonymous true

Incluimos la ubicación del certificado de la entidad certificadora:

cafile /etc/mosquitto/ca\_certificates/ca.crt

Y de las claves y certificado del servidor:

keyfile /etc/mosquitto/certs/server.key

certfile /etc/mosquitto/certs/server.crt

Indicamos explícitamente la versión de TLS que utilizaremos. En el ejemplo se utiliza la 1.2. La razón es porque los scripts de suscripción (**mosquitto\_sub**) y publicación (**mosquitto\_pub**) utilizan por defecto esta versión:

tls\_version tlsv1.2

Por último, indicamos que el puerto utilizado a partir de ahora será el 8883. Este es el puerto estándar utilizado para conexiones seguras:

port 8883

Para establecer conexión desde un cliente sólo necesitamos dos cosas:

- Tener el certificado de la entidad certificadora, que se supone que es público (ca.crt).

- Conocer el hostname o ip en la que reside el broker y para la que se expidió el certificado del servidor. En nuestro caso 172.16.0.250.

El envío de un mensaje por una conexión segura sería:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -t public --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -t public -m "Hola!" --cafile ~/Downloads/ca.crt

No incluimos ni el puerto, ni la versión de TLS, porque estamos utilizando los valores por defecto.

Pero si tuviésemos que escribirlo completo sería:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -p 8883 -t public --cafile ~/Downloads/ca.crt --tls-version tlsv1.2

**Publicación**: mosquitto\_pub -h 172.16.0.250 -p 8883 -t public -m "Hola!" --cafile ~/Downloads/ca.crt --tls-version tlsv1.2

Nota. Hay demasiados archivos y pasos en juego para que todo salga bien a la primera.

Una función bastante útil para verificar si un certificado está firmado correctamente por una entidad certificadora es la siguiente:

openssl verify -CAfile ~/Downloads/ca.crt ~/Downloads/server.crt

# Conexión segura con usuarios y reglas

En el ejemplo anterior hemos conseguido enviar y recibir mensajes con encriptación, gracias a los certificados del broker, pero habíamos desactivado los usuarios y su control de permisos (ACL).

Ahora los volveremos a activar sin miedo a que las contraseñas puedan ser interceptadas.

Modificamos la configuración del broker

sudo gedit /etc/mosquitto/mosquitto.conf

Sobreescribimos su contenido con las siguientes líneas:

pid\_file /var/run/mosquitto.pid

persistence true

persistence\_location /var/lib/mosquitto/

log\_dest file /var/log/mosquitto/mosquitto.log

log\_type all

include\_dir /etc/mosquitto/conf.d

password\_file /etc/mosquitto/pwfile

allow\_anonymous true

acl\_file /etc/mosquitto/aclfile

cafile /etc/mosquitto/ca\_certificates/ca.crt

keyfile /etc/mosquitto/certs/server.key

certfile /etc/mosquitto/certs/server.crt

tls\_version tlsv1.2

port 8883

Guardamos y salimos.

Sólo hemos reactivado el acceso al archivo de usuarios (con user1 y user3) y al archivo ACL.

Abrimos el archivo ACL:

sudo gedit /etc/mosquitto/aclfile

Sobre escribimos el contenido con las siguientes reglas:

pattern write correo/#

pattern read correo

pattern read correo/%u

Salvamos, salimos y reiniciamos el servicio:

sudo systemctl restart mosquitto

Repetimos algunas de las pruebas que en su momento hicimos con usuarios y reglas cuando no utilizábamos encriptación.

Comprobamos que un usuario puede enviar un mensaje a otro:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -u user1 -P Puser1 -t "correo/user1" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -u user3 -P Puser3 -t "correo/user1" -m "Hola!" --cafile ~/Downloads/ca.crt

Comprobamos que un usuario no puede leer los mensajes de otro:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -u user1 -P Puser1 -t "correo/user3" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -u user3 -P Puser3 -t "correo/user3" -m "Hola!" --cafile ~/Downloads/ca.crt

Comprobamos que un usuario puede enviar mensajes a todos:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -u user1 -P Puser1 -t "correo/#" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -u user3 -P Puser3 -t "correo" -m "Hola!" --cafile ~/Downloads/ca.crt

Comprobamos que un usuario anónimo puede enviar mensajes a uno registrado:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -u user1 -P Puser1 -t "correo/#" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -t "correo/user1" -m "Hola!" --cafile ~/Downloads/ca.crt

Comprobamos que un usuario anónimo puede enviar mensajes a todos:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -u user1 -P Puser1 -t "correo/#" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -t "correo" -m "Hola!" --cafile ~/Downloads/ca.crt

Un usuario anónimo puede recibir mensajes enviados a todos:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -t "correo/#" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -u user3 -P Puser3 -t "correo" -m "Hola!" --cafile ~/Downloads/ca.crt

Comprobamos que un usuario anónimo no puede recibir mensajes de un usuario registrado:

**Suscripción**: mosquitto\_sub -h 172.16.0.250 -t "correo/user3" --cafile ~/Downloads/ca.crt

**Publicación**: mosquitto\_pub -h 172.16.0.250 -u user3 -P Puser3 -t "correo/user3" -m "Hola!" --cafile ~/Downloads/ca.crt

# Certificados de cliente

Además de con usuario y contraseña, también podemos autenticarnos como clientes con nuestro certificado.

Habitualmente los certificados cliente los entrega el servidor para facilitar la gestión al cliente porque:

* El certificado del cliente debe estar firmado con la clave de la misma entidad certificadora que se utilizó con el servidor.
* Una vez generado, el servidor debe reconocer el certificado del cliente, como si fuese un usuario más.

Un certificado cliente puede ser compartido en varios clientes.

Se pueden utilizar simultáneamente ambos sistemas de autenticación: credenciales y certificados.

Los archivos de cliente (claves y certificado) los generaremos también en la carpeta de Downloads. Así que abriremos una terminal en esta carpeta.

El objetivo es generar un certificado para el usuario **user2** que actualmente no existe.

Comenzaremos generando un par de claves complementarias para el cliente en el archivo **client2.key**:

openssl genrsa -out client2.key 2048

A continuación creamos la solicitud de certificado (.csr) para pedirlo a la entidad certificadora en el archivo **client2.csr**:

openssl req -new -out client2.csr -key client2.key

Como es habitual, se nos hacen unas cuantas preguntas…

Nos pide el código de 2 letras del país. Supondremos que es España = **SP**.

Nos pide la provincia. Supondremos que es **Barcelona**.

Nos pide la localidad. Supondremos que también es **Barcelona**.

Nos pide el nombre de nuestra organización. Supondremos que es **RoJo**.

Nos pide el nombre de la unidad organizativa. Supondremos que es **IT**.

Nos pide el nombre del contacto o del servidor. Esta pregunta es importante. Debemos escribir el nombre del usuario que será reconocido en el servidor. En este caso es **user2**

Nos pide el correo del contacto. Supondremos que es [**user2@reyesmagos.com**](mailto:melchor@reyesmagos.com)

Nos pide la **password challenge**. No pondremos ninguna. Directamente pulsamos enter.

Nos pide un nombre de empresa opcional. Tampoco responderemos. Pulsamos enter.

Y con esto finaliza la generación de la solicitud del certificado (**client2.csr**).

Ahora haremos de entidad certificadora y con las claves de la entidad (**ca.key**), el certificado de la entidad (**ca.crt**) y la solicitud de certificado (**client2.csr**), generamos el certificado para el cliente (**client2.crt**). El certificado tendrá una validez de una año (365 días)

openssl x509 -req -in client2.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client2.crt -days 365

Nos pide la contraseña de las claves de la entidad certificadora, que era **passCA**

Ya tenemos los 3 archivos que necesita el cliente:

* **ca.crt** Certificado de la entidad certificadora
* **client2.crt** Certificado del cliente
* **client2.key** Clave privada del cliente

Aunque el cliente ya está preparado para trabajar con certificados, el servidor no.

Tendremos que hacer alguna modificación en su configuración.

Abrimos el archivo de configuración:

sudo gedit /etc/mosquitto/mosquitto.conf

Sobreescribimos su contenido con las siguientes líneas:

pid\_file /var/run/mosquitto.pid

persistence true

persistence\_location /var/lib/mosquitto/

log\_dest file /var/log/mosquitto/mosquitto.log

log\_type all

include\_dir /etc/mosquitto/conf.d

#password\_file /etc/mosquitto/pwfile

#allow\_anonymous true

acl\_file /etc/mosquitto/aclfile

cafile /etc/mosquitto/ca\_certificates/ca.crt

keyfile /etc/mosquitto/certs/server.key

certfile /etc/mosquitto/certs/server.crt

tls\_version tlsv1.2

port 8883

require\_certificate true

use\_identity\_as\_username true

Guardamos y salimos.

Hemos activado el parámetro **require\_certificate**. A partir de ahora es obligatorio que los cliente se conecten con certificado. No se admitirá usuario y contraseña. Esto invalida el uso del archivo de usuarios, así que comentamos el parámetro **password\_file** que indica dónde está.

Si no hacemos nada más, los usuarios están obligados a conectarse con certificados, pero no se les identifica individualmente. No se pueden aplicar reglas. Todos son anónimos.

Como nos interesa seguir identificando al usuario, activamos el parámetro **use\_identity\_as\_username** para que extraiga el nombre de usuario del certificado.

Puesto que todos los certificados tienen un nombre, ya no tienen sentido las conexiones anónimas y por eso comentamos el parámetro **allow\_anonymous**.

Mantenemos el archivo de reglas, que no cambiaremos. Simplemente comprobamos que es correcto. Abrimos el archivo ACL:

sudo gedit /etc/mosquitto/aclfile

Sobreescribimos el contenido con las siguientes reglas:

pattern write correo/#

pattern read correo

pattern read correo/%u

Salvamos, salimos y reiniciamos el servicio:

sudo systemctl restart mosquitto

Hacemos alguna prueba para comprobar que funciona.

Recordemos que sólo tenemos el usuario user2 con posibilidad de conectarse con certificado.

Comprobamos que user2 puede enviarse mensajes a él mismo:

**Suscripción**: mosquitto\_sub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/#"

**Publicación**: mosquitto\_pub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/user2" -m "Hola!"

Comprobamos que user2 puede enviar mensajes a todos los usuarios:

**Suscripción**: mosquitto\_sub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/#"

**Publicación**: mosquitto\_pub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo" -m "Hola!"

Comprobamos que user2 no puede leer mensajes enviados directamente a otros de otros usuarios:

**Suscripción**: mosquitto\_sub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/#"

**Publicación**: mosquitto\_pub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/user3" -m "Hola!"

Para poder hacer alguna prueba más necesitaríamos otro usuario con certificado.

Generaremos los archivos necesarios para el usuario user4:

openssl genrsa -out client4.key 2048

openssl req -new -out client4.csr -key client4.key

Respondemos a las preguntas...

Nos pide el código de 2 letras del país. Supondremos que es España = **SP**.

Nos pide la provincia. Supondremos que es **Barcelona**.

Nos pide la localidad. Supondremos que también es **Barcelona**.

Nos pide el nombre de nuestra organización. Supondremos que es **RoJo**.

Nos pide el nombre de la unidad organizativa. Supondremos que es **IT**.

Nos pide el nombre del contacto o del servidor. Esta pregunta es importante. Debemos escribir el nombre del usuario que será reconocido en el servidor. En este caso es **user4**

Nos pide el correo del contacto. Supondremos que es [**user4@reyesmagos.com**](mailto:melchor@reyesmagos.com)

Nos pide la **password challenge**. No pondremos ninguna. Directamente pulsamos enter.

Nos pide un nombre de empresa opcional. Tampoco responderemos. Pulsamos enter.

openssl x509 -req -in client4.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client4.crt -days 365

Nos pide la contraseña de las claves de la entidad certificadora, que era **passCA**

Ok. Ahora podemos hacer alguna prueba más.

Comprobamos que user2 puede enviar mensajes a user4:

**Suscripción**: mosquitto\_sub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client4.crt --key ~/Downloads/client4.key -h 172.16.0.250 -t "correo/#"

**Publicación**: mosquitto\_pub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/user4" -m "Hola!"

# Funcionalidades adicionales

A continuación se indican algunas funcionalidades adicionales que no se cubren en las explicaciones de este documento, pero que es aconsejable saber que existen:

* Un broker MQTT puede ser conectado a otros brokers para compartir ciertos tópicos.
* Se pueden seleccionar los puertos de escucha del broker.
* Un broker puede estar escuchando varios puertos simultáneamente para el mismo servicio.
* Se puede limitar el número de conexiones simultáneas.
* Los mensajes pueden ser retenidos.
* Los certificados utilizados por los clientes para conectarse pueden ser revocados. Se utiliza un archivo externo al que se referencia en la configuración.

Comprobamos que user2 puede enviar mensajes a user4:

**Suscripción**: mosquitto\_sub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client4.crt --key ~/Downloads/client4.key -h 172.16.0.250 -t "$SYS/#"

**Publicación**: mosquitto\_pub --cafile ~/Downloads/ca.crt --cert ~/Downloads/client2.crt --key ~/Downloads/client2.key -h 172.16.0.250 -t "correo/user4" -m "Hola!"