7.5

Mobile Messaging and M2M

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información en "Avisos" en la página 193.

Esta edición se aplica a la versión 7 release 5 de IBM® WebSphere MQ y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

Contenido

Introducción a MQTT

Información sobre el envío de mensajes entre aplicaciones móviles utilizando el transporte de telemetría MQ (MQTT). El protocolo está pensado para su uso en redes inalámbricas y de bajo ancho de banda. Una aplicación móvil que utiliza MQTT envía y recibe mensajes llamando a una biblioteca de MQTT. Los mensajes se intercambian a través de un servidor de mensajería de MQTT. El cliente y el servidor de MQTT manejan las complejidades de entregar mensajes de forma fiable para la aplicación móvil y mantener bajo el precio de la gestión de red.

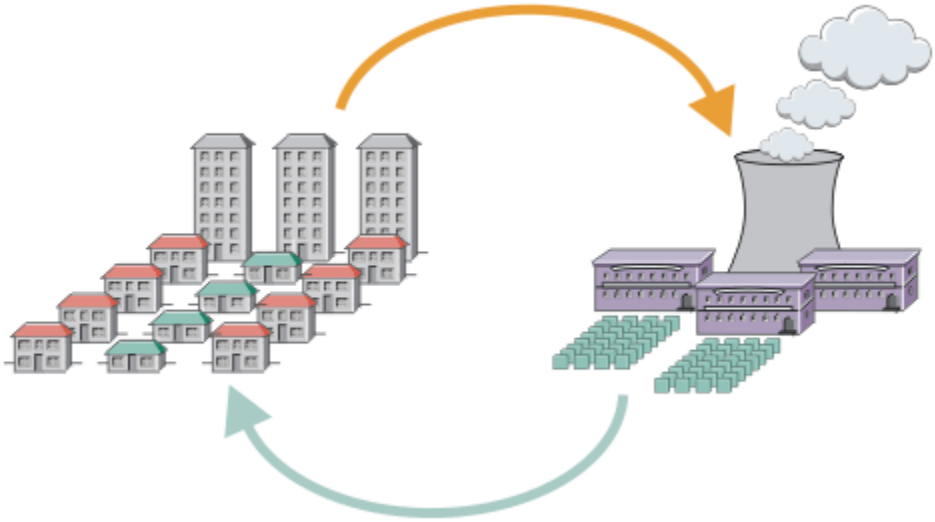
Las aplicaciones de MQTT se ejecutan en dispositivos móviles, tales como teléfonos inteligentes y tabletas. MQTT también se utiliza en telemetría para recibir datos de sensores y para controlarlos de forma remota. Para dispositivos móviles y sensores, MQTT ofrece un protocolo de publicación/suscripción altamente escalable con entrega asegurada. Para enviar y recibir mensajes MQTT, debe añadir una biblioteca de clientes de MQTT a su aplicación.

La biblioteca de clientes de MQTT es pequeña. La biblioteca actúa como buzón, enviando y recibiendo mensajes con otras aplicaciones MQTT conectadas a un servidor de MQTT. Al enviar los mensajes, en lugar de mantenerse conectadas a un servidor que esté esperando una respuesta, las aplicaciones MQTT conservan la vida de la batería. La biblioteca envía mensajes a otros dispositivos a través de un servidor de MQTT que ejecuta el protocolo MQTT version 3.1. Puede enviar mensajes a un cliente específico o utilizar mensajería de publicación/suscripción para conectar muchos dispositivos.

Las bibliotecas de cliente de MQTT conectan aplicaciones para sensores y dispositivos móviles a un servidor MQTT utilizando el protocolo MQTT.

IBM MessageSight y IBM WebSphere MQ son servidores MQTT. Pueden conectar grandes volúmenes de aplicaciones cliente MQTT, y pueden conectar redes MQTT y IBM WebSphere MQ entre sí. Consulte “Iniciación a los servidores MQTT” en la página 139. IBM WebSphere MQ y IBM MessageSight pueden formar un puente entre aplicaciones web externas que se estén ejecutando en sensores y dispositivos móviles y otros tipos de aplicaciones de publicación/suscripción y de mensajería que se estén ejecutando dentro de la empresa. El puente facilita la creación de "soluciones inteligentes" que incorporen sensores y dispositivos móviles.

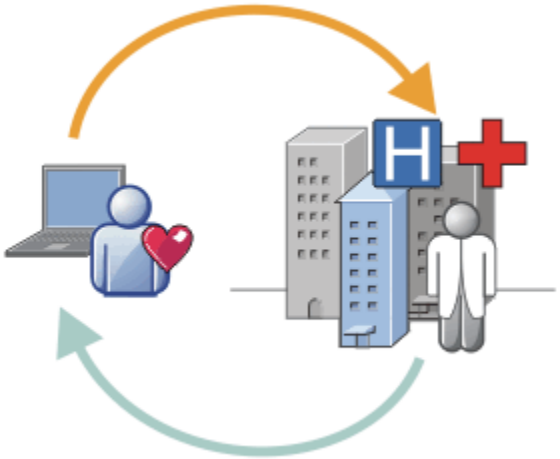
Las soluciones inteligentes ponen a disposición de las aplicaciones que se ejecutan en dispositivos móviles y sensores la abundante información que puede encontrarse en Internet. Dos ejemplos de aplicaciones inteligentes basadas en telemetría son la electricidad inteligente y los servicios de salud inteligentes.



- Se envía un mensaje MQTT que contiene datos de uso de energía al proveedor del servicio.
- Una aplicación de telemetría envía mandatos de control basados en el análisis de los datos de uso de energía.
- Para obtener más información, consulte Escenario de telemetría: supervisión y control de la energía doméstica.

Figura 1. Medición eléctrica inteligente

Figura 2. Supervisión de salud inteligente



- Una aplicación de telemetría envía sus datos de salud a su hospital y a su doctor.
- Se pueden enviar alertas de mensajes MQTT o comentarios en base al análisis de sus datos de salud.
- Para obtener más información, consulte Escenario de telemetría: supervisión de pacientes domiciliarios.

Puede compilar MQTT en dispositivos pequeños escribiendo su propia aplicación para la MQTT protocol. Para ayudarle, IBM proporciona bibliotecas de cliente que admiten aplicaciones que se ejecuten sobre MQTT. Consulte “Iniciación a los clientes MQTT” en la página 11. IBM proporciona bibliotecas de clientes para aplicaciones iOS, y para aplicaciones Android V 7.5.0.1, y un cliente navegador JavaScript para aplicaciones web sin plataforma. V 7.5.0.1 Las páginas de cliente de JavaScript le conectan a IBM MessageSight y IBM WebSphere MQ con el protocolo MQTT sobre WebSockets. IBM también proporciona aplicaciones de ejemplo MQTT para C y Java en Linux® y Windows.

Las bibliotecas de C y Java funcionan sobre iOS, Android, Windows, y diversas plataformas UNIX and Linux. Puede portar el código fuente C de la biblioteca de cliente MQTT a otras plataformas. Las bibliotecas de cliente MQTT para C y Java están disponibles con una licencia de código abierto del proyecto Eclipse Paho. Consulte Eclipse Paho. La especificación MQTT protocol es abierta y disponible en MQTT.org.

MQTT protocol

MQTT protocol es ligera en el sentido de que los clientes son pequeños, y que utiliza el ancho de banda de red de forma eficiente. El protocolo MQTT da soporte a la entrega asegurada y a transferencias 'dispara y olvida'. En el protocolo, la entrega de mensajes es independiente (desacoplada) de la aplicación. El grado de desacoplamiento de una aplicación depende de la forma en que están escritos el cliente MQTT y el servidorMQTT. La entrega desacoplada libera a una aplicación de tener que estar conectada a un servidor y esperando mensajes. El modelo de interacción es como en el correo electrónico, pero optimizado para la programación de aplicaciones.

El protocolo MQTT V3.1 está publicado; consulte MQTT V3.1 Protocol Specification. En la especificación se identifican diversos rasgos distintivos del protocolo:

- Es un protocolo de publicación/suscripción.

  Además de proporcionar distribución 'de uno a muchos', la publicación/suscripción desacopla las aplicaciones. Ambas funciones resultan útiles en aplicaciones que tengan muchos clientes.
- No depende en modo alguno del contenido del mensaje.
- Se ejecuta sobre TCP/IP, que proporciona conectividad de red básica.
- Tiene tres calidades de servicio para la entrega de mensajes:

  "Como máximo una vez"
  Los mensajes de entregan en base a los mejores esfuerzos de la red de Protocolo Internet subyacente. Se puede producir pérdida de mensajes.

  Utilice esta calidad de servicio con la comunicación de datos de sensores ambientales, por ejemplo. No importa si una lectura individual se pierde, si la siguiente se publica poco después.

  "Al menos una vez"
  Se asegura que los mensajes llegan, pero se pueden producir duplicados.

  "Exactamente una vez"
  Se asegura que los mensajes llegan exactamente una sola vez.

  Utilice esta calidad de servicio con sistemas de facturación, por ejemplo. Los mensajes duplicados o perdidos pueden provocar un problema o generar cargos incorrectos.
- Es económico en la forma en que gestiona el flujo de mensajes en la red. Por ejemplo, la cabecera de longitud fija tiene sólo 2 bytes de longitud y se minimizan los intercambios de protocolo para reducir el tráfico en la red.
- Dispone de una función "Última voluntad y testamento" que notifica a los suscriptores si se produce una desconexión de un cliente de un servidor MQTT. Consulte “Publicación de última voluntad y testamento” en la página 132

MQTT version 3.1 está soportado en IBM IBM WebSphere MQ y en IBM MessageSight. MQTT está implementado sobre TCP/IP. Hay disponible otra versión del protocolo, MQTT-S, para redes que no son TCP/IP. Consulte Especificación MQTT-S version 1.2.

Comunidades MQTT

IBM mantiene Comunidad deIBM Developer Mensajería para MQTT desarrolladores que escriben aplicaciones para IBM MessageSight y IBM WebSphere MQ.

MQTT.org es un buen lugar para obtener información y debatir sobre implementaciones y extensiones del protocolo MQTT.

MQTT es un proyecto de código abierto Eclipse, bajo el Eclipse Technology Project. La comunidad Paho desarrolla clientes y servidores de código abierto. Consulte Eclipse Paho.

Introducción a MQTT

Información sobre el envío de mensajes entre aplicaciones móviles utilizando el transporte de telemetría MQ (MQTT). El protocolo está pensado para su uso en redes inalámbricas y de bajo ancho de banda. Una aplicación móvil que utiliza MQTT envía y recibe mensajes llamando a una biblioteca de MQTT. Los mensajes se intercambian a través de un servidor de mensajería de MQTT. El cliente y el servidor de MQTT manejan las complejidades de entregar mensajes de forma fiable para la aplicación móvil y mantener bajo el precio de la gestión de red.

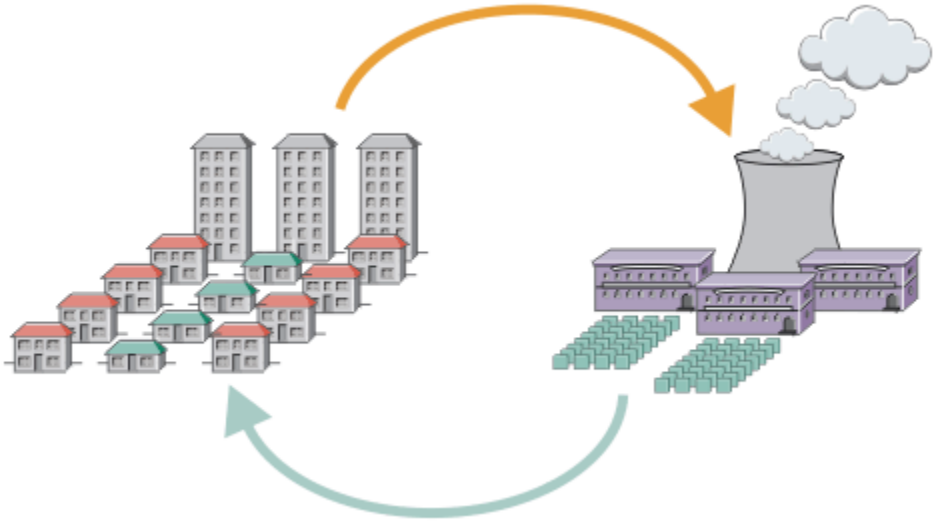
Las aplicaciones de MQTT se ejecutan en dispositivos móviles, tales como teléfonos inteligentes y tabletas. MQTT también se utiliza en telemetría para recibir datos de sensores y para controlarlos de forma remota. Para dispositivos móviles y sensores, MQTT ofrece un protocolo de publicación/suscripción altamente escalable con entrega asegurada. Para enviar y recibir mensajes MQTT, debe añadir una biblioteca de clientes de MQTT a su aplicación.

La biblioteca de clientes de MQTT es pequeña. La biblioteca actúa como buzón, enviando y recibiendo mensajes con otras aplicaciones MQTT conectadas a un servidor de MQTT. Al enviar los mensajes, en lugar de mantenerse conectadas a un servidor que esté esperando una respuesta, las aplicaciones MQTT conservan la vida de la batería. La biblioteca envía mensajes a otros dispositivos a través de un servidor de MQTT que ejecuta el protocolo MQTT version 3.1. Puede enviar mensajes a un cliente específico o utilizar mensajería de publicación/suscripción para conectar muchos dispositivos.

Las bibliotecas de cliente de MQTT conectan aplicaciones para sensores y dispositivos móviles a un servidor MQTT utilizando el protocolo MQTT.

IBM MessageSight y IBM WebSphere MQ son servidores MQTT. Pueden conectar grandes volúmenes de aplicaciones cliente MQTT, y pueden conectar redes MQTT y IBM WebSphere MQ entre sí. Consulte “Iniciación a los servidores MQTT” en la página 139. IBM WebSphere MQ y IBM MessageSight pueden formar un puente entre aplicaciones web externas que se estén ejecutando en sensores y dispositivos móviles y otros tipos de aplicaciones de publicación/suscripción y de mensajería que se estén ejecutando dentro de la empresa. El puente facilita la creación de "soluciones inteligentes" que incorporen sensores y dispositivos móviles.

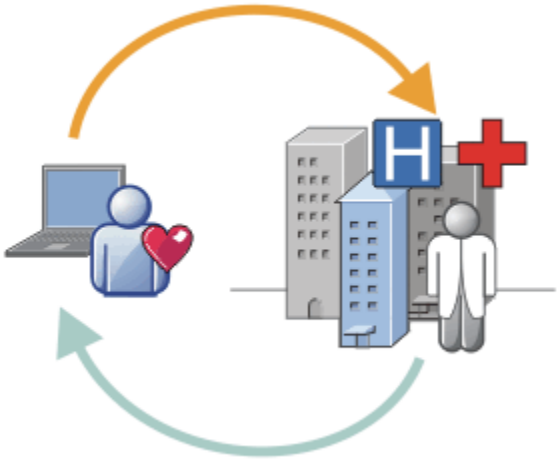
Las soluciones inteligentes ponen a disposición de las aplicaciones que se ejecutan en dispositivos móviles y sensores la abundante información que puede encontrarse en Internet. Dos ejemplos de aplicaciones inteligentes basadas en telemetría son la electricidad inteligente y los servicios de salud inteligentes.



- Se envía un mensaje MQTT que contiene datos de uso de energía al proveedor del servicio.
- Una aplicación de telemetría envía mandatos de control basados en el análisis de los datos de uso de energía.
- Para obtener más información, consulte Escenario de telemetría: supervisión y control de la energía doméstica.

Figura 3. Medición eléctrica inteligente

Figura 4. Supervisión de salud inteligente



- Una aplicación de telemetría envía sus datos de salud a su hospital y a su doctor.
- Se pueden enviar alertas de mensajes MQTT o comentarios en base al análisis de sus datos de salud.
- Para obtener más información, consulte Escenario de telemetría: supervisión de pacientes domiciliarios.

Puede compilar MQTT en dispositivos pequeños escribiendo su propia aplicación para la MQTT protocol. Para ayudarle, IBM proporciona bibliotecas de cliente que admiten aplicaciones que se ejecuten sobre MQTT. Consulte “Iniciación a los clientes MQTT” en la página 11. IBM proporciona bibliotecas de clientes para aplicaciones iOS, y para aplicaciones Android V 7.5.0.1, y un cliente navegador JavaScript para aplicaciones web sin plataforma. V 7.5.0.1 Las páginas de cliente de JavaScript le conectan a IBM MessageSight y IBM WebSphere MQ con el protocolo MQTT sobre WebSockets. IBM también proporciona aplicaciones de ejemplo MQTT para C y Java en Linux y Windows.

Las bibliotecas de C y Java funcionan sobre iOS, Android, Windows, y diversas plataformas UNIX and Linux. Puede portar el código fuente C de la biblioteca de cliente MQTT a otras plataformas. Las bibliotecas de cliente MQTT para C y Java están disponibles con una licencia de código abierto del proyecto Eclipse Paho. Consulte Eclipse Paho. La especificación MQTT protocol es abierta y disponible en MQTT.org.

MQTT protocol

MQTT protocol es ligera en el sentido de que los clientes son pequeños, y que utiliza el ancho de banda de red de forma eficiente. El protocolo MQTT da soporte a la entrega asegurada y a transferencias 'dispara y olvida'. En el protocolo, la entrega de mensajes es independiente (desacoplada) de la aplicación. El grado de desacoplamiento de una aplicación depende de la forma en que están escritos el cliente MQTT y el servidorMQTT. La entrega desacoplada libera a una aplicación de tener que estar conectada a un servidor y esperando mensajes. El modelo de interacción es como en el correo electrónico, pero optimizado para la programación de aplicaciones.

El protocolo MQTT V3.1 está publicado; consulte MQTT V3.1 Protocol Specification. En la especificación se identifican diversos rasgos distintivos del protocolo:

- Es un protocolo de publicación/suscripción.

  Además de proporcionar distribución 'de uno a muchos', la publicación/suscripción desacopla las aplicaciones. Ambas funciones resultan útiles en aplicaciones que tengan muchos clientes.
- No depende en modo alguno del contenido del mensaje.
- Se ejecuta sobre TCP/IP, que proporciona conectividad de red básica.
- Tiene tres calidades de servicio para la entrega de mensajes:

  "Como máximo una vez"
  Los mensajes de entregan en base a los mejores esfuerzos de la red de Protocolo Internet subyacente. Se puede producir pérdida de mensajes.

  Utilice esta calidad de servicio con la comunicación de datos de sensores ambientales, por ejemplo. No importa si una lectura individual se pierde, si la siguiente se publica poco después.

  "Al menos una vez"
  Se asegura que los mensajes llegan, pero se pueden producir duplicados.

  "Exactamente una vez"
  Se asegura que los mensajes llegan exactamente una sola vez.

  Utilice esta calidad de servicio con sistemas de facturación, por ejemplo. Los mensajes duplicados o perdidos pueden provocar un problema o generar cargos incorrectos.
- Es económico en la forma en que gestiona el flujo de mensajes en la red. Por ejemplo, la cabecera de longitud fija tiene sólo 2 bytes de longitud y se minimizan los intercambios de protocolo para reducir el tráfico en la red.
- Dispone de una función "Última voluntad y testamento" que notifica a los suscriptores si se produce una desconexión de un cliente de un servidor MQTT. Consulte “Publicación de última voluntad y testamento” en la página 132

MQTT version 3.1 está soportado en IBM IBM WebSphere MQ y en IBM MessageSight. MQTT está implementado sobre TCP/IP. Hay disponible otra versión del protocolo, MQTT-S, para redes que no son TCP/IP. Consulte Especificación MQTT-S version 1.2.

Comunidades MQTT

IBM mantiene Comunidad deIBM Developer Mensajería para MQTT desarrolladores que escriben aplicaciones para IBM MessageSight y IBM WebSphere MQ.

MQTT.org es un buen lugar para obtener información y debatir sobre implementaciones y extensiones del protocolo MQTT.

MQTT es un proyecto de código abierto Eclipse, bajo el Eclipse Technology Project. La comunidad Paho desarrolla clientes y servidores de código abierto. Consulte Eclipse Paho.

Iniciación a los clientes MQTT

Puede empezar a desarrollar una aplicación móvil o de máquina a máquina (M2M - machine-to-machine) compilando y ejecutando una aplicación cliente MQTT de ejemplo que utilice una biblioteca de cliente MQTT. Las aplicaciones de ejemplo y las bibliotecas de cliente asociadas están disponibles en el Mobile Messaging and M2M Paquete de clientes de IBM. Hay versiones de las aplicaciones y bibliotecas de cliente escritas en Java, en JavaScripty en C. Puede ejecutar estas aplicaciones en la mayoría de plataformas y dispositivos, incluidos los dispositivos y productos de Android desde Apple.

Antes de empezar

Para compilar y ejecutar su propia aplicación, necesita adquirir de experiencia sobre cómo compilar aplicaciones para el dispositivo o plataforma de destino y el lenguaje de programación a utilizar. Con un poco de experiencia suele ser necesario para conseguir que una aplicación de ejemplo funcione en el dispositivo o plataforma de su elección.

Si utiliza un servidor MQTT con capacidad empresarial como IBM WebSphere MQ o IBM MessageSight, podrá intercambiar información desde la aplicación de ejemplo con las aplicaciones existentes en la empresa.

Acerca de esta tarea

Sus objetivos son los siguientes:

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.
2. Descargue el Mobile Messaging and M2M Paquete de clientes.
3. Compila las aplicaciones de ejemplo desde el paquete de cliente, para el dispositivo o plataforma de destino que haya elegido.
4. Compruebe que los ejemplos funcionan bien conectándolos al servidor MQTT.

Como resultado de compilar y probar las aplicaciones de ejemplo para su dispositivo o plataforma, creará un entorno de desarrollo de trabajo que podrá utilizar después para compilar sus propias aplicaciones cliente.

El Mobile Messaging and M2M Paquete de clientes contiene el SDK de MQTT. Este SDK le proporciona los recursos siguientes:

- Aplicaciones cliente MQTT de ejemplo escritas en Java, en JavaScript y en C.
- Bibliotecas de cliente MQTT que admiten estas aplicaciones cliente y les permiten ejecutarse en la mayoría de plataformas y dispositivos.

El SDK también incluye el código fuente del Cliente MQTT para C. Puede adaptar este código fuente para compilar las bibliotecas de cliente MQTT para C para otras plataformas. Para obtener ayuda para hacer esto, puede consultar “Creación de las bibliotecas de cliente MQTT para C” en la página 31. El código fuente del Cliente MQTT para C está también disponible con una licencia de código abierto en Eclipse Paho.

Procedimiento

En los artículos siguientes encontrará una guía con los pasos específicos para cada plataforma para compilar y ejecutar una aplicación MQTT de ejemplo en un ordenador de sobremesa o en un dispositivo móvil Android o Apple:

- “Iniciación al cliente MQTT para Java” en la página 12
- “Iniciación a Cliente MQTT para Java en Android” en la página 18
- V 7.5.0.1

"Iniciación a Cliente MQTT de mensajería para JavaScript" en la página 24
- "Iniciación al cliente MQTT para C" en la página 26
- "Iniciación al cliente MQTT para C en iOS" en la página 48

Qué hacer a continuación

Para desarrollar una nueva aplicación MQTT, debe tener o adquirir los conocimientos siguientes:

- Programación en el lenguaje necesario para el dispositivo o plataforma.
- Programación para el dispositivo o plataforma de destino.
- Diseño de aplicaciones de publicación/suscripción.
- Diseño de programas para el modelo de programación MQTT.
- Diseño de programas para ejecutar en el dispositivo móvil elegido.
- Uso de SSL y JAAS para proteger programas.

No es necesario tener conocimientos de programación de redes para conectar un cliente de MQTT a otro dispositivo o aplicación, porque MQTT es un sistema de mensajería y gestión de colas. Las bibliotecas de cliente MQTT gestionan las conexiones de red de su aplicación.

Para integrar el cliente MQTT con las aplicaciones de empresa existentes, tiene dos opciones. Puede compartir los temas de publicación/suscripción de MQTT con (por ejemplo) una aplicación de IBM WebSphere MQ o JMS, o puede escribir su propio adaptador de integración como otro cliente MQTT.

Las fuentes de información a consultar hoy en día son:

- Desarrollo de aplicaciones para WebSphere MQ Telemetry
- MQTT.org
- Eclipse Paho

Conceptos relacionados

"Iniciación a los servidores MQTT" en la página 139

Iniciación al cliente MQTT para Java

Empezar a trabajar con el cliente MQTT para aplicaciones de ejemplo de Java, utilizando IBM MessageSight o IBM WebSphere MQ como servidor de MQTT. Las aplicaciones de ejemplo utilizan una biblioteca de cliente del kit de desarrollo de software (SDK) de MQTT de IBM. La aplicación de ejemplo `SampleAsyncCallBack` es un modelo para escribir aplicaciones MQTT para Android y otros sistemas operativos controlados por sucesos.

Antes de empezar

- Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java". Consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.
- Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .

Acerca de esta tarea

El propósito de esta tarea es comprobar que puede generar y ejecutar un cliente MQTT para una aplicación de ejemplo Java, conectarlo a IBM WebSphere MQ o a IBM MessageSight como servidor MQTT version 3 e intercambiar mensajes.

Siga esta tarea para ejecutar la aplicación de ejemplo desde el entorno de trabajo de Eclipse o desde una línea de mandatos. Los pasos que se detallan en el ejemplo son para Windows. Con ligeras modificaciones, puede ejecutar la aplicación de ejemplo en cualquier plataforma que admita JSE 1.5 o superior.

Puede ejecutar aplicaciones en el mismo servidor que IBM WebSphere MQ, donde ya tiene configurado el entorno para ejecutar aplicaciones que conectan a IBM WebSphere MQ. Siga la tarea "Configuración del servicio de MQTT desde la línea de mandatos" en la página 143 para instalar y configurar IBM WebSphere MQ con la opción IBM WebSphere MQ Telemetry en Windows o Linux. Una vez el entorno esté instalado y configurado, ejecute la aplicación de ejemplo, `MQTTV3Sample`, para verificar la instalación.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 . Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte "Iniciación a los servidores MQTT" en la página 139.
2. Opcional: Configure el servidor MQTT.
   - En IBM WebSphere MQ, debe completar una u otra de las tareas siguientes para definir un gestor de colas y configurar el servicio de telemetría (MQXR):
     - "Configuración del servicio de MQTT desde la línea de mandatos" en la página 143
     - "Configuración del servicio de MQTT con IBM WebSphere MQ Explorer" en la página 146
   - En otros servidores, consulte la documentación del servidor. Para Really Small Message Broker no hay que hacer ninguna configuración. Consulte Really Small Message Broker.
3. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.

   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre `MQTT`. La vía de acceso a esta carpeta se denomina aquí *sdkroot*.

   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *sdkroot*. La expansión crea un árbol de directorios que empieza en *sdkroot*\SDK.
4. Instale un kit de desarrollo de Java (JDK) Versión 6 o posterior.

   Debido a que está desarrollando una Java app for Android, JDK debe proceder de Oracle. Puede obtener el JDK de Descargas de Java SE.
5. Compile y ejecute uno o varios de los clientes de MQTT para las aplicaciones de ejemplo Java:
   - "Compilar y ejecutar los programas de ejemplo de Paho desde la línea de mandatos" en la página 14
   - "Compilar y ejecutar todas las aplicaciones Java de cliente MQTT desde Eclipse" en la página 16
   - "Iniciación a Cliente MQTT para Java en Android" en la página 18

   En el SDK se incluyen las siguientes aplicaciones Java de cliente MQTT de ejemplo:

   `MQTTV3Sample`
   Este ejemplo también se incluye en IBM WebSphere MQ y enlaza al paquete `com.ibm.micro.client.mqttv3.jar`.

   `Sample`
   `Sample` está en el paquete Paho y enlaza con el paquete `org.eclipse.paho.client.mqttv3` . Es parecido a `MQTTV3Sample`; espera hasta que todas las acciones MQTT se hayan completado.

   `SampleAsyncWait`
   `SampleAsyncWait` está en el paquete `org.eclipse.paho.client.mqttv3`. Utiliza la API MQTT asíncrona; espera en una hebra distinta hasta que se completa una acción. La hebra principal puede hacer otras cosas hasta sincronizarse con la hebra que está esperando que se complete la acción MQTT.

SampleAsyncCallBack
SampleAsyncCallBack está en el paquete `org.eclipse.paho.client.mqttv3`. Llama a la API MQTT asíncrona. La API asíncrona no espera a que MQTT complete el proceso de una llamada; vuelve a la aplicación. La aplicación continúa con otras tareas y espera a que llegue el siguiente suceso para procesarlo. MQTT publica una notificación de sucesos en la aplicación cuando completa el procesamiento. La interfaz MQTT basada en sucesos es adecuada para el modelo de servicios y actividades de programación de Android y de otros sistema operativos basados en sucesos.

A modo de ejemplo, observe cómo el ejemplo `mqttExerciser` integra MQTT en Android utilizando el modelo de servicios y actividades de programación.

mqttExerciser
`mqttExerciser` es un programa de ejemplo para Android. Como se compila y se ejecuta de forma distinta, se describe por separado. Consulte "Iniciación a Cliente MQTT para Java en Android" en la página 18.

Resultados

Ha compilado y ejecutado las aplicaciones MQTT Java de ejemplo que están conectadas a IBM WebSphere MQ o a IBM MessageSight, como el servidor MQTT.

Qué hacer a continuación

Consulte la información de referencia de Javadoc; consulte el paso "3" en la página 17 de "Compilar y ejecutar todas las aplicaciones Java de cliente MQTT desde Eclipse" en la página 16. Si no, abra los archivos html de Javadoc html que se encuentran en el directorio `SDK\clients\java\doc\javadoc` de Mobile Messaging and M2M Paquete de clientes.

Compilar y ejecutar los programas de ejemplo de Paho desde la línea de mandatos

Compile y ejecute la Paho aplicación de ejemplo `Sample.java` desde la línea de mandatos. El ejemplo se encuentra en el SDK de MQTT. El ejemplo se crea con las bibliotecas de cliente de MQTT Paho en el paquete `org.eclipse.paho.client.mqttv3` . Ejemplifica un publicador y un suscriptor MQTT. Se pueden compilar y ejecutar del mismo modo otros dos ejemplos de Paho en el mismo directorio. Difieren en que llaman a la biblioteca MQTT de forma asíncrona.

Antes de empezar

Siga los pasos del "1" en la página 13 al "4" en la página 13 de la tarea principal para configurar un servidor MQTT y descargue Mobile Messaging and M2M Paquete de clientes.

Acerca de esta tarea

Compile y ejecute `Sample.java` desde el subdirectorio de ejemplos de cliente de SDK, `SDK\clients\java\samples`. El código Java se encuentra en el directorio, `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`.

Procedimiento

1. Cree un script en el directorio de ejemplos de cliente para compilar y ejecutar `Sample` en la plataforma elegida.

   El siguiente script compila y ejecuta el ejemplo en Windows.

```
@echo on
setlocal
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b localhost -p 1883
pause
endlocal
```

Figura 5. Compilar y ejecutar `Sample.java`

2. Ejecute el script .

   Resultados:

```
Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
Subscribing to topic "Sample/#" qos 2
Press <Enter> to exit
Time:   2012-11-09 17:23:22.718  Topic: Sample/Java/v3  Message:        Message
from blocking MQTTv3 Java client sample  QoS:   2
```

Figura 6. Suscriptor a `MQTTV3Sample`

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 17:22:07.734 to topic "Sample/Java/v3" qos 2
Disconnected
```

Figura 7. Publicador de `MQTTV3Sample`

   Si deja la aplicación de suscriptor en ejecución, no puede volver a ejecutar el mismo suscriptor. El identificador de cliente del nuevo suscriptor es el mismo que el del anterior. No se puede ejecutar dos clientes MQTT con el mismo identificador de cliente a la vez. Puede utilizar la opción `-i` para definir el identificador de cliente, así podrá ejecutar distintos suscriptores a la vez.

   Si vuelve a ejecutar el mismo cliente, puede utilizar la opción `-c` para iniciarlo, e iniciar el cliente con `cleansession` definido en false. Con esta opción puede observar el comportamiento de las sesiones de cliente interrumpidas.

3. Finalice el suscriptor pulsando intro o cerrando la ventana.

Qué hacer a continuación

Cree scripts para compilar y ejecutar los demás ejemplos del subdirectorio `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`. Copie el script en Figura 5 en la página 15 y sustituya `Sample` por `SampleAsyncWait` o `SampleAsyncCallBack`. Los otros ejemplos son versiones asíncronas del programa síncrono `Sample`.

`SampleAsyncWait`
`SampleAsyncWait` está en el paquete `org.eclipse.paho.client.mqttv3`. Utiliza la API MQTT asíncrona; espera en una hebra distinta hasta que se completa una acción. La hebra principal puede hacer otras cosas hasta sincronizarse con la hebra que está esperando que se complete la acción MQTT.

`SampleAsyncCallBack`
`SampleAsyncCallBack` está en el paquete `org.eclipse.paho.client.mqttv3`. Llama a la API MQTT asíncrona. La API asíncrona no espera a que MQTT complete el proceso de una llamada; vuelve a la aplicación. La aplicación continúa con otras tareas y espera a que llegue el siguiente suceso para procesarlo. MQTT publica una notificación de sucesos en la aplicación cuando completa el procesamiento. La interfaz MQTT basada en sucesos es adecuada para el modelo de servicios y actividades de programación de Android y de otros sistema operativos basados en sucesos.

A modo de ejemplo, observe cómo el ejemplo `mqttExerciser` integra MQTT en Android utilizando el modelo de servicios y actividades de programación.

Los ejemplos asíncronos ejemplifican cómo reducir la cantidad de tiempo que se bloquea una aplicación MQTT mientras espera el cliente MQTT. Es importante eliminar las llamadas de bloqueo de la hebra principal para aumentar la capacidad de respuesta y la vida de la batería en un entorno móvil.

Con los ejemplos se ejemplifican dos patrones para llamar a interfaces asíncronas en el cliente MQTT.

1. `SampleAsyncWait` no se bloquea mientras el MQTT está a la espera de interacciones de red.
2. `SampleAsyncCallBack` no se bloquea mientras espera que el cliente MQTT complete alguna acción o acciones. Esto último es necesario cuando una página JavaScript llama al cliente MQTT desde un navegador. La páginas JavaScript no se deben bloquear. Las respuestas a las acciones se deben publicar en la hebra principal del navegador, que llama al manejador de eventos MQTT que haya escrito para procesar la notificación.

Compilar y ejecutar todas las aplicaciones Java de cliente MQTT desde Eclipse

Compile y ejecute las aplicaciones Java de ejemplo de cliente de MQTT que se encuentran en Mobile Messaging and M2M Paquete de clientes. Ejemplifican un publicador y un suscriptor MQTT.

Acerca de esta tarea

Compile y ejecute los ejemplos de MQTT Java , `Sampley` `MQTTV3Sample` en Eclipse. `Sample` se encuentra en el subdirectorio de clientes de SDK *`sdkroot`*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` y `MQTTV3Sample.java` se encuentra en *`sdkroot`*`\SDK\clients\java\samples`.

Procedimiento

1. Descargue Eclipse IDE for Java Developers.
2. Cree un proyecto Java que se llame `MQTT Samples` en Eclipse.
   a) **Archivo** > **Nuevo** > **proyecto Java** y escriba `MQTT Samples`. Pulse **Siguiente**.

      Compruebe que el JRE está en la versión correcta o posterior.JSE debe estar en la versión 1.5 o posterior.

   b) En la ventana **Ajustes de Java**, pulse **Enlazar carpetas de origen adicionales**.

   c) Vaya al directorio donde haya instalado la carpeta SDK de Java de MQTT. Seleccione la carpeta *`sdkroot`*`\SDK\clients\java\samples` y pulse **Aceptar** > **Siguiente** > **Finalizar**.

   d) En la ventana **Ajustes de Java**, pulse **Bibliotecas** > **Añadir JARS externos**

   e) Vaya al directorio donde haya instalado la carpeta SDK de Java de MQTT. Localice la carpeta *`sdkroot`*`\SDK\clients\java` y seleccione los archivos `org.eclipse.paho.client.mqttv3.jar` y `com.ibm.micro.client.mqttv3.jar` ; pulse **Abrir** > **Finalizar**.

      El ejemplo `MQTTV3Sample.java` enlaza a `com.ibm.micro.client.mqttv3.jar` y los ejemplos del árbol de directorios de paho enlazan a `org.eclipse.paho.client.mqttv3.jar`. El `com.ibm.micro.client.mqttv3.jar` se conserva para que las aplicaciones MQTT existentes continúen compilándose y ejecutándose sin cambios.

   El proyecto `MQTT Samples` genera algunos avisos al compilarse pero ningún error.

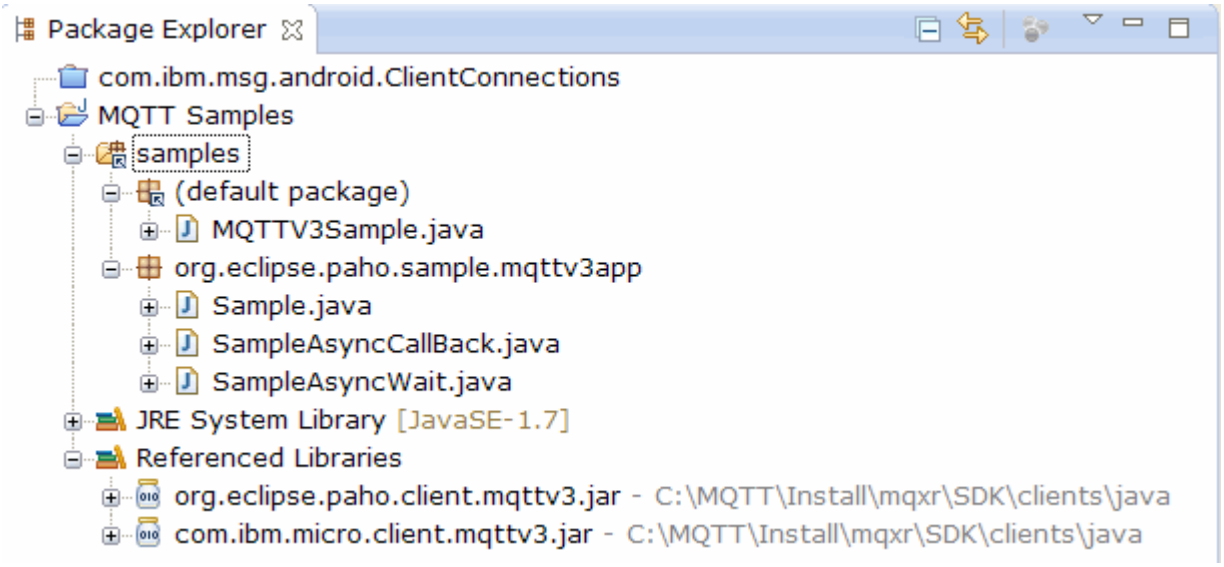


Figura 8. Proyecto cliente MQTT para Java

3. Opcional: Instale el Javadoc del cliente MQTT.

   Con el cliente de MQTT Javadoc instalado, el editor Java describe las clases MQTT en la ayuda contextual.

   a) Abra **Explorador de paquetes** > **Bibliotecas referenciadas** en el proyecto Java. Pulse con el botón derecho `org.eclipse.paho.client.mqttv3.jar` **> Propiedades**.

   b) En el navegador Propiedades pulse **Ubicación de Javadoc**.

   c) Pulse **URL de Javadoc** > **Examinar** en la página **Ubicación de Javadoc** y busque la carpeta `SDK\clients\java\doc\javadoc` **> Aceptar**.

   d) Pulse **Validar** > **Aceptar**

      Se le pedirá que abra un navegador para ver la documentación.

   e) Repita el procedimiento para el archivo `com.ibm.micro.client.mqttv3.jar`.

4. Cree una configuración de publicador y suscriptor para ejecutar la aplicación de ejemplo `mqttv3app.Sample`.

   a) Pulse con el botón derecho del ratón en la clase **Ejemplo** , pulse **Ejecutar como** > **Ejecutar configuraciones**.

   b) Pulse con el botón derecho del ratón en **Aplicación Java** > **Nueva** y escriba el nombre `SampleSubscriber`.

   c) Pulse en la pestaña de argumentos, escriba los argumentos de programa y después pulse **Aplicar**.

   ```
   -a subscribe -b localhost -p 1883
   ```

   d) Repita el último paso para crear una configuración para `SamplePublisher` omitiendo el parámetro `-a subscribe`.

5. Ejecute el suscriptor `mqttv3app.Sample` y después el publicador.

   a) Pulse **Ejecutar** > **configuraciones de ejecución**

   b) Pulse **SampleSubscriber** > **Ejecutar**.

      Abra la vista **Consola**. El suscriptor está esperando una publicación.

   ```
   Connected to tcp://localhost:1883 with client ID SampleJavaV3_subscribe
   Subscribing to topic "Sample/#" qos 2
   Press <Enter> to exit
   ```

   c) Pulse **SamplePublisher** > **Ejecutar**.

Abra la vista **Consola**. Verá la publicación que ha creado el publicador:

```
Connecting to tcp://localhost:1883 with client ID SampleJavaV3_publish
Connected
Publishing at: 2012-11-09 14:09:29.859 to topic "Sample/Java/v3" qos 2
Disconnected
```

d) Cambie a la vista de consola del suscriptor.

El icono para cambiar consolas es  .

El suscriptor ha recibido la publicación:

```
...
Time: 2012-11-09 14:09:30.593  Topic: Sample/Java/v3  Message: Message from blocking
MQTTv3 Java client sample  QoS: 2
```

6. Opcional: Cree una configuración de ejecución de publicador y suscriptor para `MQTTV3Sample`.
   a) Pulse con el botón derecho del ratón en la clase **MQTTV3Sample** y pulse **Ejecutar como** > **Ejecutar configuraciones**.
   b) Pulse con el botón derecho del ratón en **Aplicación Java** > **Nueva** y escriba el nombre `MQTTV3SampleSubscriber`.
   c) Pulse en la pestaña de argumentos, escriba los argumentos de programa y después pulse **Aplicar**.

```
-a subscribe -b localhost -p 1883
```

   d) Repita el último paso para crear una configuración para `MQTTV3SamplePublisher` omitiendo el parámetro `-a subscribe`.
7. Opcional: Ejecute el suscriptor `MQTTV3Sample` y después el publicador.
   a) Pulse **Ejecutar** > **configuraciones de ejecución**
   b) Pulse **MQTTV3SampleSubscriber** > **Ejecutar**.

Abra la vista **Consola**. El suscriptor está esperando una publicación.

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
```

   c) Pulse **MQTTV3SamplePublisher** > **Ejecutar**.

Abra la vista **Consola**. Verá la publicación que ha creado el publicador.

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
```

   d) Cambie a la vista de consola del suscriptor.

El icono para cambiar consolas es  .

El suscriptor ha recibido la publicación:

```
MQTTV3Sample/Java/v3
Message: Message from MQTTv3 Java client
QoS:  2
```

Iniciación a Cliente MQTT para Java en Android

Puede instalar un Aplicación Java de ejemplo de cliente MQTT para Android que intercambie mensajes con un servidor MQTT. La aplicación utiliza una biblioteca de cliente del SDK de MQTT SDK de IBM. Puede compilar la aplicación usted mismo o descargar una aplicación de ejemplo ya compilada.

Antes de empezar

- Para saber las plataformas soportadas y de referencia de cliente MQTT, consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.
- Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
- La aplicación de ejemplo de cliente MQTT funciona en Ice Cream Sandwich (Android 4.0) y superior. Esta versión de Android también ofrece una resolución de pantalla más nítida en tabletas.

Acerca de esta tarea

El Aplicación Java de ejemplo de cliente MQTT para Android se denomina "`mqttExerciser`". Esta aplicación utiliza una biblioteca de cliente del SDK de MQTT e intercambia mensajes con un servidor MQTT.

Puede compilar la aplicación de ejemplo usted mismo y después exportarla de Eclipse como `mqttExerciser.apk`, o utilizar la aplicación de ejemplo precompilada disponible en el archivo `mqttExerciser.apk` en la carpeta *sdkroot*`\SDK\clients\android\samples\apks` de Mobile Messaging and M2M Paquete de clientes. Si decide compilar usted mismo la aplicación, el entorno de desarrollo que compile está ajustada para incluir mensajería móvil en aplicaciones for Android. Esto le resultará de ayuda cuando empiece a incluir mensajería móvil en sus aplicaciones.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 . Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte “Iniciación a los servidores MQTT” en la página 139.
2. Obtenga las herramientas adecuadas.

   Instale un kit de desarrollo de Java (JDK) Versión 6 o posterior. Debido a que está desarrollando una Java app for Android, JDK debe proceder de Oracle. Puede obtener el JDK de Descargas de Java SE.

   También necesita un entorno de desarrollo de Eclipse. Debe ser Eclipse 3.6.2 (Helios) o superior. Eclipse debe tener un compilador Java de nivel 6 por lo menos, para coincidir con el JDK. Puede obtener todo esto de Eclipse Foundation.

   Finalmente, necesitará el de Android. Puede obtenerlo en Obtener el SDK de Android.
3. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.

   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre `MQTT`. La vía de acceso a esta carpeta se denomina aquí *sdkroot*.

   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *sdkroot*. La expansión crea un árbol de directorios que empieza en *sdkroot*`\SDK`.
4. Opcional: Compile la aplicación de ejemplo mqttExerciser for Android.

   Configure las herramientas Eclipse y Android e importe y compile el proyecto `mqttExerciser` desde el SDK de MQTT .

   Nota: SI no desea hacerlo ahora, puede utilizar la aplicación de ejemplo precompilada disponible con el nombre de archivo `mqttExerciser.apk` en la carpeta *sdkroot*`\SDK\clients\android\samples\apks` del SDK de MQTT.

   a) Inicie el entorno de desarrollo de Eclipse con el JRE desde el JDK.

   ```
   eclipse -vm "JRE path"
   ```

b) Seleccione e instale un conjunto de paquetes y plataformas del SDK de Android.

Consulte Adición de plataformas y paquetes para ver la lista de plataformas y paquetes que Google recomienda.

Nota: La plataforma del SDK debe ser la API Android de nivel 16 o posterior. Con niveles de API anteriores el proyecto no se podría compilar satisfactoriamente.

c) Añada el plug-in de ADT (Android Development Tools) a Eclipse.

d) Importe el proyecto de aplicación `mqttExerciser` en Eclipse, y arregle los errores.

i) Importe el proyecto de aplicación de ejemplo desde el SDK de MQTT , en la vía de acceso *sdkroot*`\SDK\clients\android\samples\mqttExerciser`.

En la vista **Problemas** se muestra una lista con muchos errores de compilación. Resolverá los errores de compilación en los próximos pasos.

ii) Copie la biblioteca `org.eclipse.paho.client.mqttv3.jar` en la carpeta **libs** del proyecto Android . Windows Por ejemplo, en Windows, se encuentra bajo la carpeta *sdkroot*`\SDK\clients\java` . Se muestra la ventana **Operación de archivos**. Acepte la selección de **Copiar archivos** y pulse **Aceptar**.

iii) Pulse con el botón derecho del ratón en la carpeta del proyecto, `com.ibm.msg.android`; pulse **Herramientas de Android ...** > **Añadir biblioteca de soporte ...**. Lea y acepte los términos de la licencia y pulse **Instalar**.

iv) Pulse con el botón derecho del ratón en la carpeta del proyecto, `com.ibm.msg.android`; pulse **Herramientas de Android ...** > **Arreglar propiedades de proyecto**.

v) Si el espacio de trabajo aún tiene unos 84 errores, sobre sobrescribir un método de super clase, probablemente el nivel de conformidad está definido a 1.5 o inferior. El SDK de Android versión 16 espera que el nivel de nivel del compilador no sea superior a 1.5. Para arreglar los errores restantes, siga los pasos siguientes:

a) Compruebe y (en caso necesario) actualice el SDK de Android y los plug-ins de Eclipse correspondientes a la versión 17 del SDK de Android.

b) Pulse con el botón derecho del ratón en la carpeta de proyecto **com.ibm.msg.android** y, a continuación, seleccione **Propiedades** > **Compilador Java**. Compruebe el nivel de conformidad del compilador, defínalo como mínimo a 1.6 y recompile el espacio de trabajo.

El proyecto se compilará con algunos avisos pero sin errores.

5. Instale e inicie el Aplicación Java de ejemplo de cliente MQTT en un dispositivo Android.

Consulte la página de `developer.android.com` Ejecutar la aplicación.

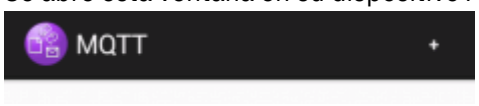
Si ha compilado usted mismo la aplicación como un proyecto Eclipse, puede iniciar la aplicación desde Eclipse.

Si tiene el archivo de paquete de aplicación (APK) `mqttExerciser.apk`, puede instalarlo fuera de Eclipse utilizando el mandato de instalación (install) de Android Debug Bridge (ADB). Este mandato toma la ubicación del archivo APK como argumento. Si utiliza la aplicación de ejemplo precompilada, la ubicación es *sdkroot*`\SDK\clients\android\samples\apks\mqttExerciser.apk`.

6. Utilice la aplicación de ejemplo mqttExerciser for Android para conectarse, suscribirse y publicar en un tema.

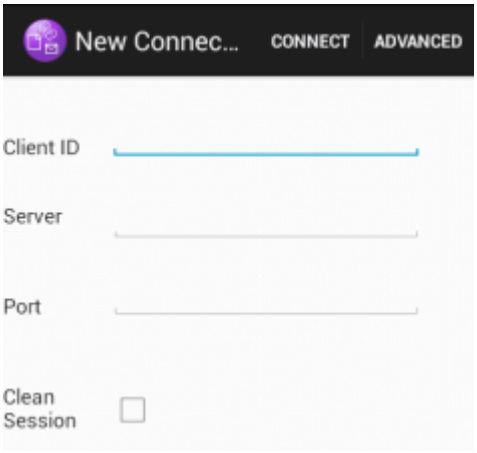
a) Abra Aplicación Java de ejemplo de cliente MQTT para Android.

Se abre esta ventana en su dispositivo Android:



b) Conectar con un servidor MQTT .

i) Pulse en el signo **+** para abrir una nueva conexión MQTT.



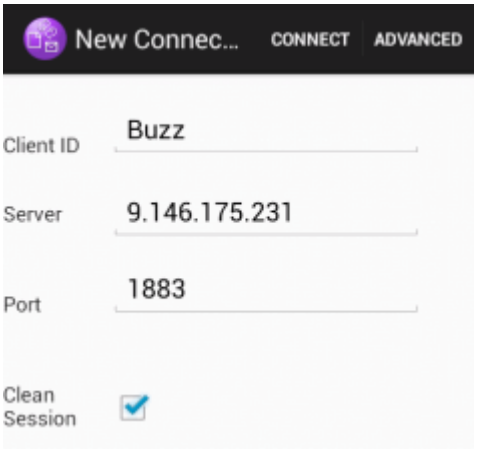
ii) Escriba cualquier identificador exclusivo en el campo **ID de cliente**. Tenga paciencia, puede que, al escribir, las pulsaciones de teclas sean lentas.

iii) En el campo **Servidor** escriba la dirección IP del servidor MQTT.

Éste es el servidor que ha elegido en el paso principal la primera vez. La dirección IP no puede ser `127.0.0.1`

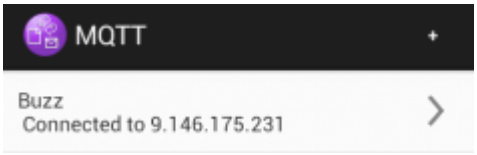
iv) Escriba el número de puerto de la conexión MQTT.

El número de puerto predeterminado para una conexión MQTT normal es 1883.



v) Pulse **Conectar**.

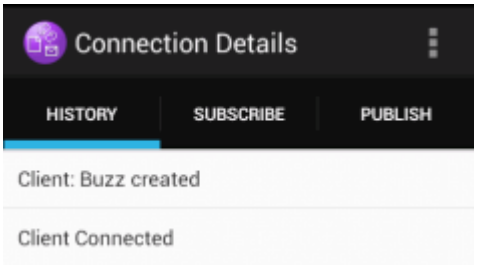
Si la conexión es satisfactoria, verá un mensaje "Conectando" seguido de esta ventana:



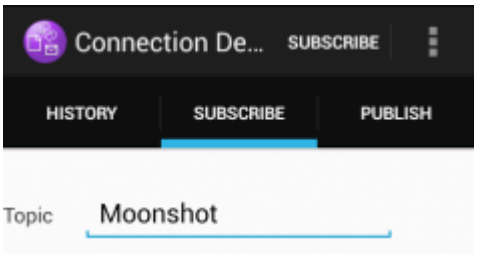
c) Suscríbase a un tema.

i) Pulse en el mensaje **Conectado**.

Se abre la ventana **Detalles de conexión** donde se muestra el historial:



ii) Pulse en la pestaña **Suscribir** y especifique una serie de tema.

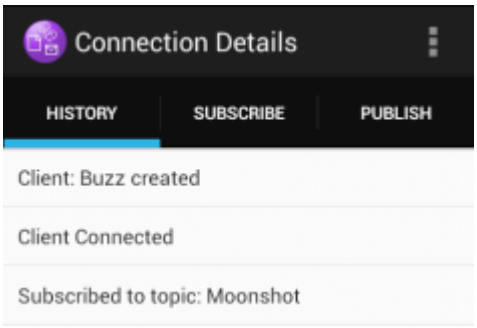


iii) Pulse en la acción **Suscribir**.

Aparece un mensaje "Suscrito" durante un breve tiempo.

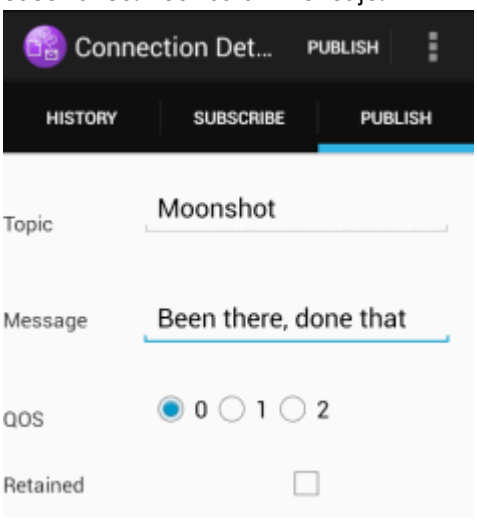
iv) Pulse en la pestaña **Historial**.

Ahora en el historial se incluye la suscripción:



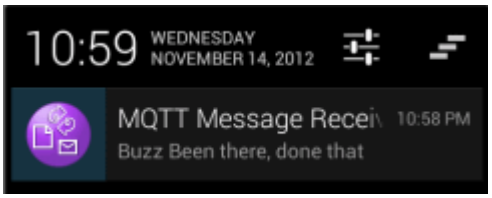
d) Publique en ese mismo tema.

i) Pulse en la pestaña **Publicar** y especifique la misma serie de tema que ha especificado al suscribirse. Escriba un mensaje.

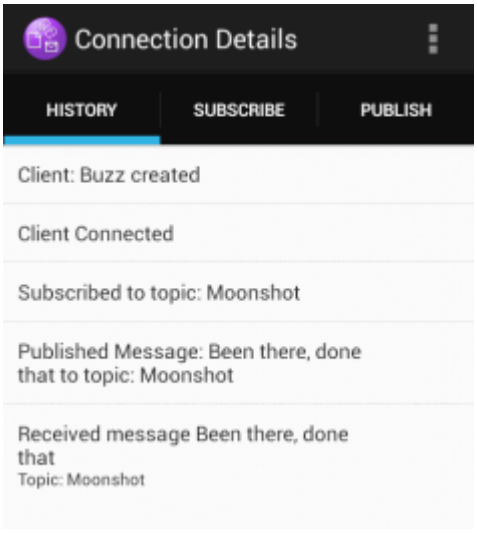


ii) Pulse en la acción **Publicar**.

Se muestran dos mensajes durante un breve tiempo, "Publicado" seguido de "Suscrito". La publicación se muestra en el área de estado (tire de la barra de separación hacia abajo para abrir la ventana de estado).



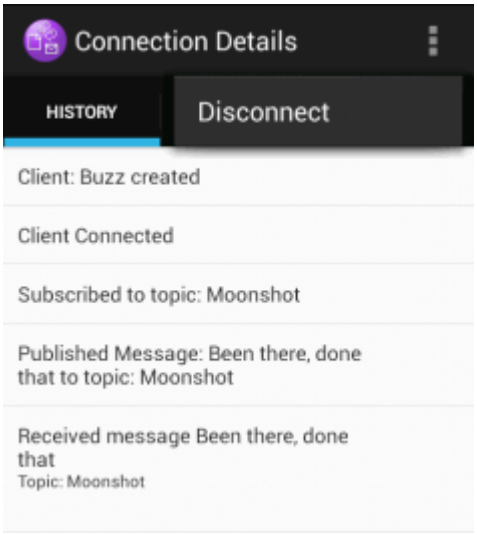
iii) Pulse en la pestaña **Historial** para ver el historial completo.



e) Desconecte la instancia de cliente.

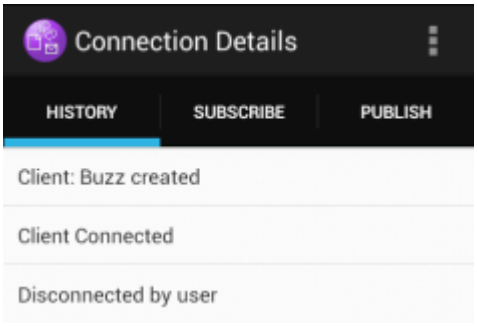
i) Pulse en el icono de menú de la barra de acciones. 

El Aplicación Java de ejemplo de cliente MQTT para Android añade el botón **Desconectar** en la ventana MQTT **Detalles de conexión**.

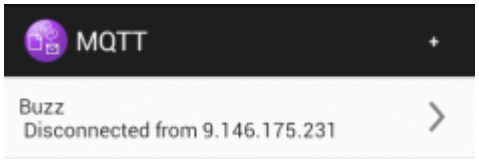


ii) Pulse **Desconectar**.

El estado de conexión cambia a desconectado:



f) Pulse **Atrás** para volver a la lista de sesiones de Aplicación Java de ejemplo de cliente MQTT.



- Pulse en el signo más para iniciar una nueva sesión de Aplicación Java de ejemplo de cliente MQTT.
- Pulse en el cliente desconectado para reconectarlo.
- Pulse **Atrás** para volver al launchpad.

g) Pulse el botón de tarea para ver una lista de las aplicaciones en ejecución. Busque el Aplicación Java de ejemplo de cliente MQTT. Deslice el icono fuera de la pantalla para cerrarlo.

Qué hacer a continuación

Si ha compilado usted la aplicación de ejemplo, ya está preparado para desarrollar sus propias aplicaciones Android que llamen a bibliotecas MQTT para intercambiar mensajes. Puede modelar sus apps de Android en las clases de `mqttExerciser`. Para estudiar el ejemplo, genere el Javadoc de las clases de `com.ibm.msg.android` y `com.ibm.msg.android.service` en el proyecto `mqttExerciser`.

Información relacionada

Gestión de Proyectos de Eclipse con ADT

Iniciación a Cliente MQTT de mensajería para JavaScript

Puede empezar a trabajar con Cliente MQTT de mensajería para JavaScript yendo a la página inicial de ejemplo del cliente de mensajería y examinando los recursos a los que enlaza. Para ver esta página de inicio, debe configurar un servidor MQTT para aceptar conexiones de la Páginas de JavaScript de ejemplo de cliente de mensajería MQTT, a continuación, escriba el URL que ha configurado en el servidor en un navegador web. Cliente MQTT de mensajería para JavaScript se inicia automáticamente en el dispositivo y se muestra la página inicial de ejemplo del cliente de mensajería. Esta página contiene enlaces a programas de utilidad, documentación de la interfaz de programación, una guía de aprendizaje y otra información útil.

Antes de empezar

Para usos avanzados o para uso en producción, deseará modificar o eliminar la página de inicio de ejemplo del cliente de mensajería. Tenga en cuenta que no se garantiza que las interfaces de usuario generadas a partir del código de ejemplo sean compatibles con los estándares o los requisitos de accesibilidad.

Necesita un servidor MQTT para dar soporte a Cliente MQTT de mensajería para JavaScript. Este servidor debe dar soporte al protocolo MQTT V3.1 a través de WebSockets. IBM MessageSight, y IBM WebSphere MQ Version 7.5.0, Fix Pack 1 y versiones posteriores, dan soporte a MQTT protocol sobre WebSockets. Consulte “Iniciación a los servidores MQTT” en la página 139. Para instalar una versión gratuita de

evaluación de 90 días de IBM WebSphere MQ, consulte "Instalación de IBM WebSphere MQ" en la página 142.

El WebSocket protocol está establecido recientemente. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquee el tráfico de WebSockets. De forma similar, si el navegador todavía no da soporte a WebSocket protocol[1]no podrá utilizar el programa de utilidad cliente o las guías de aprendizaje que hay disponibles en la página inicial de ejemplo del cliente de mensajería. En la tabla Tabla 1 en la página 25 se listan los navegadores para los cuales se ha comprobado que la versión más reciente funciona con el cliente de mensajería.

Tabla 1. Navegadores soportados para utilizar con Cliente MQTT de mensajería para JavaScript

Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 y posterior Chrome for Android 25.0 y posterior	Safari 6.0 y posterior Chrome 14.0 y posterior	Firefox 6.0 y posterior Chrome 14.0 y posterior	Firefox 6.0 y posterior Chrome 14.0 y posteriores

Acerca de esta tarea

La mayoría de los pasos de esta tarea son para configurar el servidor MQTT. Todo lo necesario para acceder al cliente de mensajería para JavaScript es ejecutar un navegador que admita WebSocket protocol.

En IBM WebSphere MQ, siga los pasos para habilitar IBM WebSphere MQ Telemetry creando los canales de ejemplo. Establezca conexión con el canal MQTT WebSockets de ejemplo predeterminado en el puerto 1883. El URL de la página de inicio de ejemplo del cliente de mensajería es `http://`*`hostname`*`:1883` en IBM WebSphere MQ.

En IBM MessageSight, instale y configure el dispositivo, configure el concentrador de mensajería para que acepte conexiones y cree un punto final MQTT WebSockets.

Procedimiento

1. Descargue Mobile Messaging and M2M Paquete de clientes, y elija un servidor MQTT al que pueda conectar la aplicación cliente.

   Consulte "Iniciación a los clientes MQTT" en la página 11.
2. Configure el servidor MQTT para que acepte conexiones de las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript.
   - En IBM WebSphere MQ:
     - Si ya tiene un gestor de colas de IBM WebSphere MQ configurado para MQTT, cambie el protocolo de la definición de canal para que admita tanto MQTT como HTTP. Consulte **ALTER CHANNEL**.
     - Para crear un gestor de colas de IBM WebSphere MQ y configurar el punto final MQTT WebSockets de ejemplo, complete una de las tareas siguientes:
       - "Configuración del servicio de MQTT desde la línea de mandatos" en la página 143
       - "Configuración del servicio de MQTT con IBM WebSphere MQ Explorer" en la página 146
3. Abra un navegador web en el dispositivo.
4. Escriba el URL de la página inicial de ejemplo del cliente de mensajería.
   - En IBM WebSphere MQ, es `http://`*`hostname`*`:1883`
   - En IBM MessageSight, es `http://`*`hostname`*`:`*`port`*

[1] En concreto, si no da soporte al estándar RFC 6455 (WebSocket).

donde *hostname* es el nombre DNS o la dirección IP del socket Ethernet que ha configurado en el dispositivo IBM MessageSight como el punto final al que se va a conectar el cliente, y *port* es el número de puerto TCP/IP que ha asignado al punto final para el cliente.

Se abre la página de inicio de ejemplo del cliente de mensajería.

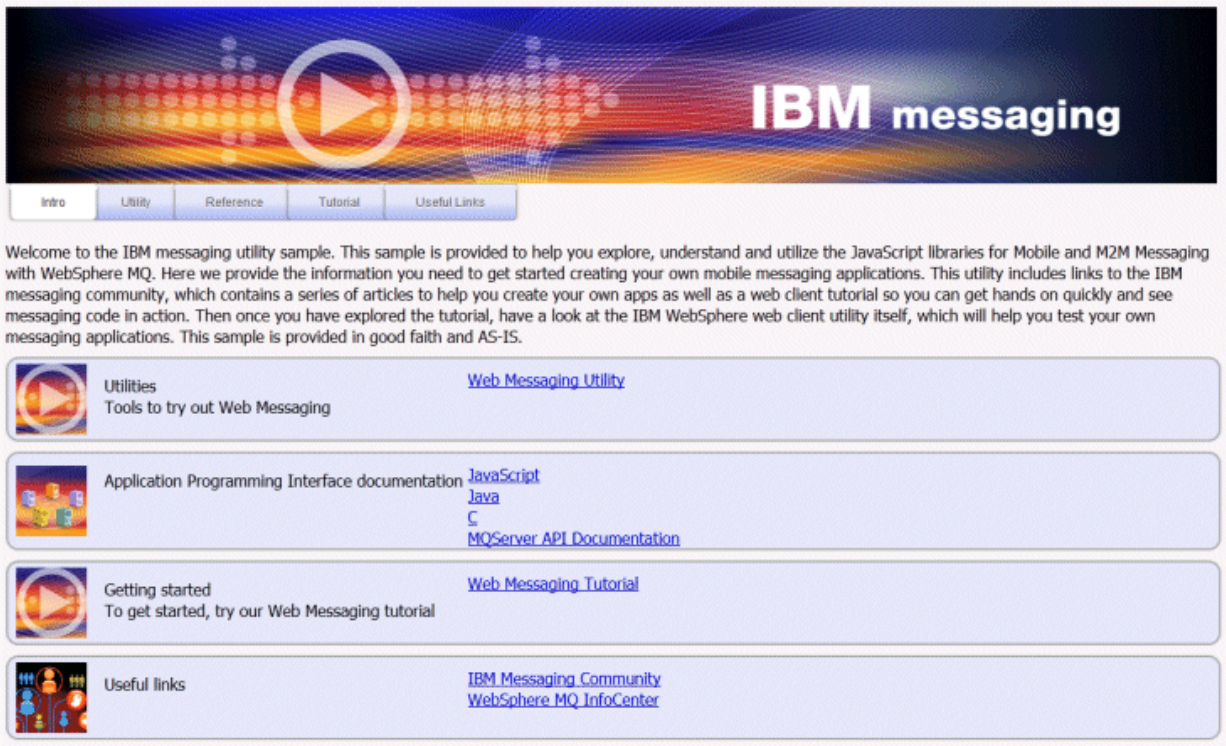


Figura 9. Página de inicio de ejemplo de Cliente MQTT de mensajería para JavaScript

Resultados

Ha configurado un canal MQTT para WebSockets.

En la página de inicio de ejemplo de Cliente MQTT de mensajería para JavaScript, pulse **Utilidad de mensajería web** para probar las distintas funciones de la API del cliente de mensajería. Por ejemplo, puede conectarse al gestor de colas, suscribirse a los mensajes y después publicar algunos mensajes. También puede pulsar **Guía de aprendizaje de mensajería web** para aprender a crear una página web que llame a la API del cliente de mensajería MQTT para JavaScript.

Conceptos relacionados

"El Cliente MQTT de mensajería para JavaScript y las aplicaciones web" en la página 119

"Cómo programar aplicaciones de mensajería en JavaScript" en la página 123

Tareas relacionadas

"Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets" en la página 79
Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

Iniciación al cliente MQTT para C

Puede empezar a trabajar con el ejemplo de cliente MQTT seguro para C en cualquier plataforma en la que pueda compilar el código C fuente. Verifique que puede ejecutar el cliente MQTT de ejemplo para C con IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

Antes de empezar

- Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
- Para saber las plataformas soportadas y de referencia de cliente MQTT para C. Consulte el apartado Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.

Acerca de esta tarea

Siga esta tarea para compilar y ejecutar el cliente MQTT de ejemplo para C en Windows desde la línea de mandatos o desde Microsoft Visual Studio 2010. En el ejemplo de línea de mandatos también se utiliza Microsoft Visual Studio 2010 para compilar el cliente. Modifique los scripts de línea de mandatos para compilar y ejecutar el ejemplo en otras plataformas.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 . Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte "Iniciación a los servidores MQTT" en la página 139.
2. Instale un entorno de desarrollo C en la plataforma en la que está trabajando.

   Los archivos make de los ejemplos de este tema están dirigidos a las siguientes herramientas:

   - 

Para iOS, en Apple Mac con OS X 10.8.2 con las herramientas de desarrollo de iOS de Xcode.
   - 

Para Linux, gcc versión 4.4.6 desde Red Hat® Enterprise Linux versión 6.2.

     El nivel mínimo soportado de la biblioteca C `glibc` es el 2.12, y del kernel de Linux es el 2.6.32.
   - 

Para Microsoft Windows, Visual Studio versión 10.0.
3. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.

   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre `MQTT`. La vía de acceso a esta carpeta se denomina aquí *`sdkroot`*.

   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *`sdkroot`*. La expansión crea un árbol de directorios que empieza en *`sdkroot`*\SDK.
4. Opcional: Siga los pasos de "Creación de las bibliotecas de cliente MQTT para C" en la página 31.

   Realice este paso únicamente si el Mobile Messaging and M2M Paquete de clientes no incluye la biblioteca de cliente C segura para la plataforma de destino.
5. Compile y ejecute la aplicación C de ejemplo de cliente MQTT , `MQTTV3Sample.c`.

   - Desde la línea de mandatos, siga los pasos que se detallan en "Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde la línea de mandatos" en la página 27.
   - Desde un IDE, siga los pasos que se detallan en "Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde Microsoft Visual Studio" en la página 28.

Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde la línea de mandatos

Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde la línea de mandatos. El ejemplo se encuentra en el SDK de MQTT. Ejemplifica un publicador y un suscriptor MQTT.

Antes de empezar

Instale un entorno de desarrollo C; por ejemplo, Microsoft Visual Studio 2010 como se utiliza en el ejemplo.

Acerca de esta tarea

Compile y ejecute el ejemplo de C, `MQTTV3Sample`, en el subdirectorio de clientes del SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedimiento

Cree un script en el directorio de ejemplos de cliente para compilar y ejecutar `Sample` en la plataforma elegida.

El siguiente script compila y ejecuta el ejemplo en una plataforma Windows de 32 bits, compilado en Microsoft Visual Studio 2010. Ejecute el script desde el subdirectorio*sdkroot*`\SDK\clients\c\samples`.

```
@echo off
setlocal
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

Resultados

El publicador y el suscriptor escriben la salida en sus ventanas de mandatos:

```
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
MQTTV3Sample.c
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 10. Salida del publicador

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:           MQTTV3Sample/C/v3
Message:         Message from MQTTv3 C client
QoS:             2
```

Figura 11. Salida del suscriptor

Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde Microsoft Visual Studio

Compilar y ejecutar la aplicación C de ejemplo de cliente MQTT desde Microsoft Visual Studio. El ejemplo se encuentra en Mobile Messaging and M2M Paquete de clientes. Ejemplifica un publicador y un suscriptor MQTT.

Antes de empezar

El ejemplo utiliza Microsoft Visual Studio 2010. Puede utilizar otros entornos de desarrollo de C en Windows y otras plataformas; por ejemplo Eclipse IDE for C/C++ Developers.

Acerca de esta tarea

Compile y ejecute el ejemplo C, `MQTTV3Sample` con Microsoft Visual Studio 2010. `MQTTV3Sample.c` se encuentra en el subdirectorio de clientes del SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedimiento

1. Inicie Microsoft Visual Studio.
2. Cree un proyecto nuevo a partir de código existente.
   a) Pulse **Archivo** > **Nuevo** > **Proyecto a partir de código existente**.
   b) Seleccione **Visual C++** como el tipo de proyecto a crear.
   c) Pulse **Siguiente**.
3. Especifique los parámetros en la ventana **Ubicación de proyecto y archivos fuente**.
   a) Pulse en **Examinar** y vaya al directorio *sdkroot*`\SDK\clients\c\samples`.
   b) Ponga como nombre de proyecto: `MQTTV3Sample`.
   c) Pulse **Siguiente**.
4. Seleccione **Proyecto de aplicación de consola** en la lista **Tipo de proyecto**. Pulse **Finalizar**.
5. Configure únicamente la configuración de depuración.

   De forma predeterminada, Microsoft Visual Studio crea una configuración de release y una configuración de depuración. En la guía de aprendizaje, configurará la configuración de depuración. Para eliminar los errores de compilación, deseleccione la opción **Compilar** en la configuración de release.

   a) Pulse **Proyecto** > **MQTTV3Sample Propiedades** > **Gestor de configuración**. Seleccione **Release** como **Configuración de solución activa** y deseleccione **Compilar**.
   b) Seleccione **Depurar** como **Configuración de solución activa** > **Cerrar**.

      Compruebe que modifica la configuración de depuración en todos los pasos siguientes.
6. Modifique la configuración de **C/C++** en las **Páginas de propiedades de MQTTV3Sample**.
   a) En la ventana **MQTTV3Sample Páginas de propiedades** , abra **Propiedades de configuración** > **C/C++** > **General**.
   b) En la lista de propiedades generales, pulse **Incluir directorios adicionales** y añada la vía de acceso de directorio a *sdkroot*`\SDK\clients\c\include` y pulse **Aplicar**.
7. Modifique la configuración del **Enlazador**
   a) Abra **Propiedades de configuración** > **Enlazador** > **General**.
   b) En la lista de propiedades generales, pulse **Directorios de biblioteca adicionales** y añada la vía de acceso de directorio a *sdkroot*`\SDK\clients\c\windows_ia32`
   c) En la lista de Propiedades del enlazador, pulse **Línea de mandatos**. Escriba `mqttv3c.lib` en el área de entrada de datos **Opciones adicionales** y pulse **Aplicar**.
8. Elimine el archivo de origen `MQTTV3SSample.c` del proyecto.
   a) Abra la carpeta **MQTTV3sample** > **Archivos de origen** en la ventana **Explorador de soluciones** .
   b) Pulse con el botón derecho **MQTTV3SSample.c** > **Excluir del proyecto**
9. Compile el proyecto `MQTTV3Sample`.
   a) Pulse con el botón derecho del ratón sobre el proyecto **MQTTV3sample** en el Explorador de soluciones y pulse **Compilar**.

      La compilación se completará sin errores.
10. Añada dos nuevos proyectos para ejecutar **`MQTTV3Sample`** como instancia de tiempo de ejecución tanto de Suscriptor como de Publicador.

    Los proyectos de Publicador y de Suscriptor contendrán los mandatos para ejecutar **`MQTTV3Sample`**. No se compilan y no contienen código.

a) En el **Explorador de soluciones**, pulse con el botón derecho del ratón en **Solución ` MQTTV3Sample `** > **Añadir** > **Nuevo proyecto**.

b) Escriba `Subscriber` en el campo **Nombre** . Deje **Aplicación de consola de Win32** seleccionado. Pulse **Aceptar**.

La **Aplicación de consola de Win32** se iniciará.

c) En el **Asistente de aplicación de Win32**, pulse **Siguiente**. Seleccione **Proyecto vacío** > **Finalizar**

d) Repita estos pasos para añadir un proyecto de Publicador.

e) Pulse con el botón derecho del ratón sobre el proyecto de Suscriptor y pulse en **Definir como proyecto de Inicio**

Se mostrará la ventana **Explorador de soluciones** en Figura 12 en la página 30.

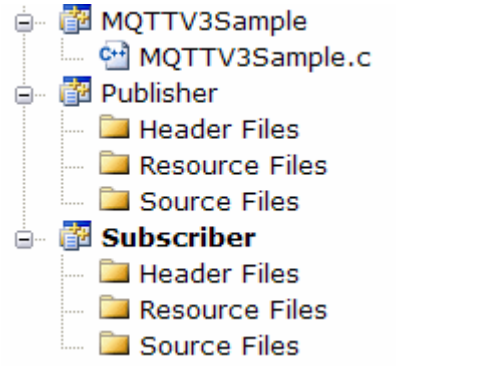


Figura 12. Solución MQTTV3Sample

11. Configure las páginas de propiedades del Suscriptor.

a) Pulse con el botón derecho del ratón en **Suscriptor** en el Explorador de soluciones **Propiedades** > **Propiedades de configuración** > **Propiedades** > **Depuración**

Compruebe que el título de la ventana sea **Páginas de propiedades del Suscriptor**.

b) Pulse **Entorno**. Escriba `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` y pulse **Aplicar**.

Cambie *`sdkroot`* para que se adapte a su entorno.

c) Pulse en **Mandato** y reemplace `$(TargetPath)` por la vía de acceso al módulo `MQTTV3Sample`

Por ejemplo, *`sdkroot`*`\SDK\clients\c\samples\debug\MQTTV3Sample`

Cambie *`sdkroot`* para que se adapte a su entorno.

d) Pulse **Argumentos de mandato** y escriba `-a subscribe -b localhost -p 1883` y pulse **Aplicar**.

12. Configure las páginas de propiedades del Publicador.

Consejo: Puede cambiar el proyecto del que se muestran las páginas de propiedades pulsando en los proyectos en la ventana **Explorador de soluciones**.

a) Repita los pasos del Suscriptor para el Publicador.

El argumento del mandato es `-b localhost -p 1883`

13. Detenga el proceso de compilación que compila los proyectos de Publicador y Suscriptor.

a) En las páginas de propiedades de alguno de los proyectos pulse en **Gestor de configuración** y deseleccione**Compilar** para el Publicador y el Suscriptor tanto en la configuración de release como de depuración. Pulse **Cerrar**.

14. Ejecute el ejemplo.

a) Pulse **F5** para iniciar el Suscriptor

b) Pulse con el botón derecho del ratón en **Publicador** en el Explorador de soluciones, **Depurar** > **Iniciar una nueva instancia**

Resultados

El publicador y el suscriptor escriben la salida en las ventanas de mandatos. Visual Studio cierra la ventana del publicador. Mire en la ventana de suscriptor, la cual aparece en la imagen siguiente, y ciérrela luego.

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Topic:          MQTTV3Sample/C/v3
Message:        Message from MQTTv3 C client
QoS:            2
```

Figura 13. Salida del suscriptor

Qué hacer a continuación

Compilar y ejecutar el publicador y el suscriptor asíncronos.Los ejemplos son `MQTTV3ASample.c` y `MQTTV3ASSample.c` en *sdkroot*`\SDK\clients\c\samples`.

Creación de las bibliotecas de cliente MQTT para C

Siga estos pasos para crear las bibliotecas de cliente MQTT para C. En el tema se incluyen los conmutadores de compilación y enlace para diversas plataformas, y ejemplos de cómo crear las bibliotecas en iOS y Windows.

Antes de empezar

1. Cree la biblioteca de cliente C sólo cuando sea necesario. Enlace las bibliotecas de cliente preconstruidas del SDK (Software Development Kit) en el subdirectorio SDK`\clients\c` si alguna coincide con su plataforma de destino.
2. Configure un servidor MQTT para probar la biblioteca que vaya a crear con Aplicación C de ejemplo de cliente MQTT. Consulte “Iniciación a los servidores MQTT” en la página 139. Verifique la configuración del servidor ejecutando una de las aplicaciones de cliente MQTT de ejemplo.
3. Si va a compilar una versión seguro de la biblioteca C, que admita SSL (Secure Sockets Layer), también debe crear la biblioteca de OpenSSL. Consulte “Compilación del paquete de OpenSSL” en la página 45.

   Importante: La descarga y redistribución del paquete de OpenSSL está sujeta a una estricta regulación de importación y exportación y a condiciones de licencia de código abierto. Lea atentamente las restricciones y advertencias antes de decidir descargar el paquete.

Acerca de esta tarea

Siga las instrucciones de “Compilación del paquete de OpenSSL” en la página 45 para descargar y compilar la biblioteca OpenSSL. Debe compilar OpenSSL para compilar una versión segura de la biblioteca de cliente MQTT para C. No necesita OpenSSL para compilar una versión no segura de la biblioteca MQTT. En los pasos se incluyen ejemplos de cómo compilar la biblioteca para iOS y Windows.

Compile la biblioteca de cliente MQTT para C descargando las herramientas de biblioteca de desarrollo para C y el kit de desarrollo de software (SDK) de MQTT en su plataforma de compilación. Escriba un archivo make para compilar la biblioteca para su plataforma de destino, incorporando las opciones que se documentan en “Opciones de compilación deMQTT para distintas plataformas” en la página 32. A continuación se describen los pasos específicos para cada plataforma para crear y ejecutar un archivo make:

- iOS “Compilación de las bibliotecas de cliente MQTT para C en un Apple Mac para utilizar con dispositivos iOS” en la página 34
- Windows “Compilación de las bibliotecas de MQTT en Windows” en la página 40

Procedimiento

1. Instale un entorno de desarrollo C en la plataforma en la que está trabajando.

   Los archivos make de los ejemplos de este tema están dirigidos a las siguientes herramientas:

   - 

Para iOS, en Apple Mac con OS X 10.8.2 con las herramientas de desarrollo de iOS de Xcode.
   - 

Para Linux, gcc versión 4.4.6 desde Red Hat Enterprise Linux versión 6.2.

     El nivel mínimo soportado de la biblioteca C `glibc` es el 2.12, y del kernel de Linux es el 2.6.32.
   - 

Para Microsoft Windows, Visual Studio versión 10.0.
2. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.
   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre `MQTT`. La vía de acceso a esta carpeta se denomina aquí *`sdkroot`*.
   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *`sdkroot`*. La expansión crea un árbol de directorios que empieza en *`sdkroot`*\SDK.
3. Expanda el código fuente para las bibliotecas de cliente MQTT para C.

   El archivo comprimido con el código fuente es *`sdkroot`*\SDK\`clients\c\source.zip`.
4. Opcional: Compile OpenSSL.

   Consulte “Compilación del paquete de OpenSSL” en la página 45.
5. Compile las bibliotecas de cliente MQTT para C.

   En “Opciones de compilación deMQTT para distintas plataformas” en la página 32 se enumeran los mandatos y opciones para compilar las bibliotecas.

   Siga los pasos de los ejemplos siguientes para escribir un archivo make para compilar las bibliotecas de cliente MQTT para C para su plataforma de destino.

   - “Compilación de las bibliotecas de cliente MQTT para C en un Apple Mac para utilizar con dispositivos iOS” en la página 34
   - “Compilación de las bibliotecas de MQTT en Windows” en la página 40

Opciones de compilación deMQTT para distintas plataformas

En la siguiente tabla se listan el compilador y las opciones de compilación para compilar las bibliotecas de cliente MQTT para C en diversas plataformas.

Tabla 2. Opciones de compilación deMQTT para distintas plataformas

Plataforma	Compiler	Compiler Options	Linker Options	Extra Options
AIX	gcc	-fPIC -Os -Wall -DREVERSED -I *MQTTCLIENT_DIR*	-Wl,-G	
Linux s390x			-shared -Wl,-soname, libmqttv3c.so	-m 64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m 32
Linux ARM (glibc)	arm-linux-gcc			
Linux ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			
Windows 32 bits	cl	/D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS" / nologo /c /O2 /W3 /Fd /MD /TC	/nologo /machine:x86 /manifest kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib ws2_32.lib / implib:mqttv3c.lib) /pdb:mqttv3c.pdb) / map:mqttv3c.map)	
iOS ARMv7	gcc -arch armv7	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot / Applications/Xcode.app/\ Contents/Developer/Platforms/\ iPhoneOS.platform/Developer/SDKs/\ iPhoneOS6.0.sdk	-L/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk/usr/lib/system	
iOS ARMv7s	gcc -arch armv7s			

Tabla 2. Opciones de compilación deMQTT para distintas plataformas (continuación)

Plataforma	Compiler	Compiler Options	Linker Options	Extra Options
iOS ARMi386	gcc -arch i386	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot / Applications/ Xcode.app/\ Contents/Developer/ Platforms/\ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk	-L/Applications/Xcode.app/ Contents/Developer/ Platforms/ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk/usr /lib/system	

iOS *Compilación de las bibliotecas de cliente MQTT para C en un Apple Mac para utilizar con dispositivos iOS*

Siga estos pasos para escribir un archivo make para compilar las bibliotecas de cliente MQTT para C en un Apple Mac, para ser utilizadas posteriormente en dispositivo iOS.

Antes de empezar

1. Instale las herramientas de compilación, desarrolle y ejecute el archivo make en Apple Mac con OS X 10.8.2, o posterior.
2. Instale las herramientas de línea de mandatos para Xcode, que incluyen el programa **make** . Descargue las herramientas de línea de mandatos de Xcode.

Acerca de esta tarea

Cree un archivo make que compile las bibliotecas de cliente MQTT para C para iPhone o iPad ejecutando un procesador ARMv7 o ARMv7s y el simulador de iPhone que se ejecuta en un procesador i386-64 bits. Consulte List of iOS devices.

Consejo: En “Listado del archivo make MQTTios.mak” en la página 38 se lista el archivo make completo.

1. Copie y pegue el listado en un archivo.
2. Convierta el carácter inicial de cada línea después de un destino en un tabulador; consulte el paso “8” en la página 36.
3. Ejecútelo con el mandato que se detalla en el paso “9” en la página 38 del procedimiento.

Procedimiento

1. Descargue e instale las herramientas de desarrollo de iOS .
   a. Inicie sesión con un ID de usuario que tenga privilegios de administración.
   b. Compruebe que su Apple Mac tiene la versión 10.8.2 o posterior.

c. Vaya al sitio web Xcode para descargar Xcode desde el App Store de Mac.

d. Instale Xcode, el entorno de línea de mandatos y el simulador.

Si AppStore de Mac ofrece diversas versiones del simulador, elija la versión que sea compatible con el nivel de iOS que tiene previsto para su aplicación.

2. Cree el archivo make `MQTTios.mak`

Añada un prólogo:

```
# La salida de compilación se genera en el directorio actual.
# MQTTCLIENT_DIR debe apuntar al directorio base que contiene el código fuente del cliente
MQTT.
# El valor predeterminado de MQTTCLIENT_DIR es el directorio actual
# El valor predeterminado de TOOL_DIR es /Applications/Xcode.app/Contents/Developer/
Platforms
# El valor predeterminado de OPENSSL_DIR es sdkroot/openssl, relativo a sdkroot/sdk/
clients/c/mqttv3c/src
# OPENSSL_DIR debe apuntar al directorio base que contiene la compilación OpenSSL .
# Ejemplo: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. Establezca la ubicación del código fuente de MQTT .

Ejecute el archivo make en el mismo directorio que los archivos de origen MQTT o establezca el parámetro de línea de mandatos `MQTTCLIENT_DIR` :

```
make -f makefile MQTTCLIENT_DIR=raíz_sd/SDK/clients/c/mqttv3c/src
```

Añada las siguientes líneas al archivo make:

```
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
```

El ejemplo establece VPATH en el directorio donde **make** busca los archivos de origen que no se han identificado explícitamente; por ejemplo, todos los archivos de cabecera que son necesarios en la compilación.

4. Opcional: Establezca la ubicación de las bibliotecas de OpenSSL .

Este paso es necesario para crear las versiones SSL del cliente MQTT para bibliotecas C.

Establezca la vía de acceso predeterminada a las bibliotecas de OpenSSL en el mismo directorio que ha expandido el SDK de MQTT . Si no, defina `OPENSSL_DIR` como parámetro de línea de mandatos.

```
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../openssl-1.0.1c
endif
```

Consejo: *OpenSSL* es el directorio OpenSSL que contiene todos los subdirectorios OpenSSL . Puede que tenga que mover el árbol de directorios desde donde lo ha expandido, ya que contiene directorios padre vacíos innecesario.

5. Establezca los directorios de las herramientas de desarrollo.

Si ha instalado Xcode en otra ubicación, defina `TOOL_DIR` en la línea de mandatos.

```
dir_TOOL_ifndef
   TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}
```

6. Seleccione todos los archivos de origen necesarios para compilar cada biblioteca de MQTT . Asimismo, establezca el nombre y la ubicación de la biblioteca MQTT que desea crear.

Añada la línea siguiente al archivo make para listar todos los archivos de origen MQTT :

```
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
```

Los archivos fuente dependen de si va a compilar una biblioteca síncrona o asíncrona y de si ésta incluye SSL o no.

Añada una o más de estas líneas, que dependerán de lo que se tenga que compilar. Las bibliotecas compartidas se crean en el directorio `darwin_x86_64`.

- Síncrona, no segura:

```
MQTTLIB = mqttv3c
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c,
${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
```

- Síncrona, segura:

```
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
```

- Asíncrona, no segura:

```
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
SSLSocket.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
```

- Asíncrona, segura:

```
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a
```

7. Defina el compilador y las opciones del compilador.

   Consulte las opciones para distintas plataformas que se muestran en Opciones de compilación MQTT para distintas plataformas.

   a) Establezca el proyecto Gnu C y C++ (**gcc**) como compilador.

   Seleccione estos compiladores cruzados para compilar la biblioteca para distintos dispositivos y el simulador de iPhone:

```
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
```

   b) Añada las opciones del compilador.

```
CCFLAGS = -Os -Wall -fomit-frame-pointer
```

   c) Añada las vías de acceso de inclusión.

```
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L$
{SDK_ARM}/usr/lib/system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include -L$
{SDK_i386}/usr/lib/system
```

8. Defina los destinos de compilación.

   Consejo: Cada línea sucesiva que define la implementación de un destino debe comenzar con un carácter de tabulación.

   a) Defina como destino **all**.

   Cuando el destino es "all", se compilan todas las bibliotecas.

```
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}
```

   Al ponerlo primero en la lista, es el destino predeterminado.

a) Compile la biblioteca síncrona no segura `libmqttv3c.a`.

```
${MQTTLIB_DARWIN}: ${SOURCE_FILES}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

La sentencia `rm *.o` suprime todos los archivos de objeto que se crean para cada biblioteca. `lipo` concatena las tres bibliotecas en un archivo.

b) Compile la biblioteca asíncrona no segura `libmqttv3a.a`.

```
${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

La sentencia `rm *.o` suprime todos los archivos de objeto que se crean para cada biblioteca. `lipo` concatena las tres bibliotecas en un archivo.

c) Compile la biblioteca síncrona segura `libmqttv3cs.a`.

```
${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
$@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
$@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
$@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

La sentencia `rm *.o` suprime todos los archivos de objeto que se crean para cada biblioteca. `lipo` concatena las tres bibliotecas en un archivo.

d) Compile la biblioteca asíncrona segura `libmqttv3as.a`.

```
${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o
$@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o
$@.armv7s *.o
    rm *.o
```

```
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o
$@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@
```

La sentencia `rm *.o` suprime todos los archivos de objeto que se crean para cada biblioteca. `lipo` concatena las tres bibliotecas en un archivo.

e) Defina como destino **`clean`**.

Cuando se especifica "clean" como destino, se eliminan todos los archivos y directorios generados por el archivo make

```
.PHONY: limpiar
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. Ejecute el archivo make.

```
make -f  MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
```

Resultados

Se crean los siguientes archivos en el directorio *sdkroot/sdk/clients/c/mqttv3c/src/*darwin_x86_64.

```
libmqttv3c.a.armv7
libmqttv3c.a.armv7s
libmqttv3c.a.i386
libmqttv3c.a
libmqttv3a.a.armv7
libmqttv3a.a.armv7s
libmqttv3a.a.i386
libmqttv3a.a
libmqttv3cs.a.armv7
libmqttv3cs.a.armv7s
libmqttv3cs.a.i386
libmqttv3cs.a
libmqttv3as.a.armv7
libmqttv3as.a.armv7s
libmqttv3as.a.i386
libmqttv3as.a
```

Listado del archivo make `MQTTios.mak`

```
# La salida de compilación se genera en el directorio actual.
# MQTTCLIENT_DIR debe apuntar al directorio base que contiene el código fuente del cliente
MQTT.
# El valor predeterminado de MQTTCLIENT_DIR es el directorio actual
# El valor predeterminado de TOOL_DIR es /Applications/Xcode.app/Contents/Developer/Platforms
# El valor predeterminado de OPENSSL_DIR es sdkroot/openssl, relativo a sdkroot/sdk/clients/c/
mqttv3c/src
# OPENSSL_DIR debe apuntar al directorio base que contiene la compilación OpenSSL .
# Ejemplo: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
dir_TOOL_ifndef
   TOOL_DIR = /Applications/Xcode.app/Contents/Developer/Platforms endif
IPHONE_SDK = iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
IPHONESIM_SDK = iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
SDK_ARM = ${TOOL_DIR}/${IPHONE_SDK}
SDK_i386 = ${TOOL_DIR}/${IPHONESIM_SDK}

MQTTLIB = mqttv3c
```

```
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a

# compiler
CC = iPhoneOS.platform/Developer/usr/bin/gcc
CC_armv7 = ${TOOL_DIR}/${CC} -arch armv7
CC_armv7s = ${TOOL_DIR}/${CC} -arch armv7s
CC_i386 = ${TOOL_DIR}/${CC} -arch i386
CCFLAGS = -Os -Wall -fomit-frame-pointer
CCFLAGS_SO_ARM = ${CCFLAGS} -isysroot ${SDK_ARM} -I${OPENSSL_DIR}/include -L${SDK_ARM}/usr/lib/
system
CCFLAGS_SO_i386 = ${CCFLAGS} -isysroot ${SDK_i386} -I${OPENSSL_DIR}/include -L$
{SDK_i386}/usr/lib/system

# targets
all: ${MQTTLIB_DARWIN} ${MQTTLIB_DARWIN_A} ${MQTTLIB_DARWIN_AS} ${MQTTLIB_DARWIN_S}

${MQTTLIB_DARWIN}: ${SOURCE_FILES}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES} -DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -o $@.armv7s *.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_i386} -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_S}: ${SOURCE_FILES_S}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_S} -DOPENSSL -DUSE_NAMED_SEMAPHORES
-DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@

${MQTTLIB_DARWIN_AS}: ${SOURCE_FILES_AS}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7 -lssl -lcrypto -o $@.armv7 *.o
    rm *.o
    ${CC_armv7s} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
```

```
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/armv7s -lssl -lcrypto -o $@.armv7s
*.o
    rm *.o
    ${CC_i386} ${CCFLAGS_SO_i386} -c ${SOURCE_FILES_AS} -DMQTT_ASYNC -DOPENSSL
-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE
    libtool -static -syslibroot ${SDK_ARM} -L${OPENSSL_DIR}/i386 -lssl -lcrypto -o $@.i386 *.o
    rm *.o
    lipo -create $@.armv7 $@.armv7s $@.i386 -output $@

.PHONY: limpiar
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *Compilación de las bibliotecas de MQTT en Windows*

Siga estos pasos para escribir un archivo make para compilar las bibliotecas de cliente MQTT para C en Windows.

Antes de empezar

1. Si es necesario, instale una versión de **Make** en la estación de trabajo de compilación que sea compatible con los archivos make escritos para Gnu make; de lo contrario, descargue Gnu make and build it. Consulte Gnu Make. El sitio web, Make for Windows, proporciona una versión instalable de **Make** for Windows.
2. También necesitará mandatos de Linux para Windows para utilizar el 'clean' como destino en el ejemplo de archivo make. Puede obtener mandatos Linux para Windows en sitios web tales como Cygwin.

Acerca de esta tarea

Cree un archivo make que compile las bibliotecas de cliente MQTT para C para Windows de 32 bit.

Consejo: En “Listado del archivo make MQTTwin.mak” en la página 44 se lista el archivo make completo.

1. Copie y pegue el listado en un archivo.
2. Convierta el carácter inicial de cada línea después de un destino en un tabulador; consulte el paso “7” en la página 42.
3. Ejecútelo con el mandato que se detalla en el paso “9” en la página 44 del procedimiento.

Procedimiento

1. Cree el archivo make `MQTTwin.mak`

   Añada un prólogo:

   ```
   # La salida de compilación se genera en el directorio actual.
   # MQTTCLIENT_DIR debe apuntar al directorio base que contiene el código fuente del cliente
   MQTT.
   # El valor predeterminado de MQTTCLIENT_DIR es el directorio actual
   # El valor predeterminado de OPENSSL_DIR es sdkroot\openSSL, relativo a
   sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR debe apuntar al directorio base que contiene la compilación OpenSSL .
   # Ejemplo: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # Establecer el entorno de compilación, por ejemplo:
   # %comspec% /k "" C:\Archivos de programa \Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
   # set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
   ```

2. Establezca la ubicación del código fuente de MQTT .

   Ejecute el archivo make en el mismo directorio que los archivos de origen MQTT o establezca el parámetro de línea de mandatos `MQTTCLIENT_DIR` :

   ```
   make -f makefile MQTTCLIENT_DIR=raíz_sd/SDK/clients/c/mqttv3c/src
   ```

Añada las siguientes líneas al archivo make:

```
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
```

El ejemplo establece VPATH en el directorio donde **make** busca los archivos de origen que no se han identificado explícitamente; por ejemplo, todos los archivos de cabecera que son necesarios en la compilación.

3. Opcional: Establezca la ubicación de las bibliotecas de OpenSSL .

   Este paso es necesario para crear las versiones SSL del cliente MQTT para bibliotecas C.

   Establezca la vía de acceso predeterminada a las bibliotecas de OpenSSL en el mismo directorio que ha expandido el SDK de MQTT . Si no, defina OPENSSL_DIR como parámetro de línea de mandatos.

   ```
   ifndef OPENSSL_DIR
     OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../openssl-1.0.1c
   endif
   ```

   Consejo: *OpenSSL* es el directorio OpenSSL que contiene todos los subdirectorios OpenSSL . Puede que tenga que mover el árbol de directorios desde donde lo ha expandido, ya que contiene directorios padre vacíos innecesario.

4. Seleccione todos los archivos de origen necesarios para compilar cada biblioteca de MQTT . Asimismo, establezca el nombre y la ubicación de la biblioteca MQTT que desea crear.

   Añada la línea siguiente al archivo make para listar todos los archivos de origen MQTT :

   ```
   ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
   ```

   Los archivos fuente dependen de si va a compilar una biblioteca síncrona o asíncrona y de si ésta incluye SSL o no.

   Añada una o más de estas líneas, que dependerán de lo que se tenga que compilar. Las bibliotecas compartidas y los manifiestos se crean en el directorio `windows_ia32`.

   - Síncrona, no segura:

     ```
     MQTTLIB = mqttv3c
     MQTTDLL = windows_ia32/${MQTTLIB} .dll
     SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c,
     ${ALL_SOURCE_FILES}}
     MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
     ```

   - Síncrona, segura:

     ```
     MQTTLIB_S = mqttv3cs
     MQTTDLL_S = windows_ia32/${MQTTLIB_S} .dll
     SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
     MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
     ```

   - Asíncrona, no segura:

     ```
     MQTTLIB_A = mqttv3a
     MQTTDLL_A = windows_ia32/${MQTTLIB_A} .dll
     SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/
     SSLSocket.c, ${ALL_SOURCE_FILES}}
     MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
     ```

   - Asíncrona, segura:

     ```
     MQTTLIB_AS = mqttv3as
     MQTTDLL_AS = windows_ia32/${MQTTLIB_AS} .dll
     SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
     MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
     ```

5. Defina el compilador y las opciones del compilador.

Consulte las opciones para distintas plataformas que se muestran en Opciones de compilación MQTT para distintas plataformas.

a) Defina Microsoft Visual C++ como el compilador.

```
CC = cl
```

b) Añada las opciones del preprocesador.

```
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
```

c) Añada las opciones del compilador.

```
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
```

d) Añada las vías de acceso de inclusión.

```
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
```

e) Opcional: Añada una opción de preprocesador, si va a compilar una biblioteca segura.

```
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
```

f) Opcional: Añada los archivos de cabecera de OpenSSL, si va a compilar una biblioteca segura.

```
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/
```

Consejo: Los archivos de cabecera se encuentran en `${OPENSSL_DIR}/inc32/openssl`, pero el archivo `ssl.h` está incluido en "`openssl/ssl.h`".

6. Defina el enlazador y las opciones del enlazador.

a) Defina Microsoft Visual C++ como el enlazador.

```
LD = link
```

b) Añada las opciones del enlazador.

```
LINKFLAGS = /nologo /machine:x86 /manifest /dll
```

c) Añada las vías de acceso de las bibliotecas.

```
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
          advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
          odbc32.lib odbccp32.lib ws2_32.lib
```

d) Añada los archivos de salida intermedios.

```
IMP = /implib: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LIBMAP = /map: ${@:.dll=.map}
```

e) Opcional: Añada las bibliotecas OpenSSL, si va a compilar una biblioteca segura.

```
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
```

f) Opcional: Añada las vías de acceso a las bibliotecas OpenSSL, si va a compilar una biblioteca segura.

```
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib
```

7. Defina los cuatro destinos de compilación.

a) Defina como destino **`all`**.

Consejo: Cada línea sucesiva que define la implementación de un destino debe comenzar con un carácter de tabulación.

Cuando el destino es "all", se compilan todas las bibliotecas.

```
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}
```

b) Compile la biblioteca síncrona no segura `mqttv3c.dll`.

```
${MQTTDLL}: ${SOURCE_FILES}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL}
    ${MANIFEST}
```

Con la sentencia `-rm ${CURDIR}/MQTTAsync.obj` se suprimen los `MQTTAsync.obj` creados para un destino anterior. `MQTTAsync.obj` y `MQTTClient.obj` son mutuamente exclusivos.

c) Compile la biblioteca asíncrona no segura `mqttv3a.dll`.

```
${MQTTDLL_A}: ${SOURCE_FILES_A}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL_A}
    ${MANIFEST_A}
```

Con la sentencia `-rm ${CURDIR}/MQTTClient.obj` se suprimen los `MQTTClient.obj` creados para un destino anterior. `MQTTAsync.obj` y `MQTTClient.obj` son mutuamente exclusivos.

d) Compile la biblioteca síncrona segura `mqttv3cs.dll`.

```
${MQTTDLL_S}: ${SOURCE_FILES_S}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:
${MQTTDLL_S}
    ${MANIFEST_S}
```

Con la sentencia `-rm ${CURDIR}/MQTTAsync.obj` se suprimen los `MQTTAsync.obj` creados para un destino anterior. `MQTTAsync.obj` y `MQTTClient.obj` son mutuamente exclusivos.

e) Compile la biblioteca asíncrona segura `mqttv3as.dll`.

```
${MQTTDLL_AS}: ${SOURCE_FILES_AS}
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out:
$(MQTTDLL_AS}
    $(MANIFEST_AS}
```

Con la sentencia `-rm $(CURDIR}/MQTTClient.obj` se suprimen los `MQTTClient.obj` creados para un destino anterior. `MQTTAsync.obj` y `MQTTClient.obj` son mutuamente exclusivos.

f) Defina como destino **`clean`**.

Cuando se especifica "clean" como destino, se eliminan todos los archivos y directorios generados por el archivo make

```
.PHONY: limpiar
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. Defina la vía de acceso de Windows donde ejecutar el archivo make.

Cambie las partes en cursiva por los valores correspondientes según so instalación.

a) Defina el entorno Microsoft Visual Studio.

```
%comspec% /k ""C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
```

b) Establezca la variable `Path` para incluir el programa make y el entorno de mandatos Linux .

```
set path=%path%;C:\Program Files\GnuWin32\bin;C:\cygwin\bin
```

9. Ejecute el archivo make.

```
make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/SDK/clients/c/mqttv3c/src
```

Consejo: El carácter separador de archivo debe ser una barra inclinada hacia delante, no hacia atrás.

Resultados

Se crean los siguientes archivos en el directorio *sdkroot*\SDK*clients*\c*mqttv3c*\src\windows_ia32.

```
mqttv3a.dll
mqttv3a.dll.manifest
mqttv3a.exp
mqttv3a.lib
mqttv3a.map
mqttv3as.dll
mqttv3as.dll.manifest
mqttv3as.exp
mqttv3as.lib
mqttv3as.map
mqttv3c.dll
mqttv3c.dll.manifest
mqttv3c.exp
mqttv3c.lib
mqttv3c.map
mqttv3cs.dll
mqttv3cs.dll.manifest
mqttv3cs.exp
mqttv3cs.lib
mqttv3cs.map
```

Listado del archivo make `MQTTwin.mak`

```
# La salida de compilación se genera en el directorio actual.
# MQTTCLIENT_DIR debe apuntar al directorio base que contiene el código fuente del cliente MQTT.
# El valor predeterminado de MQTTCLIENT_DIR es el directorio actual
# El valor predeterminado de OPENSSL_DIR es sdkroot\openSSL, relativo a
sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR debe apuntar al directorio base que contiene la compilación OpenSSL .
# Ejemplo: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Establecer el entorno de compilación, por ejemplo:
# %comspec% /k "" C:\Archivos de programa \Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
# set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/../../../../../../openssl-1.0.1c
endif

ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}

MQTTLIB = mqttv3c
MQTTDLL = windows_ia32/${MQTTLIB} .dll
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MANIFEST = mt -manifest ${MQTTDLL}.manifest -outputresource: ${MQTTDLL}\; 2
MQTTLIB_S = mqttv3cs
MQTTDLL_S = windows_ia32/${MQTTLIB_S} .dll
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_A = mqttv3a
MQTTDLL_A = windows_ia32/${MQTTLIB_A} .dll
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
MQTTLIB_AS = mqttv3as
MQTTDLL_AS = windows_ia32/${MQTTLIB_AS} .dll
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MANIFEST_S = mt -manifest ${MQTTDLL_S}.manifest -outputresource: ${MQTTDLL_S}\; 2
```

```
# compiler
CC = cl
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/

# linker
LD = link
LINKFLAGS = /nologo /machine:x86 /manifest /dll
WINLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
          advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib\
          odbc32.lib odbccp32.lib ws2_32.lib
IMP = /implib: ${@:.dll=.lib}
LIBPDB = /pdb: ${@:.dll=.pdb}
LIBMAP = /map: ${@:.dll=.map}
WINLIBS_S = ${WINLIBS} crypt32.lib ssleay32.lib libeay32.lib
LIBPATH_S = /LIBPATH:${OPENSSL_DIR}/lib

# targets
all: ${MQTTDLL} ${MQTTDLL_A} ${MQTTDLL_AS} ${MQTTDLL_S}

${MQTTDLL}: ${SOURCE_FILES}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL}
    ${MANIFEST}

${MQTTDLL_A}: ${SOURCE_FILES_A}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS} ${CFLAGS} ${INC} /Fo ${SOURCE_FILES_A}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS} *.obj /out: ${MQTTDLL_A}
    ${MANIFEST_A}

${MQTTDLL_S}: ${SOURCE_FILES_S}
    -mkdir windows_ia32
    -rm ${CURDIR}/MQTTAsync.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
{MQTTDLL_S}
    ${MANIFEST_S}

${MQTTDLL_AS}: ${SOURCE_FILES_AS}
    -rm ${CURDIR}/MQTTClient.obj
    ${CC} ${CPPFLAGS_S} ${CFLAGS} ${INC_S} /Fo ${SOURCE_FILES_AS}
    ${LD} ${LINKFLAGS} ${IMP} ${LIBPDB} ${LIBMAP} ${WINLIBS_S} ${LIBPATH_S} *.obj /out: $
(MQTTDLL_AS}
    $(MANIFEST_AS}

.PHONY: limpiar
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

Compilación del paquete de OpenSSL

Cree el paquete OpenSSL antes de crear las bibliotecas de cliente MQTT seguras para C, `mqttv3cs` y `mqttv3as`. La compilación crea las bibliotecas necesarias para compilar una versión segura de la biblioteca de cliente MQTT para C y la herramienta de gestión de certificados de OpenSSL.

Antes de empezar

1. iOS La personalización del archivo make de iOS es para dispositivos de destino que se ejecuten en iOS6. La personalización puede ser distinta para las versiones anteriores y las posteriores de iOS.
2. Windows La personalización del archivo make de Windows es para Windows de 32 bits.

Acerca de esta tarea

Descargue e instale el paquete de OpenSSL y el software de requisito previo. Personalice los archivos make OpenSSL y cree bibliotecas de OpenSSL para su plataforma de destino. En Windows y Linux, make compila también la herramienta de creación y gestión de claves de OpenSSL.

Procedimiento

1. Instale el paquete de OpenSSL.
   a) Descargue el paquete de OpenSSL de OpenSSL

      Importante: La descarga y redistribución del paquete de OpenSSL está sujeta a una estricta regulación de importación y exportación y a condiciones de licencia de código abierto. Lea atentamente las restricciones y advertencias antes de decidir descargar el paquete.

   b) Expanda el contenido del archivo comprimido en *sdkroot*.

      Busque en la pestaña **Novedades** del sitio de OpenSSL la ubicación de descarga del último paquete. El paquete está comprimido como archivo tar, con la extensión `tar.gz`. Cuando se expande, el paquete crea una carpeta de nivel superior `openssl`*version*; por ejemplo, *openssl-1.0.1c*. Los ejemplos hacen referencia a la vía de acceso a la carpeta como %openssl% en Windows y $openssl en iOS; por ejemplo, en Windows, %openssl% es *sdkroot*\openssl-1.0.1c.

      Consejo: Compruebe la vía de acceso de directorios que se crea al extraer el paquete OpenSSL. Algunos paquetes tienen niveles duplicados de la carpeta `openssl`*version* .

2. Windows

   Opcional: En Windows, descargue e instale perl. Consulte perl.org.

   Para el ejemplo, perl se ha descargado de Descargas de ActivePerl.

3. iOS

   Opcional: En iOS, cree tres directorios más.

   ```
   $ssarm7 = $openssl/arm7
   $sslarm7s = $openssl/arm7s
   $ssli386 = $openssl/i386
   ```

   Para iOS debe compilar el paquete de OpenSSL para tres plataformas de hardware distintas.

4. Genere el archivo make de OpenSSL para compilar el paquete de OpenSSL para su hardware y su sistema operativo.
   a) Abra una ventana de mandatos en el directorio %openssl% o $openssl.
   b) Ejecute el mandato de perl **Configure** con los parámetros adecuados.
      - Windows

        ```
        perl Configure VC-WIN32 enable-capieng no-asm no-idea no-mdc2 no-rc5 --prefix=%openssl%
        ms\do_ms.bat
        ```

      - iOS

        ```
        ./Configure BSD-generic32 no-idea no-mdc2 no-rc5 --prefix=$openssl
        ```

5. iOS

   En iOS, personalice el archivo make de OpenSSL generado para distintos dispositivos Apple.
   a) Haga tres copias del archivo make generado, $openssl/Makefile

   ```
   $openssl/Makefile_armv7
   $openssl/Makefile_armv7s
   $openssl/Makefile_i386
   ```

b) Cambie la sentencia "CC=gcc" en cada uno de los archivos make.

La sentencia CC=gcc se encuentra aproximadamente por la línea 62. Cámbiela por los mandatos siguientes:

$openssl/Makefile_armv7

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7
```

$openssl/Makefile_armv7s

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch armv7s
```

$openssl/Makefile_i386

```
CC=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/usr/bin/gcc -arch i386
```

c) Cambie la sentencia "CFLAG=..." en cada uno de los archivos make.

Esta sentencia se encuentra aproximadamente por la línea 63 (partida en tres líneas para una mejor legibilidad):

```
CFLAG= -DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

La ubicación que se muestra para los SDK depende de sus opciones de instalación de Xcode. La versión de los SDK depende del nivel de sistema operativo para el que esté compilando el archivo make.

Simulador de iPhone

El archivo make simulador de iPhone es $openssl/Makefile_i386.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.0.sdk
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

iOS

Los archivos make de iOS son $openssl/Makefile_arm7 y $openssl/Makefile_arm7s.

```
CFLAG= -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneOS.platform/Developer/SDKs/iPhoneOS6.0.sdk
-DOPENSSL_THREADS -pthread -D_THREAD_SAFE
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DTERMIOS
-O3 -fomit-frame-pointer -Wall
```

6. Ejecute el archivo make generado.

- Windows

```
nmake -clean
nmake -f ms\nt.mak
nmake -f ms\nt.mak install
```

- iOS En iOS:

```
make clean
make -f $openssl/Makefile_arm7
mv $openssl/libcrypto.a $ssarm7/libcrypto.a
mv $openssl/libssl.a $ssarm7/libssl.a
make clean
make -f $openssl/Makefile_arm7s
mv $openssl/libcrypto.a $sslarm7s/libcrypto.a
mv $openssl/libssl.a $sslarm7s/libssl.a
make clean
```

```
make -f $openssl/Makefile_i386
mv $openssl/libcrypto.a $ssli386/libcrypto.a
mv $openssl/libssl.a $ssli386/libssl.a
```

Resultados

La compilación genera las bibliotecas compartidas, archivos lib y archivos de cabecera necesarios para compilar versiones seguras de la biblioteca de cliente MQTT para C.

Iniciación al cliente MQTT para C en iOS

Información sobre cómo hacer que las aplicaciones iOS intercambien mensajes con un servidor MQTT. Para utilizar en dispositivos iOS (es decir, iPhone y iPad), debe compilar la biblioteca de cliente MQTT para C con el código fuente que se proporcionar como parte del kit de desarrollo de software de MQTT.

Antes de empezar

1. Enlace a iOS Dev Center y sepa cómo desarrollar aplicaciones para iOS.
2. Obtenga un Mac de Apple con OS X 10.8.2, o posterior para ejecutar el entorno de desarrollo integrado (IDE) de Xcode.
3. (Opcional) Configure un entorno de desarrollo de C en Windows o Linux. Resulta útil compilar y ejecutar las aplicaciones C de ejemplo de cliente MQTT en Windows o Linux antes de desarrollar una aplicación MQTT iOS. También puede estudiar el código fuente de ejemplo sin compilar los ejemplos.
4. Para saber las plataformas soportadas y de referencia de cliente MQTT para C, consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.
5. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .

Acerca de esta tarea

El procedimiento le guía a través de los pasos siguientes:

1. Aprender a programar para MQTT estudiando, compilando y ejecutando las aplicaciones de cliente MQTT de ejemplo y las bibliotecas de cliente MQTT para C.
2. Instalar el entorno de desarrollo de Xcode para iOS en Apple Mac.
3. Realizar la tarea “Creación de las bibliotecas de cliente MQTT para C” en la página 31 para compilar las bibliotecas de cliente MQTT para C para dispositivos iOS.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 . Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte “Iniciación a los servidores MQTT” en la página 139.
2. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.
   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre MQTT. La vía de acceso a esta carpeta se denomina aquí *sdkroot*.
   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *sdkroot*. La expansión crea un árbol de directorios que empieza en *sdkroot*\SDK.
3. Opcional: Familiarizarse con la API de MQTT estudiando la Aplicación C de ejemplo de cliente MQTT.

a) Compilar la aplicación C de ejemplo de cliente MQTT, `MQTTV3sample.c`, para Windows o Linux. Consulte "Iniciación al cliente MQTT para C" en la página 26.

b) Conectar a un servidor MQTT version 3 y suscribirse y publicar en temas en el servidor.

c) Estudie el código fuente y la documentación de la API de MQTT . Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.

Aprender a crear y reanudar clientes MQTT y a suscribirse y publicar en temas MQTT a través del estudio del ejemplo síncrono. El ejemplo síncrono es más sencillo que el asíncrono. Si no ha programado MQTT antes, puede escribir un programa MQTT síncrono para familiarizarse con el modelo de programación y la API de MQTT.

d) Compilar la biblioteca de cliente MQTT para C en Windows o Linux. Consulte "Creación de las bibliotecas de cliente MQTT para C" en la página 31.

e) Compilar y ejecutar la aplicación C de cliente MQTT de publicación y suscripción de ejemplo.

f) Estudiar el código de la aplicación C asíncrona de cliente MQTT y la documentación de referencia de MQTT.

Debe utilizar la interfaz asíncrona para escribir una aplicación MQTT para un dispositivo móvil. Las aplicaciones bien escritas que llaman a la interfaz asíncrona responden mejor y alargan más la vida de la batería que las aplicaciones escritas para la interfaz síncrona.

La interfaz asíncrona tiene dos grados de asincronicidad.

i) El primer grado consiste en desbloquear la aplicación mientras la biblioteca de cliente MQTT espera las publicaciones del servidor.

ii) El segundo grado consiste en desbloquear la aplicación mientras la biblioteca de cliente se conecta al servidor, crea suscripciones y envía publicaciones.

4. Descargue e instale las herramientas de desarrollo de iOS .
   a. Inicie sesión con un ID de usuario que tenga privilegios de administración.
   b. Compruebe que su Apple Mac tiene la versión 10.8.2 o posterior.
   c. Vaya al sitio web Xcode para descargar Xcode desde el App Store de Mac.
   d. Instale Xcode, el entorno de línea de mandatos y el simulador.

   Si AppStore de Mac ofrece diversas versiones del simulador, elija la versión que sea compatible con el nivel de iOS que tiene previsto para su aplicación.
5. Compilar las bibliotecas de cliente MQTT para C en iOS. Consulte "Creación de las bibliotecas de cliente MQTT para C" en la página 31.

Qué hacer a continuación

1. Verifique las bibliotecas de cliente MQTT para C que ha compilado:
   a. Utilice el entorno de desarrollo de Xcode para compilar el Aplicación C de ejemplo de cliente MQTT asíncrono, y enlazar a la biblioteca de cliente MQTT para C asíncrono.
   b. Utilice el entorno de desarrollo de Xcode para ejecutar la aplicación C de ejemplo de cliente MQTT en un dispositivo iOS. Conecte el ejemplo al servidor MQTT version 3 que ha configurado, consulte "Configuración del servicio de MQTT desde la línea de mandatos" en la página 143.
2. Cree una aplicación C de cliente MQTT para iOS. Pueden serle útiles los ejemplos de codificación de ka aplicación C asíncrona. Los ejemplos son `MQTTV3ASample.c` y `MQTTV3ASSample.c` en *sdkroot*`\SDK\clients\c\samples`. Como ejercicio, empiece implementando el ejemplo de MQTT de publicación/suscripción.

   Consejo: Para hacerse una idea de cómo puede ser la aplicación y de lo que puede hacer, mire las captura de pantalla de la Aplicación C de ejemplo de cliente MQTT. Consulte "Iniciación a Cliente MQTT para Java en Android" en la página 18.

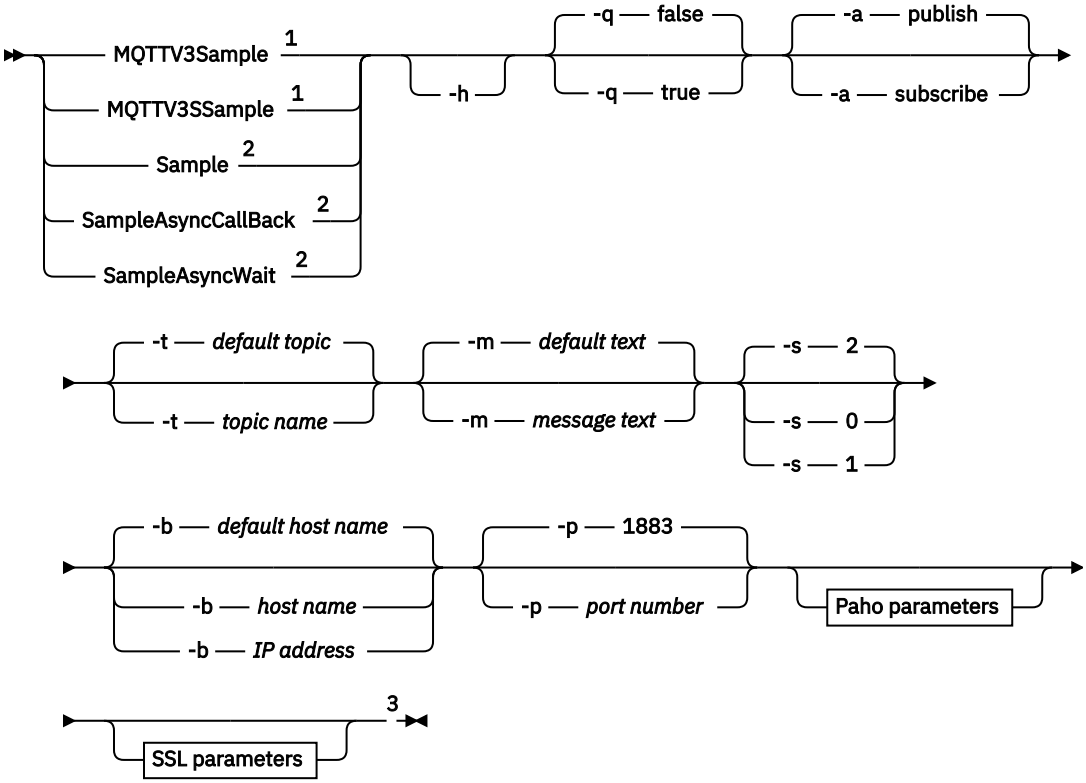
Programas MQTT de ejemplo de línea de mandatos

sintaxis y parámetros de los programas MQTT de ejemplo de línea de mandatos.

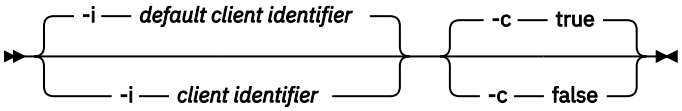
Finalidad

Publicar y suscribirse a un tema.

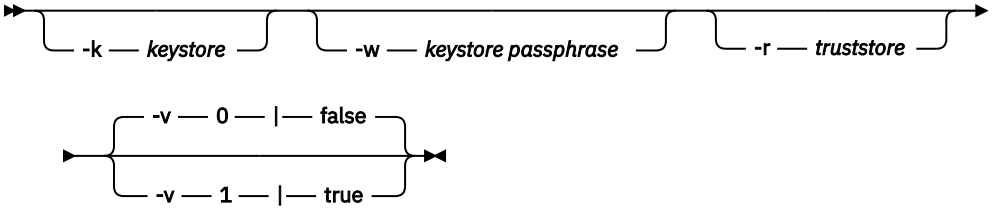
Syntax



Paho parameters



SSL parameters



Notas:

[1] IBM WebSphere MQ sample
[2] Paho sample
[3] *Not* `MQTTV3Sample`.

Parámetros

-h
Imprimir el texto de ayuda y salir

-q
Establecer la modalidad silenciosa (-q), en lugar de utilizar la modalidad predeterminada, que es false.

-a publish|subscribe
Establezca la acción en `publish` o `subscribe`, en lugar de asumir la acción predeterminada de publicación.

-t *nombre de tema*
Publique o suscríbase a *`topic name`*, en lugar de publicar o suscribirse al tema predeterminado. Los valor predeterminados son los siguientes:

Ejemplos de Paho

Publicar
`Sample/Java/v3`

Suscribir
`Sample/#`

Ejemplos de IBM WebSphere MQ

Publicar
`MQTTV3Sample/Java/v3` o `MQTTV3Sample/C/v3`

Suscribir
`MQTTV3Sample/#`

-m *texto del mensaje*
Publique *`message text`* en lugar de enviar el texto predeterminado. El texto predeterminado es `"Message from MQTTv3 C client"`o `"Message from MQTTv3 Java client"`

-s 0|1|2
Defina la calidad de servicio (QoS) en lugar de utilizar la predeterminada QoS, 2.

-b *nombre de host*
Conéctese a *`host name`* o a la dirección IP en lugar de conectarse al nombre de host predeterminado. El nombre de host predeterminado para los ejemplos de Paho es `m2m.eclipse.org`. Para los ejemplos de IBM WebSphere MQ es `localhost`.

-p *número de puerto*
Utilice el puerto *`port number`* en lugar de utilizar el puerto predeterminado, 1883.

Parámetros Paho

-i *identificador de cliente*
Establezca el identificador de cliente en *`client identifier.`* El identificador de cliente predeterminado es `SampleJavaV3_"+action`, donde `action` es `publish` o `subscribe`.

-c true|false
Establecer el distintivo de sesión limpia. El valor predeterminado es `true`: las suscripciones son no duraderas.

Parámetros de SSL

-k *almacén de claves*
Establezca la vía de acceso al almacén de claves que contiene la clave privada que identifica el cliente en *`keystore`*. Para los ejemplos de C, el almacén es un archivo PEM (Privacy-Enhanced Mail). Para los ejemplos de Java, se trata de un almacén de claves Java (JKS).

-w *frase de contraseña del almacén de claves*
Establezca la frase de contraseña para autorizar al cliente a acceder al almacén de claves en *`keystore passphrase`*.

-r ***almacén de confianza***
Establezca la vía de acceso al almacén de claves que contiene las claves públicas de los servidores MQTT en los que confía el cliente en *truststore*. El almacén de claves es un archivo PEM (Privacy-Enhanced Mail). Para los ejemplos de C, el almacén es un archivo PEM (Privacy-Enhanced Mail). Para los ejemplos de Java, se trata de un almacén de claves Java (JKS).

-v **0|false|1|true**
Establecer la opción verificar a 1|`true` para exigir un certificado de servidor. El valor predeterminado es 0|`false`: el certificado de servidor no está seleccionado. El canal SSL siempre está cifrado.

Establezca la opción en 0|1 para programas C y `true`|`false` para programas Java .

Tareas relacionadas

"Iniciación al cliente MQTT para Java" en la página 12
Empezar a trabajar con el cliente MQTT para aplicaciones de ejemplo de Java, utilizando IBM MessageSight o IBM WebSphere MQ como servidor de MQTT. Las aplicaciones de ejemplo utilizan una biblioteca de cliente del kit de desarrollo de software (SDK) de MQTT de IBM. La aplicación de ejemplo `SampleAsyncCallBack` es un modelo para escribir aplicaciones MQTT para Android y otros sistemas operativos controlados por sucesos.

"Iniciación al cliente MQTT para C" en la página 26
Puede empezar a trabajar con el ejemplo de cliente MQTT seguro para C en cualquier plataforma en la que pueda compilar el código C fuente. Verifique que puede ejecutar el cliente MQTT de ejemplo para C con IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

"Creación de las bibliotecas de cliente MQTT para C" en la página 31
Siga estos pasos para crear las bibliotecas de cliente MQTT para C. En el tema se incluyen los conmutadores de compilación y enlace para diversas plataformas, y ejemplos de cómo crear las bibliotecas en iOS y Windows.

Seguridad de MQTT

Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La identidad consiste en dar nombre al cliente que se va a autorizar y dar autorización. La autenticación consiste en probar la identidad del cliente y la autorización consiste en gestionar los derechos que se otorgan al cliente.

Pruebe los ejemplos de seguridad

- "Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro" en la página 56
- "Conexión de la aplicación Java de ejemplo de cliente MQTT en Android a través de SSL" en la página 64
- "Autenticación de una aplicación Java de cliente MQTT con JAAS" en la página 74
- V 7.5.0.1 "Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets" en la página 79
- "Creación y ejecución de Aplicación C de ejemplo de cliente MQTT seguro" en la página 87

Identidad

Puede identificar un cliente MQTT por su identificador de cliente, su ID de usuario o su certificado digital público. Uno u otro de estos atributos definen la identidad del cliente. Un servidor MQTT autentica el certificado que envía el cliente con el protocolo SSL, o la identidad del cliente con una contraseña definida por el cliente. El servidor controla los recursos a los que puede acceder el cliente, en base a la identidad del mismo.

El servidor MQTT se identifica ante el cliente con su dirección IP y su certificado digital. El cliente MQTT utiliza el protocolo SSL para autenticar el certificado enviado por el servidor. En algunos casos, utiliza el nombre DNS del servidor para verificar que el servidor que ha enviado el certificado está registrado como el poseedor del certificado.

Defina la identidad del cliente de una de las siguientes formas:

Identificador de cliente
La clase `MqttClient` (`MQTTClient_create` o `MQTTAsync_create` en C) define el identificador de cliente. Llame al constructor de clase para establecer el identificador de cliente como parámetro, o devolver un identificador de cliente generado aleatoriamente.El identificador de cliente debe ser exclusivo entre todos los clientes que se conectan al servidor y no debe ser el mismo que el nombre del gestor de colas en el servidor. Todos los clientes deben tener un identificador de cliente, incluso si no se utiliza para la comprobación de identidad. Consulte “Identificador de cliente” en la página 130.

ID de usuario
La clase `MqttClient` (`MQTTClient_create` o `MQTTAsync_create` en C) define el ID de usuario del cliente como un atributo de `MqttConnectOptions` (`MqttClient_ConnectOptions` en C). No es necesario que el ID de usuario sea exclusivo de cada cliente.

Certificado digital del cliente
El certificado digital del cliente se almacena en el almacén de claves del cliente. La ubicación del almacén de claves depende del cliente:

- **Java**

  Defina la ubicación y las propiedades del almacén de claves del cliente llamando al método `setSSLProperties` de `MqttConnectOptions` y pasando las propiedades del almacén de claves. Consulte Modificaciones SSL a Example.java. La herramienta **`keytool`** gestiona claves y almacenes de claves Java.

- **C**

  `MQTTClient_create` o `MQTTAsync_create` definen las propiedades del almacén de claves como atributos de `MQTTClient_SSLOptions ssl_opts`. La herramienta **openSSL** crea y gestiona las claves y almacenes de claves a los que accede el cliente de MQTT para C.

- **Android**

  Puede gestionar el almacén de claves de un dispositivo Android desde el menú **Ajustes** > **Seguridad**. Cargue los nuevos certificados desde la tarjeta SD.

Defina la identidad del servidor almacenando su clave privada en el almacén de claves del servidor:

IBM WebSphere MQ
El almacén de claves del servidor MQTT es un atributo del canal de telemetría al que está conectado el cliente.

Establezca la ubicación del almacén de claves y los atributos con IBM WebSphere MQ Explorer, o con el mandato **`DEFINE CHANNEL`** ; consulte DEFINE CHANNEL (MQTT). Varios canales pueden compartir un almacén de claves.

Autenticación

Un cliente MQTT puede autenticar el servidor MQTT al que se conecta y, a su vez, el servidor puede autenticar el cliente que se conecta a él.

Un cliente autentica un servidor con el protocolo SSL. Un servidor MQTT autenticar un cliente con el protocolo SSL o con la contraseña, o con ambas cosas.

Si el cliente autentica el servidor pero el servidor no autentica el cliente,el cliente se suele conocer como 'cliente anónimo'. Frecuentemente, al establecer una conexión de cliente anónimo por SSL, se autentica el cliente con una contraseña cifrada por la sesión SSL. Es mucho más frecuente autenticar un cliente con una contraseña que con un certificado de cliente, debido al problema de distribución y gestión de certificados. Es mucho más frecuente que se utilicen certificados de cliente en dispositivos de alto valor tales como máquinas para tarjetas de crédito y de chip, y en dispositivos personalizados como medidores inteligentes de electricidad.

Autenticación del servidor por parte de un cliente

Un cliente MQTT verifica que está conectado al servidor correcto autenticando el certificado de servidor con el protocolo SSL. Esta forma de verificación la utiliza a menudo al navegar a un sitio web con el protocolo HTTPS.

El servidor envía su certificado público, firmado por una entidad emisora de certificados, al cliente. El cliente utiliza la clave pública de la entidad emisora de certificados para verificar la firma de la entidad emisora de certificados en el certificado del servidor. También comprueba que el certificado es actual. Estas comprobaciones establecen que el certificado es válido.

Los certificados de autoridad emisora de certificados, a menudo denominados certificados raíz, se guardan en el almacén de confianza del cliente.

- **Java**

  Llame al método `setSSLProperties` de `MqttConnectOptions`, vaya a las propiedades del almacén de confianza y defina la ubicación y las propiedades del almacén de confianza del cliente. Consulte Modificaciones SSL a Example.java. Puede gestionar los certificados y los almacenes de confianza con la herramienta **keytool**.

- **C**

  `MQTTClient_create` o `MQTTAsync_create` definen las propiedades del almacén de confianza como atributos de `MQTTClient_SSLOptions ssl_opts`. Puede gestionar los certificados y los almacenes de confianza con la herramienta **openSSL**.

- **Android**

  Puede gestionar el almacén de confianza de un dispositivo Android desde el menú **Ajustes** > **Seguridad**. Cargue los nuevos certificados raíz desde la tarjeta SD.

Autenticación de cliente por parte de un servidor

Un servidor MQTT verifica que está conectado al cliente adecuado autenticando el certificado de cliente con el protocolo SSL o autenticando la identidad del cliente con una contraseña.

Autentica el cliente con la contraseña que envía el cliente al servidor en una cabecera MQTT protocol. El servidor puede elegir autenticar el identificador de cliente, el ID de usuario o el certificado con la contraseña. Depende del servidor. Generalmente, el servidor autentica el ID de usuario. Verifique las contraseñas a través de una conexión SSL que se haya protegido verificando el servidor, para evitar enviar contraseñas porque sí.

- **IBM WebSphere MQ**

  IBM WebSphere MQ autentica un certificado de cliente con el protocolo SSL. Guarda los certificados raíz en el almacén de claves IBM WebSphere MQ Telemetry. Sólo puede autenticar un certificado de cliente como parte de una autenticación SSL mutua. Es decir, debe proporciona al cliente el certificado público del servidor además de proporcionar al servidor el certificado público del cliente.

  IBM WebSphere MQ Telemetry utiliza el mismo almacén para sus propios certificados público y privado y para otros certificados públicos, como por ejemplo los certificados raíz proporcionados por las entidades emisoras de certificados.

  Establezca la ubicación del almacén de claves y los atributos con IBM WebSphere MQ Explorer, o con el mandato **DEFINE CHANNEL** ; consulte DEFINE CHANNEL (MQTT). Varios canales pueden compartir un almacén de claves.

  IBM WebSphere MQ autentica el ID de usuario cliente o el identificador de cliente llamando al servicio de autenticación y autorización de Java (JAAS).

  Configure JAAS en una stanza de configuración `MQXRConfig` almacenada en el archivo `jaas.config`. El archivo se almacena en el directorio `qmgrs\`*`QmgrName`*`\mqxr` en la vía de acceso a datos de IBM WebSphere MQ.

  Compruebe la autenticidad del cliente escribiendo un método `login` para el módulo `JAASLoginModule`. Consulte “Configuración JAAS del canal de telemetría” en la página 117.

IBM WebSphere MQ Telemetry pasa al método `JAASLoginModule.login` los parámetros siguientes:

- ID de usuario
- Contraseña
- Identificador de cliente
- Identificador de red
- Nombre de canal
- ValidPrompts

Autorización

La autorización no forma parte de MQTT protocol. La proporcionan los servidores MQTT. LO que se autoriza depende de lo que hace el servidor. Los servidores MQTT son intermediarios de publicación/suscripción, y unas útiles reglas de autorización de MQTT controlan los clientes que se pueden conectar al servidor y los temas en los que un cliente puede publicar o suscribirse. Si un cliente MQTT puede administrar el servidor, hay otras reglas de autorización que controlan los clientes que pueden administrar los distintos aspectos del servidor.

El número de posibles clientes es enorme, por lo que no es factible autorizar cada cliente por separado. Un servidor MQTT tendrá los medios para agrupar a los clientes por perfiles o grupos.

La identidad del cliente, desde el punto de vista del acceso y la autorización, no es exclusiva de un cliente MQTT. No confunda la identidad de un cliente con el identificador de cliente. Pueden ser lo mismo, pero normalmente son distintos. Por ejemplo, es posible que tenga un mismo nombre de usuario para diversos servicios y algunos de estos servicios pueden cooperar en "inicio de sesión único". Es posible que un servidor MQTT a escala de empresa llame a un servicio de autorización que ofrezca identidades y autoridades comunes para aplicaciones distintas.

IBM WebSphere MQ

IBM WebSphere MQ tiene un servicio de autorización al que es posible conectarse. El servicio de autorización predeterminado que se proporciona en Windows y Linux es el gestor de autorizaciones sobre objetos (OAM). Consulte Control del acceso a objetos utilizando el OAM en sistemas UNIX, Linux y Windows. Asocia los ID y grupos de usuarios del sistema operativo a operaciones en objetos IBM WebSphere MQ, tales como, por ejemplos, temas y colas.

Puede configurar un canal de telemetría para acceder a IBM WebSphere MQ con un ID de usuario fijo. Ésta es la forma en que está configurado el canal de ejemplo. O puede acceder a IBM WebSphere MQ con el ID de usuario definido por el cliente MQTT. En Autorización de clientes MQTT para acceder a objetos de WebSphere MQ se describen formas de configurar IBM WebSphere MQ Telemetry para conseguir un control de accesos de clientes de granularidad baja, media o alta.

Tareas relacionadas

“Creación y ejecución de Aplicación C de ejemplo de cliente MQTT seguro” en la página 87
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura C de ejemplo en cualquier sistema operativo en el que pueda compilar el código C fuente. Verifique que puede ejecutar la aplicación C de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

“Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro” en la página 56
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura Java de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java"

“Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets” en la página 79
Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

“Conexión de la aplicación Java de ejemplo de cliente MQTT en Android a través de SSL” en la página 64

Ponerse en marcha con el cliente de Android MQTT de ejemplo conectado a IBM WebSphere MQ a través de SSL.

"Autenticación de una aplicación Java de cliente MQTT con JAAS" en la página 74
Información sobre cómo autenticar un cliente con JAAS. Complete los pasos de esta tarea para modificar el programa de ejemplo `JAASLoginModule.java` y configurar IBM WebSphere MQ para autenticar una aplicación Java de cliente MQTT con JAAS.

Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro

Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura Java de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java"

Antes de empezar

1. Debe tener acceso a un servidor MQTT version 3.1 que admita MQTT protocol sobre SSL.
2. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
3. Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java". Consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.
4. Los canales SSL deben estar iniciados.

Acerca de esta tarea

A modo de ilustración, en este artículo se muestra cómo compilar y ejecutar el Aplicación Java de ejemplo de cliente MQTT seguro en Windows desde la línea de mandatos.

Proteja el canal SSL con claves firmadas por una entidad emisora de certificados o bien con claves autofirmadas.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 a través de SSL. Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte "Iniciación a los servidores MQTT" en la página 139.
2. Opcional: Instale un kit de desarrollo de Java (JDK) en la versión 7 o posterior.

   Versión 7 es necesario para ejecutar el mandato **keytool** para certificar certificados. Si no va a certificar certificados no necesita la versión 7 de JDK.
3. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.

   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre `MQTT`. La vía de acceso a esta carpeta se denomina aquí *sdkroot*.

   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *sdkroot*. La expansión crea un árbol de directorios que empieza en *sdkroot*\SDK.
4. Cree y ejecute los scripts para generar pares de claves y certificados, y configure IBM WebSphere MQ como el servidor de MQTT .

   Siga los pasos de "Generación de claves y certificados" en la página 97 para crear y ejecutar los scripts. Los scripts se enumeran también en "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 58.

5. Compruebe que los canales SSL están en ejecución y configurados adecuadamente.

   En IBM WebSphere MQ, escriba el mandato siguiente en una ventana de mandatos:

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```

6. Cree los scripts para compilar y ejecutar el Aplicación Java de ejemplo de cliente MQTT seguro.

   a) Cree y ejecute `ssjavaclient.bat` para probar un canal SSL que está protegido con certificados autofirmados.

   b) Cree y ejecute `cajavaclient.bat` para probar el canal SSL que está protegido con certificados de entidad emisora de certificados.

Scripts para ejecutar el cliente MQTT de Java seguro

Ejecute los scripts de "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 58 antes de ejecutar estos scripts.

Cliente MQTT de Java seguro con certificados autofirmados.

Ejecute este script con los certificados autofirmados que ha creado ejecutando el script `sscerts.bat`.

```
@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltsrvjkstruststore% -v true
pause
endlocal
```

Figura 14. `ssjavaclient.bat`

Ejecute el cliente MQTT seguro de Java con certificados firmados por entidad emisora de certificados.

Ejecute el script con los certificados firmados por entidad emisora de certificado que ha creado ejecutando el script `cacerts.bat`.

```
@echo off
setlocal
cd %jsamppath%
set classpath=
set JAVADIR=C:\Program Files\IBM\Java70\bin
cd %
"%JAVADIR%\javac"
-cp ..\org.eclipse.paho.client.mqttv3.jar .\org\eclipse\paho\sample\mqttv3app\Sample.java
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportopt% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportopt% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
ping -n 2 127.0.0.1 > NUL 2>&1
start "Sample Subscriber" "%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -a subscribe -b %host% -p %sslportreq% -k
%cltjkskeystore% -w %cltjkskeystorepass% -r %cltcajkstruststore% -v true
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
"%JAVADIR%\java" -cp  .;..\org.eclipse.paho.client.mqttv3.jar
org.eclipse.paho.sample.mqttv3app.Sample -b %host% -p %sslportreq% -k %cltjkskeystore% -w
%cltjkskeystorepass% -r %cltcajkstruststore% -v true
pause
endlocal
```

Figura 15. `cajavaclient.bat`

Conceptos relacionados

“Seguridad de MQTT” en la página 52
Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La identidad consiste en dar nombre al cliente que se va a autorizar y dar autorización. La autenticación consiste en probar la identidad del cliente y la autorización consiste en gestionar los derechos que se otorgan al cliente.

Tareas relacionadas

“Generación de claves y certificados” en la página 97
Siga este procedimiento para generar claves y certificados para clientes Java y C, incluyendo las aplicaciones de Android y iOS, y los servidores de IBM WebSphere MQ y IBM MessageSight.

“Conexión de la aplicación Java de ejemplo de cliente MQTT en Android a través de SSL” en la página 64
Ponerse en marcha con el cliente de Android MQTT de ejemplo conectado a IBM WebSphere MQ a través de SSL.

“Autenticación de una aplicación Java de cliente MQTT con JAAS” en la página 74
Información sobre cómo autenticar un cliente con JAAS. Complete los pasos de esta tarea para modificar el programa de ejemplo `JAASLoginModule.java` y configurar IBM WebSphere MQ para autenticar una aplicación Java de cliente MQTT con JAAS.

Scripts de ejemplo para configurar certificados SSL para Windows

** **

Estos archivos de mandato de ejemplo crean los certificados y los almacenes de certificados según se describe en los diversos pasos de la tarea. Además el ejemplo configura el gestor de colas de cliente MQTT para que utilice el almacén de certificados del servidor. El ejemplo suprime y vuelve a crear el gestor de colas llamando al script `SampleMQM.bat` que se proporciona en IBM WebSphere MQ.

initcert.bat

initcert.bat define los nombres y vías de acceso a los certificados y otros parámetros que requieren los mandatos **keytool** y **openSSL**. Los valores se describen en los comentarios dentro del script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
```

```
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

Los mandatos del script `cleancert.bat` suprimen el gestor de colas del cliente MQTT para asegurarse de que el almacén de certificados de servidor no está bloqueado y después suprimen todos los almacenes de claves y los certificados que han creado los scripts de seguridad de ejemplo.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

Los mandatos del script `genkeys.bat` crean pares de claves para su entidad emisora de certificados privada, el servidor y un cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

Los mandatos del script `sscerts.bat` exportan los certificados autofirmados de cliente y servidor de sus almacenes de claves e importan el certificado de servidor en el almacén de confianza del cliente, y el certificado de cliente en el almacén de claves del servidor. El servidor no tiene un almacén de confianza. Los mandatos crean un almacén de confianza de cliente en formato PEM a partir del almacén de confianza JKS del cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
```

```
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Este script importa el certificado raíz de la entidad emisora de certificados en los almacenes de claves privados. El certificado raíz de CA es necesario para crear la cadena de claves entre el certificado raíz y el certificado firmado. El script `cacerts.bat` exporta las solicitudes de certificados de cliente y servidor de sus almacenes de claves. El script firma las solicitudes de certificados con la clave de la entidad emisora de certificados privada en el almacén de claves `cajkskeystore.jks`, y después importa los certificados firmados otra vez en los mismos almacenes de claves de donde procedían. La importación crea la cadena de certificados con el certificado raíz de CA. El script crea un almacén de confianza del cliente en formato PEM a partir del almacén de claves del cliente en formato JKS.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

El script enumera los almacenes de claves y los certificados en el directorio de certificados. A continuación, crea el gestor de colas de ejemplo de MQTT y configura los canales de telemetría seguros.

```
@echo ________________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V 7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Conexión de la aplicación Java de ejemplo de cliente MQTT en Android a través de SSL

Ponerse en marcha con el cliente de Android MQTT de ejemplo conectado a IBM WebSphere MQ a través de SSL.

Antes de empezar

En este artículo se presupone que utiliza Android con API de nivel 14 (ICS 4.0), por lo menos. Los niveles anteriores disponían de un almacén de claves, pero sólo podían acceder al mismo las aplicaciones del sistema.

1. Debe tener acceso a un servidor MQTT version 3.1 que admita MQTT protocol sobre SSL.
2. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
3. Si prueba la conexión en un dispositivo Android anterior, puede que necesite una tarjeta SD para transferir el certificado al dispositivo.
4. Si prueba la conexión en un dispositivo Android virtual, configure una tarjeta SD para el dispositivo virtual.
5. Los canales SSL deben estar iniciados.

Acerca de esta tarea

Complete esta tarea para ejecutar el dispositivo Aplicación Java de ejemplo de cliente MQTT para Android sobre SSL. Una conexión SSL satisfactoria establece un canal cifrado seguro entre el dispositivo Android y el servidor MQTT. La identidad del servidor se autentica.

Con Android, puede autenticar el servidor con SSL. También puede autenticar el dispositivo, aunque la aplicación de ejemplo no lo hace. Para autenticar el dispositivo, utilice la API de KeyChain o bien utilice JAAS para autenticar el identificador de cliente, la dirección IP del cliente o el nombre de usuario y la contraseña proporcionados por la aplicación MQTT de Android.

Los certificados X.509 que instale en el almacén de confianza de Android deben estar firmados por una entidad emisora de certificados. En el ejemplo, se crea una entidad emisora de certificados que firma

el certificado que después se instala en el dispositivo Android. En los dispositivos Android hay diversos certificados raíz preinstalados.

Debe crear un bloqueo en su dispositivo Android antes de instalar un certificado de confianza. El bloqueo impide que alguien instale certificados en el dispositivo sin su conocimiento.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 a través de SSL. Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte "Iniciación a los servidores MQTT" en la página 139.
2. Ejecute la aplicación de cliente MQTT de ejemplo for Android "`MQTTExerciser`" en un canal MQTT no protegido. Consulte "Iniciación a Cliente MQTT para Java en Android" en la página 18.

   Vuelva a utilizar la aplicación para probar el canal seguro.

   Si ha iniciado un dispositivo Android virtual, déjelo en ejecución.
3. Opcional: Instale un kit de desarrollo de Java (JDK) en la versión 7 o posterior.

   Versión 7 es necesario para ejecutar el mandato **`keytool`** para certificar certificados. Si no va a certificar certificados no necesita la versión 7 de JDK.
4. Cree y ejecute los scripts para generar pares de claves y certificados, y configure IBM WebSphere MQ como el servidor de MQTT .

   Siga los pasos de "Generación de claves y certificados" en la página 97 para crear y ejecutar los scripts. Los scripts se enumeran también en "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 68.

   Necesita el certificado público de la entidad emisora de certificados y el almacén de claves del servidor. No necesita certificados de cliente ni ningún certificado en formato `.pem` ni `.p12`.
5. Compruebe que los canales SSL están en ejecución y configurados adecuadamente.

   En IBM WebSphere MQ, escriba el mandato siguiente en una ventana de mandatos:

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```
6. Instale el certificado de la certificado de autoridad emisora de certificados en el almacén de confianza de Android.

   El archivo de la entidad emisora de certificados, en el ejemplo, es `cacert.cer`.

   a) Cambie el nombre del certificado a `cacert.crt`

   b) Copie el certificado en el almacenamiento interno raíz o en la tarjeta SD.

      En el caso de un dispositivo virtual en ejecución, abra Eclipse o ejecute ADB (Android Debug Bridge) para copiar el certificado en el dispositivo virtual:

      Eclipse

      i) Ejecute Eclipse y abra la perspectiva DDMS.

      ii) En la vista principal, abra la ventana **Explorador de archivos**.

      iii) Abra el directorio `mnt/sdcard`.

      iv) Arrastre el archivo `cacert.crt` al directorio `mnt/sdcard`.

ADB

i) Abra una ventana de mandatos y establezca su directorio actual en `android-sdk\platform-tools` en el directorio de instalación android; por ejemplo, `C:\Program Files\Android\android-sdk\platform-tools`.

ii) Copie el certificado al directorio `mnt/sdcard`:

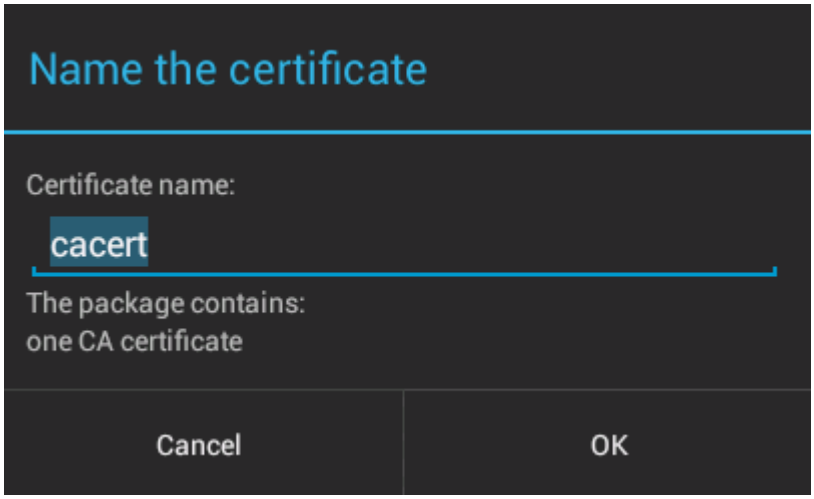
```
adb push %cacert% /mnt/sdcard/cacert.crt
```

7. Instale el certificado en el almacén de confianza de certificados del dispositivo Android.

   El certificado debe tener una cláusula de Restricciones básicas con el valor `Subject Type=CA`.

   a) Desbloquee el dispositivo y pulse el botón **widgets**.

   b) Pulse **Configuración** > **Seguridad** > **Almacenamiento de credenciales** > **Instalar desde tarjeta SD**.



   c) Confirme que el nombre de archivo del certificado es correcto y pulse **Aceptar**.

   Nota: Si no ha definido un bloqueo para el dispositivo, Android le solicitará ahora que lo haga.

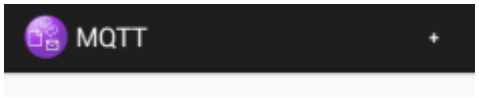
8. Confirme que el certificado está instalado en el dispositivo.

   a) Pulse **Credenciales de confianza** > **Usuario** y espere unos minutos para que el certificado se muestre en la lista de certificados de usuario.

9. Vuelva a ejecutar la aplicación `MQTTExerciser` y conéctela a un canal MQTT que haya configurado para clientes SSL anónimos.

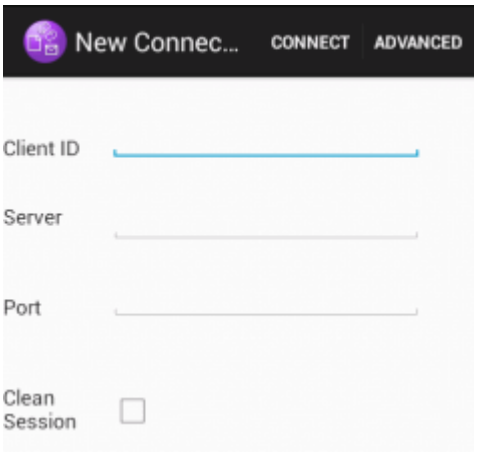
   a) Abra Aplicación Java de ejemplo de cliente MQTT para Android.

   Se abre esta ventana en su dispositivo Android:



b) Conectar con un servidor MQTT .

i) Pulse en el signo **+** para abrir una nueva conexión MQTT.



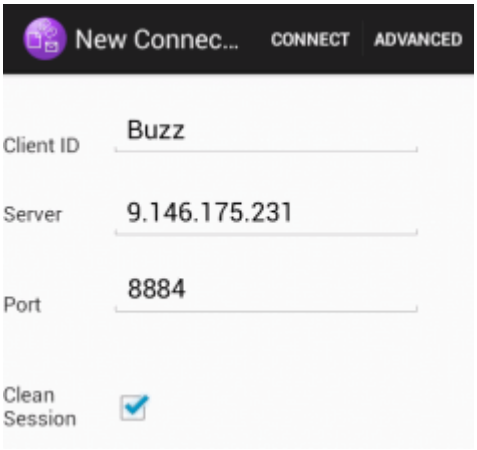
ii) Escriba cualquier identificador exclusivo en el campo **ID de cliente**. Tenga paciencia, puede que, al escribir, las pulsaciones de teclas sean lentas.

iii) En el campo **Servidor** escriba la dirección IP del servidor MQTT.

Éste es el servidor que ha elegido en el paso principal la primera vez. La dirección IP no puede ser `127.0.0.1`

iv) Escriba el número de puerto de la conexión MQTT.

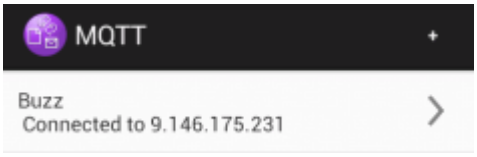
Defina el número de puerto a 8884, esto se hace mediante la variable `%sslportopt%` en los scripts de ejemplo. Este número de puerto es el número de puerto del canal MQTT que ha configurado para clientes SSL anónimos al ejecutar los scripts de ejemplo en el paso “4” en la página 65.



v) Pulse en la pestaña **Avanzados** y seleccione la opción **SSL**. Pulse **Guardar**.

vi) Pulse **Conectar**.

Si la conexión es satisfactoria, verá un mensaje "Conectando" seguido de esta ventana:



Resultados

Al conectarse a una conexión no segura, la aplicación `MQTTExerciser` tarda un poco más a conectarse y enviar mensajes, pero por lo demás funciona igual.

Conceptos relacionados

“Seguridad de MQTT” en la página 52
Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La identidad consiste en dar nombre al cliente que se va a autorizar y dar autorización. La autenticación consiste en probar la identidad del cliente y la autorización consiste en gestionar los derechos que se otorgan al cliente.

Tareas relacionadas

“Generación de claves y certificados” en la página 97
Siga este procedimiento para generar claves y certificados para clientes Java y C, incluyendo las aplicaciones de Android y iOS, y los servidores de IBM WebSphere MQ y IBM MessageSight.

“Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro” en la página 56
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura Java de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java"

“Autenticación de una aplicación Java de cliente MQTT con JAAS” en la página 74
Información sobre cómo autenticar un cliente con JAAS. Complete los pasos de esta tarea para modificar el programa de ejemplo `JAASLoginModule.java` y configurar IBM WebSphere MQ para autenticar una aplicación Java de cliente MQTT con JAAS.

Scripts de ejemplo para configurar certificados SSL para Windows

** **

Estos archivos de mandato de ejemplo crean los certificados y los almacenes de certificados según se describe en los diversos pasos de la tarea. Además el ejemplo configura el gestor de colas de cliente MQTT para que utilice el almacén de certificados del servidor. El ejemplo suprime y vuelve a crear el gestor de colas llamando al script `SampleMQM.bat` que se proporciona en IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` define los nombres y vías de acceso a los certificados y otros parámetros que requieren los mandatos **`keytool`** y **openSSL**. Los valores se describen en los comentarios dentro del script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
```

```
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

Los mandatos del script `cleancert.bat` suprimen el gestor de colas del cliente MQTT para asegurarse de que el almacén de certificados de servidor no está bloqueado y después suprimen todos los almacenes de claves y los certificados que han creado los scripts de seguridad de ejemplo.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

Los mandatos del script `genkeys.bat` crean pares de claves para su entidad emisora de certificados privada, el servidor y un cliente.

```
@rem
@echo ________________________________________________________________________________
```

```
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

Los mandatos del script `sscerts.bat` exportan los certificados autofirmados de cliente y servidor de sus almacenes de claves e importan el certificado de servidor en el almacén de confianza del cliente, y el certificado de cliente en el almacén de claves del servidor. El servidor no tiene un almacén de confianza. Los mandatos crean un almacén de confianza de cliente en formato PEM a partir del almacén de confianza JKS del cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
```

```
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Este script importa el certificado raíz de la entidad emisora de certificados en los almacenes de claves privados. El certificado raíz de CA es necesario para crear la cadena de claves entre el certificado raíz y el certificado firmado. El script `cacerts.bat` exporta las solicitudes de certificados de cliente y servidor de sus almacenes de claves. El script firma las solicitudes de certificados con la clave de la entidad emisora de certificados privada en el almacén de claves `cajkskeystore.jks`, y después importa los certificados firmados otra vez en los mismos almacenes de claves de donde procedían. La importación crea la cadena de certificados con el certificado raíz de CA. El script crea un almacén de confianza del cliente en formato PEM a partir del almacén de claves del cliente en formato JKS.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
```

```
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

El script enumera los almacenes de claves y los certificados en el directorio de certificados. A continuación, crea el gestor de colas de ejemplo de MQTT y configura los canales de telemetría seguros.

```
@echo ____________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

V7.5.0.1

```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Autenticación de una aplicación Java de cliente MQTT con JAAS

Información sobre cómo autenticar un cliente con JAAS. Complete los pasos de esta tarea para modificar el programa de ejemplo `JAASLoginModule.java` y configurar IBM WebSphere MQ para autenticar una aplicación Java de cliente MQTT con JAAS.

Antes de empezar

1. Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java". Consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes.
2. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
3. Debe tener acceso a los ejemplos de MQXR `JAASLoginModule` y `JAASPrincipal` Java en una instalación de IBM WebSphere MQ . Estos ejemplos se encuentran en la vía de acceso `%MQ_FILE_PATH%\mqxr\samples.`
4. Complete los pasos en Windows o en Linux; los ejemplos están tomados en Windows.
5. Para completar el paso “1” en la página 74, debe tener autorización para crear el gestor de colas de `MQXR_SAMPLE_QM` en IBM WebSphere MQ.

Acerca de esta tarea

En la tarea, se generan los parámetros de identificación de cliente de MQTT `Sample` de la versión de `JAASLoginModule`. Escribir los parámetros de cliente supone modificar el programa `JAASLoginModule` de ejemplo y configurar IBM WebSphere MQ para que cargue su versión de `JAASLoginModule`.

Procedimiento

1. Complete los pasos que se indican en “Compilar y ejecutar todas las aplicaciones Java de cliente MQTT desde Eclipse” en la página 16 para ejecutar el cliente Paho MQTT `Sample`.

   Su objetivo es preparar un entorno de desarrollo para desarrollar y probar la autenticación JAAS. Necesita un entorno de desarrollo Java para ajustar el módulo de autenticación JAAS. En el ejemplo, ejecutará el cliente Paho de ejemplo para Java para probar su configuración de JAAS. Para simplificar, utilice el mismo entorno de desarrollo para modificar tanto el cliente de ejemplo como el módulo de inicio de sesión de ejemplo de JAAS. También puede probar su módulo de inicio de sesión de JAAS con el cliente MQTT para C o con cualquier otro cliente MQTT.

2. Opcional: Añada un parámetro de nombre de usuario y contraseña al ejemplo MQTT de Paho.

   Nota: Si su cliente Paho para Java incluye los parámetros de nombre de usuario y contraseña, este paso no será necesario. Compruebe si hay actualizaciones disponibles en el sitio de descargas. Consulte Descargas de la comunidad de mensajería de IBM, si no, cambie su copia de `Sample.java`.

   a) Abra con el explorador de paquetes el paquete `org.eclipse.paho.sample.mqttv3app` del proyecto de ejemplos Paho.

   b) Pulse con el botón derecho del ratón en `Sample.java` **Copiar** > **Pegar**. En la ventana **Conflicto de nombre**, escriba el nombre `SampleForJAAS`.

   c) Añada las siguientes líneas de código en el método `main`.

      i) Después de la línea "`boolean ssl = false;`", declare las variables `userName` y `password`:

      ```
      String password = null;
      String userName = null;
      ```

      Para asegurar la compatibilidad con servidores MQTT anteriores, no defina los parámetros predeterminados de nombre de usuario y contraseña.

ii) Después de la línea, "`case 'v': ssl = Boolean.valueOf(args[+ +i]).booleanValue(); break;`", ponga los dos nuevos parámetro de entrada:

```
case 'u': userName = args[++i]; break;
case 'z': password = args[++i]; break;
```

iii) Antes de la línea, "`if (action.equals("publish")) {`", añada `userName` y `password` a los argumento del constructor de `Sample`:

```
Sample sampleClient = new Sample(url, clientId, cleanSession, quietMode, userName,
password);
```

d) Añada`userName` y `password` al constructor de `Sample`.

Cambie:

```
public Sample(String brokerUrl, String clientId, boolean cleanSession,
       boolean quietMode) throws MqttException {
```

A:

```
public Sample(String brokerUrl, String clientId, boolean cleanSession,
       boolean quietMode, String userName, char[] password) throws MqttException {
```

e) Añada las siguientes líneas de código en el constructor de `Sample`.

Después de la línea "`conOpt.setCleanSession(clean);`", defina las variables `userName` y `password` en el objeto `conOpt` del método `Sample`:

```
if(password != null ) {
   conOpt.setPassword(this.password.toCharArray());
}
if(userName != null) {
   conOpt.setUserName(this.userName);
}
```

f) En los métodos `publish` y `subscribe`, modifique las siguientes líneas de código:

Cambie la línea "`client.connect();`" a

```
client.connect(conOpt);
```

3. Cree un proyecto Java, `JAASSample`, para su ejemplo de JAAS.
   a) En su espacio de trabajo de Eclipse, abra el asistente **Nuevo proyecto Java**: Pulse **Archivo** > **Nuevo** > **proyecto Java**.
   b) En el campo **Nombre de proyecto**, escriba `JAASSample`.
   c) En las opciones de JRE, seleccione `J2SE-1.5` como entorno de ejecución JRE y pulse **Siguiente**.

      El JRE debe coincidir con el JRE que utilice el servidor IBM WebSphere MQ. IBM WebSphere MQ Version 7.5 ejecuta `J2SE-1.5`.
   d) En la ventana **Ajustes de Java**, pulse en la pestaña **Bibliotecas** y pulse en **Añadir JARS externos...**. Vaya a`%MQ_FILE_PATH%\mqxr\lib` y seleccione **MQXR.jar**; pulse **Finalizar**.
4. Importe las clases JAAS samples `JAASLoginModule` y `JAASPrincipal` .
   a) Pulse con el botón derecho del ratón en el proyecto `JAASSample` en el Explorador de paquetes **Importar ...** > **General** > **Sistema de archivos** y pulse **Siguiente**.
   b) Vaya a `%MQ_FILE_PATH%\mqxr\samples` y seleccione `JAASLoginModule.java` y `JAASPrincipal.java`; pulse **Finalizar**.
   c) Seleccione y pulse con el botón derecho del ratón en ambos archivos Java en el Explorador de paquetes, **Refactorizar ...** > **Mover**.
   d) En la ventana **Mover**, compruebe que `JAASSample` está seleccionado como destino para los dos elementos y pulse **Crear paquete...**

e) Escriba `samples` en el campo **Nombre** en el asistente **Nuevo paquete Java**; pulse **Finalizar** > **Aceptar**

Eclipse compila las clases Java importadas con una serie de avisos sobre valores no utilizados.

5. Renombre la clase `JAASLoginModule`

   Renombre la clase para que sea más fácil distinguirla de la clase de ejemplo `JAASLoginModule` que se proporciona con IBM WebSphere MQ.

   a) Pulse con el botón derecho del ratón en `JAASloginModule.java` en el explorador de paquetes **Refactorizar ...** > **Renombrar**.

   b) En la ventana **Renombrar unidad de compilación**, cambie el campo **Nuevo nombre** de `JAASloginModule` a `MyJAASloginModule`; pulse **Finalizar**.

6. Modifique la clase `MyJAASloginModule` para que dé como salida el contenido de los campos de devolución de llamada.

   a) Añada las siguientes líneas de código en `MyJAASloginModule.java`.

   ```
   System.out.println("Username=" + username
                   + "\nPassword=" + new String(password)
                   + "\nClientId=" + clientId
                   + "\nNetwork address=" + networkAddress);
   ```

   Añada las líneas justo antes de la sentencia:"`if (true) loggedIn = true;`"

   b) Pulse `CTRL+Shift+O` para reorganizar las importaciones y guarde el archivo.

7. Renombre `JAASPrincipal` a `MyJAASPrincipal`.

   Renombre la clase para evitar confusiones con la clase `JAASPrincipal` de ejemplo. En el ejemplo, deje el contenido de la clase `MyJAASPrincipal` inalterado.

8. Otorgue al ID de usuario que ejecuta los procesos del gestor de colas `read` y `execute` permisos a sus clases JAAS.

   a) En Windows Explorer, abra el directorio del espacio de trabajo de Eclipse. En el ejemplo, la ubicación del espacio de trabajo Eclipse se representa mediante la variable Eclipse , *`workspace_loc`*.

   b) Vaya al directorio que contiene las clases `MyJAASLoginModule` y `MyJAASPrincipal`.

   La vía de acceso del directorio es *`workspace_loc`*`\JAASSample\bin\samples`

   c) Seleccione y después pulse con el botón derecho del ratón sobre las dos clases y después pulse**Propiedades**; pulse en la pestaña **Seguridad** en la ventana **Propiedades**.

   d) Pulse **Añadir ...**, Escriba el nombre de objeto `mqm`y pulse **Comprobar nombres** para verificarlo; pulse **Aceptar**.

   e) Seleccione **mqm** en la lista **Nombres de usuarios o grupos** y seleccione **lectura y ejecución** y **lectura** en la lista de permisos para `mqm`; pulse Aceptar.

9. Configure IBM WebSphere MQ para que ejecute la clase `MyJAASLoginModule`.

   a) Añada un archivo `service.env` a la configuración IBM WebSphere MQ para definir las vías de acceso de clases para cargar la clase `MyJAASLoginModule`.

   Cree un archivo `service.env` con la siguiente sentencia de vía de acceso de clases en el directorio *`WMQ_DATA_PATH`*:

   ```
   CLASSPATH=user.dir\JAASSample\bin
   ```

   Donde *`user.dir`* es la raíz de directorio para los archivos de clase compilados en el espacio de trabajo deEclipse . El directorio *`WMQ_DATA_PATH`* contiene el directorio `qmgrs`. Consulte Variables de entorno adicionales.

   Consejo: Puede que `CLASSPATH=`*`user.dir`*`\JAASSample\bin` sea la única sentencia que haya en el archivo `service.env`

b) Añada una stanza, `MyJAASStanza`, al archivo `jaas.config` para identificar la claseMyJAASLoginModule relativa a las vías de acceso de clases en el archivo `service.env`.

`jaas.config` se encuentra en el directorio `mqxr` del gestor de colas; *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.

La stanza es:

```
MyJAASStanza {
  samples.MyJAASLoginModule required debug=true;
};
```

c) Configure un canal IBM WebSphere MQ Telemetry con el nombre de su stanza de configuración JAAS.

Ejecute el mandato siguiente desde una ventana de mandatos:

```
echo DEFINE CHANNEL('MyJAAS') CHLTYPE(MQTT) TRPTYPE(TCP) PORT(1890)
JAASCFG('MyJAASStanza') | runmqsc MQXR_SAMPLE_QM
```

Este mandato enlaza el canal MyJAAS con `MyJAASStanza` en el archivo `jaas.config`. Si no se especifica la opción MCAUSER ni la opción USECLTID en la definición de canal, el canal autoriza el acceso a los recursos del gestor de colas al nombre de usuario que proporcione el programa cliente MQTT. En el ejemplo, el nombre de usuario que proporciona el cliente es "`Guest`". En el ejemplo se utilizan también las autorizaciones existentes para `Guest` definidas por el archivo de mandatos `SampleMQM`.

10. Reinicie el servicio IBM WebSphere MQ Telemetry para leer los nuevos datos de configuración.
Para reiniciar el servicio IBM WebSphere MQ Telemetry, inicie el gestor de colas o el servicio desde IBM WebSphere MQ Explorer, o ejecute los siguientes mandatos para la configuración de ejemplo:

```
echo stop  service(SYSTEM.MQXR.SERVICE) | runmqsc  MQXR_SAMPLE_QM
echo start service(SYSTEM.MQXR.SERVICE) | runmqsc  MQXR_SAMPLE_QM
```

11. Ejecute el programa `Sample`.

Para configurar una configuración de ejecución de SampleForJAAS siga el mismo procedimiento que en el paso “1” en la página 74, con las modificaciones siguientes:

a) Establezca el número de puerto en 1890 para que coincida con la configuración de canal de MQTT.

b) Añada los parámetros `-u Guest -z password` a las contraseñas de la pestaña **(x)= Arguments** para la configuración de Suscriptor y para la de Publicador que ha creado para el programa `Sample`.

Los programas de ejemplo se ejecutan sin ningún cambio en la salida, excepto el número de puerto, que ahora es 1890 en lugar de 1883.

Vaya al directorio *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM y abra el archivo `mqxr.stdout`. La salida de `MyJAASLoginModule` se escribe en `mqxr.stdout`:

```
Username=Guest
Password=passord
ClientId=SampleJavaV3_subscribe
Network address=/127.0.0.1
```

Qué hacer a continuación

Si el ejemplo no funciona, lea el tema de resolución de problemas para JASS; “Resolución del problema: El servicio de telemetría no ha llamado al módulo de inicio de sesión JAAS” en la página 188, e intente los consejos de depuración que se proporcionan.

1. Añada -verbose a los parámetros en *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties. En este registro podrá ver si la clase se ha cargado satisfactoriamente.

   La salida se escribe en *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr.
2. Mire en *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log si aparecen excepciones en MyJAASLoginModule. Por ejemplo, si intenta dar como salida una password (contraseña) nula, que sea una matriz de caracteres en lugar de una cadena, se emite una excepción.
3. Mire en *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log. Si el nombre de usuario que aparece en userName no tiene autorización para acceder a los recursos del gestor de colas y el canal está configurado con la opción MCAUSER o USECLTID, los errores se notifican aquí.
4. Compruebe que el nombre de la stanza en *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config sea el mismo que el nombre del canal MQTT que está configurado para el puerto al que el cliente de Sample está intentando conectarse.
5. Compruebe que la vía de acceso en la stanza coincide con la vía de acceso a la clase MyJAASLoginModule en Eclipse; por ejemplo:

   ```
   MyJAASStanza {
     samples.MyJAASLoginModule required debug=true;
   };
   ```
6. Para descartar que el error resida en que la vía de acceso de clases del archivo service.env de *WMQ_DATA_PATH* no se esté seleccionando correctamente, cambie la línea "set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%" en %MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT y ponga su vía de acceso de clases. También puede repetir la vía de acceso de clases. Sin embargo, la vía de acceso de clases no contiene la vía de acceso de clases que se ha establecido en service.env, por lo tanto, esto sólo funciona si se modifica el archivo controlMQXR.BAT.

Conceptos relacionados

“Seguridad de MQTT” en la página 52
Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La identidad consiste en dar nombre al cliente que se va a autorizar y dar autorización. La autenticación consiste en probar la identidad del cliente y la autorización consiste en gestionar los derechos que se otorgan al cliente.

“Configuración JAAS del canal de telemetría” en la página 117
Configure JAAS para autenticar el valor de Username que envía el cliente.

Tareas relacionadas

“Resolución del problema: El servicio de telemetría no ha llamado al módulo de inicio de sesión JAAS” en la página 188
Averigüe si el servicio de telemetría (MQXR) no está llamando al módulo de inicio de sesión JAAS y configure JAAS para corregir el problema.

“Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro” en la página 56
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura Java de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java"

“Conexión de la aplicación Java de ejemplo de cliente MQTT en Android a través de SSL” en la página 64
Ponerse en marcha con el cliente de Android MQTT de ejemplo conectado a IBM WebSphere MQ a través de SSL.

Información relacionada

Variables de entorno adicionales

Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets

Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

Antes de empezar

1. Debe tener acceso a un servidor de MQTT version 3 que admita MQTT protocol sobre WebSockets.
2. El navegador debe admitir SSL y WebSocket protocol. Consulte "Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL" en la página 177.
3. Los canales SSL deben estar iniciados.

Acerca de esta tarea

Complete esta tarea para ejecutar el cliente de mensajería de MQTT para las páginas de ejemplo de JavaScript sobre SSL. Esta tarea le dirije a "Generación de claves y certificados" en la página 97 para crear certificados y configurar IBM WebSphere MQ.

Proteja el canal SSL con claves firmadas por una entidad emisora de certificados o bien con claves autofirmadas.

Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe admitir MQTT protocol sobre WebSockets seguro.

   - IBM MessageSight, y releases de IBM WebSphere MQ Versión 7.5.0.1 y posterior lo hacen.
2. Opcional: Instale un el kit de desarrollo de Java (JDK) de la Versión 7 o posterior.

   Si está configurando un sistema de prueba, y desea utilizar certificados autofirmados, debe utilizar el mandato **keytool** de JDK versión 7 para certificar los certificados. Si está configurando un sistema de producción y envoando solicitudes de firma de certificados (CSR) a una entidad emisora de certificados externa, no necesita la versión 7 de JDK.
3. Cree y ejecute los scripts para generar pares de claves y certificados, y configure IBM WebSphere MQ como el servidor de MQTT .

   Siga los pasos de "Generación de claves y certificados" en la página 97 para crear y ejecutar los scripts. Los scripts se enumeran también en "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 81.

   El nombre común del certificado de servidor debe coincidir con el nombre DNS name del canal servidor. Algunos navegadores aceptan certificados que contienen una lista de nombres comunes; por ejemplo:

   ```
   "CN=localhost, CN=*.example.com"
   ```

   Otros navegadores aceptan únicamente un nombre común. Por ejemplo, Firefox, hasta la versión 18, acepta sólo un nombre común. Esto puede ser distinto en versiones posteriores.
4. Compruebe que los canales SSL están en ejecución y configurados adecuadamente.

   En IBM WebSphere MQ, escriba el mandato siguiente en una ventana de mandatos:

   - 


     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

- Windows

```
echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
```

5. Instale los certificados en el almacén de certificados de navegador.

   Para el ejemplo, elija uno de los siguientes certificados de servidor:

   a. Para un certificado de servidor autofirmado, el certificado es `srvcertselfsigned.cer`.

   b. Para un certificado de servidor firmado por su entidad emisora de certificados privada, el certificado es `cacert.cer`.

   c. Para un certificado de servidor firmado por una entidad emisora de certificados externa, compruebe que el certificado raíz de la entidad emisora de certificados esté instalado en el almacén de certificados.

   En función de la compatibilidad de su navegador, `cacert.cer` en la lista de Autoridades emisoras de certificados raíz de confianza. Consulte “Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL” en la página 177.

6. Opcional: Autentique el cliente.

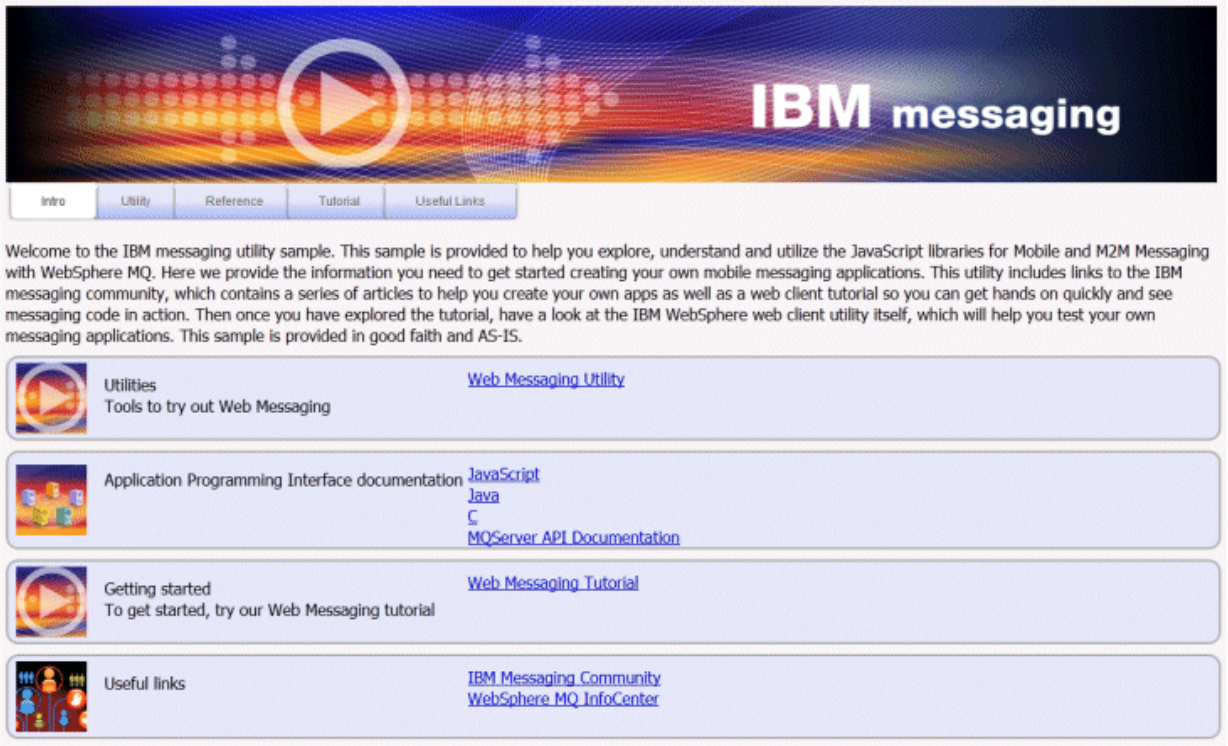
   En el ejemplo, instale `cacert.cer` para autenticar el servidor y cifrar el canal, pero no para autenticar el cliente. Para autenticar el cliente debe instalar el almacén de claves de cliente, `cltkeystore.p12`, en el navegador. No todos los navegadores permiten autenticar clientes. Consulte “Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL” en la página 177.

7. Establezca conexión con el canal WebSockets seguro.

   Abra el navegador y escriba el URL del canal WebSockets en la barra de direcciones; en el ejemplo sería:

```
https://localhost:8886
```

   IBM WebSphere MQ responde con la primera página del Páginas de JavaScript de ejemplo de cliente de mensajería MQTT.



Si falla la conexión, y ha ejecutado los scripts de ejemplo para configurar el gestor de colas de MQTT de ejemplo, pruebe a conectarse al canal WebSockets normal en el puerto 1886. Si la conexión es satisfactoria en el puerto 1886 se aísla el error a la conexión SSL.

```
https://localhost:1886
```

Conceptos relacionados

"El Cliente MQTT de mensajería para JavaScript y las aplicaciones web" en la página 119

"Cómo programar aplicaciones de mensajería en JavaScript" en la página 123

Tareas relacionadas

"Generación de claves y certificados" en la página 97

Siga este procedimiento para generar claves y certificados para clientes Java y C, incluyendo las aplicaciones de Android y iOS, y los servidores de IBM WebSphere MQ y IBM MessageSight.

"Iniciación a Cliente MQTT de mensajería para JavaScript" en la página 24

Puede empezar a trabajar con Cliente MQTT de mensajería para JavaScript yendo a la página inicial de ejemplo del cliente de mensajería y examinando los recursos a los que enlaza. Para ver esta página de inicio, debe configurar un servidor MQTT para aceptar conexiones de la Páginas de JavaScript de ejemplo de cliente de mensajería MQTT, a continuación, escriba el URL que ha configurado en el servidor en un navegador web. Cliente MQTT de mensajería para JavaScript se inicia automáticamente en el dispositivo y se muestra la página inicial de ejemplo del cliente de mensajería. Esta página contiene enlaces a programas de utilidad, documentación de la interfaz de programación, una guía de aprendizaje y otra información útil.

Referencia relacionada

"Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL" en la página 177

Las funciones de los diferentes navegadores varían según las distintas plataformas. Conocer las diferencias resulta útil para configurar las aplicaciones, autoridades de certificados (CA) y certificados de clientes para conectar utilizando Cliente MQTT de mensajería para JavaScript con SSL y WebSockets.

Scripts de ejemplo para configurar certificados SSL para Windows

** **

Estos archivos de mandato de ejemplo crean los certificados y los almacenes de certificados según se describe en los diversos pasos de la tarea. Además el ejemplo configura el gestor de colas de cliente MQTT para que utilice el almacén de certificados del servidor. El ejemplo suprime y vuelve a crear el gestor de colas llamando al script `SampleMQM.bat` que se proporciona en IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` define los nombres y vías de acceso a los certificados y otros parámetros que requieren los mandatos **`keytool`** y **openSSL**. Los valores se describen en los comentarios dentro del script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
```

```
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
```

```
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

Los mandatos del script `cleancert.bat` suprimen el gestor de colas del cliente MQTT para asegurarse de que el almacén de certificados de servidor no está bloqueado y después suprimen todos los almacenes de claves y los certificados que han creado los scripts de seguridad de ejemplo.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

Los mandatos del script `genkeys.bat` crean pares de claves para su entidad emisora de certificados privada, el servidor y un cliente.

```
@rem
@echo ------------------------------------------------------------------------
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

Los mandatos del script `sscerts.bat` exportan los certificados autofirmados de cliente y servidor de sus almacenes de claves e importan el certificado de servidor en el almacén de confianza del cliente, y el certificado de cliente en el almacén de claves del servidor. El servidor no tiene un almacén de confianza. Los mandatos crean un almacén de confianza de cliente en formato PEM a partir del almacén de confianza JKS del cliente.

```
@rem
@echo ------------------------------------------------------------------------
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ------------------------------------------------------------------------
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
```

```
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Este script importa el certificado raíz de la entidad emisora de certificados en los almacenes de claves privados. El certificado raíz de CA es necesario para crear la cadena de claves entre el certificado raíz y el certificado firmado. El script `cacerts.bat` exporta las solicitudes de certificados de cliente y servidor de sus almacenes de claves. El script firma las solicitudes de certificados con la clave de la entidad emisora de certificados privada en el almacén de claves `cajkskeystore.jks`, y después importa los certificados firmados otra vez en los mismos almacenes de claves de donde procedían. La importación crea la cadena de certificados con el certificado raíz de CA. El script crea un almacén de confianza del cliente en formato PEM a partir del almacén de claves del cliente en formato JKS.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
```

```
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

El script enumera los almacenes de claves y los certificados en el directorio de certificados. A continuación, crea el gestor de colas de ejemplo de MQTT y configura los canales de telemetría seguros.

```
@echo ------------------------------------------------------------------------------
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V 7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Creación y ejecución de Aplicación C de ejemplo de cliente MQTT seguro

Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura C de ejemplo en cualquier sistema operativo en el que pueda compilar el código C fuente. Verifique que puede ejecutar la aplicación C de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

Antes de empezar

1. Debe tener acceso a un servidor MQTT version 3.1 que admita MQTT protocol sobre SSL.
2. Si hay un cortafuegos entre el cliente y el servidor, compruebe que no bloquea el tráfico de MQTT .
3. Se proporcionan versiones binarias del cliente para bibliotecas C para diversos sistemas operativos. Para algunos de estos sistemas operativos, no se proporciona la versión segura del cliente en forma de archivo binario. Para estos sistemas operativos, debe seguir las instrucciones de "Creación de las bibliotecas de cliente MQTT para C" en la página 31.
4. Para la resolución de problemas, es posible que el servicio de soporte de IBM le pida que ejecute el cliente MQTT para C en una plataforma referencia.
5. Los canales SSL deben estar iniciados.

Para obtener una descripción general de las plataformas soportadas y de referencia, consulte Requisitos del sistema para IBM Mobile Messaging and M2M Paquete de clientes. Para obtener detalles de lo que está soportado para el cliente C, consulte las secciones relevantes de Requisitos del sistema para WebSphere MQ V7.5 Telemetry.

Acerca de esta tarea

Como ilustración, en este artículo se muestra cómo compilar y ejecutar el Aplicación C de ejemplo de cliente MQTT seguro en Windows desde la línea de mandatos. En la ilustración se utiliza Microsoft Visual Studio 2010 para compilar el cliente. Puede modificar los scripts de línea de mandatos para compilar y ejecutar la aplicación de ejemplo en otros sistemas operativos como Linux y iOS.

Nota:

En los scripts de Windows que se proporcionan en este artículo se presupone que ha generado el paquete OpenSSL completo desde el origen. Si elige utilizar las bibliotecas precompiladas que proporciona IBM, es posible que prefiera obtener una versión precompilada en binario de OpenSSL. Las bibliotecas precompiladas no están disponibles para utilizar con iOS.

Proteja el canal SSL con claves firmadas por una entidad emisora de certificados o bien con claves autofirmadas.

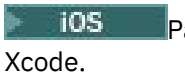
Procedimiento

1. Elija un servidor MQTT al que pueda conectar la aplicación cliente.

   El servidor debe dar soporte al protocolo MQTT version 3.1 a través de SSL. Todos los servidores MQTT de IBM lo hacen, incluidos IBM WebSphere MQ y IBM MessageSight. Consulte "Iniciación a los servidores MQTT" en la página 139.
2. Opcional: Instale un kit de desarrollo de Java (JDK) en la versión 7 o posterior.

   Versión 7 es necesario para ejecutar el mandato **keytool** para certificar certificados. Si no va a certificar certificados no necesita la versión 7 de JDK.
3. Instale un entorno de desarrollo C en la plataforma en la que está trabajando.

   Los archivos make de los ejemplos de este tema están dirigidos a las siguientes herramientas:

   - 

Para iOS, en Apple Mac con OS X 10.8.2 con las herramientas de desarrollo de iOS de Xcode.
   - 

Para Linux, gcc versión 4.4.6 desde Red Hat Enterprise Linux versión 6.2.

El nivel mínimo soportado de la biblioteca C `glibc` es el 2.12, y del kernel de Linux es el 2.6.32.

- Windows Para Microsoft Windows, Visual Studio versión 10.0.

4. Descargue Mobile Messaging and M2M Paquete de clientes e instale el SDK de MQTT .

   No hay ningún programa de instalación, sólo tiene que expandir el archivo una vez descargado.

   a. Descargue el Mobile Messaging and M2M Paquete de clientes.

   b. Cree una carpeta donde instalar el SDK.

      Quizás desee poner a esta carpeta el nombre MQTT. La vía de acceso a esta carpeta se denomina aquí *sdkroot*.

   c. Expanda el contenido del archivo Mobile Messaging and M2M Paquete de clientes comprimido en *sdkroot*. La expansión crea un árbol de directorios que empieza en *sdkroot*\SDK.

5. Opcional: Siga los pasos de "Creación de las bibliotecas de cliente MQTT para C" en la página 31.

   Realice este paso únicamente si el SDK de MQTT no incluye la biblioteca de cliente C segura para su sistema operativo de destino.

   - Windows Las bibliotecas son `mqttv3cs.lib` para compilar y `mqttv3cs.dll` para ejecutar.
   - Linux La biblioteca es `libmqttv3cs.so`
   - iOS La biblioteca es `libmqttv3cs.a`

6. Cree y ejecute los scripts para generar pares de claves y certificados, y configure IBM WebSphere MQ como el servidor de MQTT .

   Siga los pasos de "Generación de claves y certificados" en la página 97 para crear y ejecutar los scripts. Los scripts se enumeran también en "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 91.

7. Compruebe que los canales SSL están en ejecución y configurados adecuadamente.

   En IBM WebSphere MQ, escriba el mandato siguiente en una ventana de mandatos:

   - Linux

     ```
     echo 'DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     echo 'DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL' | runmqsc MQXR_SAMPLE_QM
     ```

   - Windows

     ```
     echo DISPLAY CHSTATUS(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     echo DISPLAY CHANNEL(SSL*) CHLTYPE(MQTT) ALL | runmqsc MQXR_SAMPLE_QM
     ```

8. Cree los scripts para compilar y ejecutar el Aplicación C de ejemplo de cliente MQTT seguro.

   a) Cree y ejecute `sscclient.bat` para probar el canal SSL que está protegido con certificados autofirmados.

   b) Cree y ejecute `cacclient.bat` para probar el canal SSL que está protegido con certificados de entidad emisora de certificados.

Resultados

Los resultados son similares a ejecutar el cliente no protegido.

```
Connected to ssl://localhost:8884
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:          MQTTV3SSample/C/v3
Message:        Message from MQTTv3 SSL C client
QoS:            2
Connected to ssl://localhost:8885
Subscribing to topic "MQTTV3SSample/#" qos 2
Topic:          MQTTV3SSample/C/v3
Message:        Message from MQTTv3 SSL C client
QoS:            2
```

Figura 16. Suscriptor seguro

```
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
MQTTV3SSample.c
Connected to ssl://localhost:8884
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
Connected to ssl://localhost:8885
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 17. Publicador seguro

Scripts para ejecutar el Aplicación C de ejemplo de cliente MQTT seguro

Ejecute los scripts de "Scripts de ejemplo para configurar certificados SSL para Windows" en la página 91 antes de ejecutar estos scripts.

Proteja Aplicación C de ejemplo de cliente MQTT con certificados autofirmados.
Ejecute este script con los certificados autofirmados que ha creado ejecutando el script `sscerts.bat`.

```
@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal
```

Figura 18. `sscclient.bat`

Ejecute la aplicación C de ejemplo de cliente MQTT seguro con certificados firmados por entidad emisora de certificados.

Ejecute el script con los certificados firmados por entidad emisora de certificado que ha creado ejecutando el script `cacerts.bat`.

```
@echo off
setlocal
cd %csamppath%
erase MQTTV3SSample.obj
erase MQTTV3SSample.exe
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3SSample.c" /link /
nologo ..\windows_ia32\mqttv3cs.lib
set path=%path%;%csamppath%\..\windows_ia32
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportopt% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportopt% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
@echo start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
start "MQTT Subscriber" MQTTV3SSample -a subscribe -b %host% -p %sslportreq% -k
%cltpemkeystore% -w %cltpemkeystorepass% -r %cltsrvpemtruststore% -v 1
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
@echo MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass%
-r %cltsrvpemtruststore% -v 1
MQTTV3SSample -b %host% -p %sslportreq% -k %cltpemkeystore% -w %cltpemkeystorepass% -r
%cltsrvpemtruststore% -v 1
pause
ping -n 2 127.0.0.1 > NUL 2>&1
endlocal
```

Figura 19. `cacclient.bat`

Conceptos relacionados

“Seguridad de MQTT” en la página 52

Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La identidad consiste en dar nombre al cliente que se va a autorizar y dar autorización. La autenticación consiste en probar la identidad del cliente y la autorización consiste en gestionar los derechos que se otorgan al cliente.

Tareas relacionadas

“Generación de claves y certificados” en la página 97

Siga este procedimiento para generar claves y certificados para clientes Java y C, incluyendo las aplicaciones de Android y iOS, y los servidores de IBM WebSphere MQ y IBM MessageSight.

Scripts de ejemplo para configurar certificados SSL para Windows

Ejemplo

Estos archivos de mandato de ejemplo crean los certificados y los almacenes de certificados según se describe en los diversos pasos de la tarea. Además el ejemplo configura el gestor de colas de cliente MQTT para que utilice el almacén de certificados del servidor. El ejemplo suprime y vuelve a crear el gestor de colas llamando al script `SampleMQM.bat` que se proporciona en IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` define los nombres y vías de acceso a los certificados y otros parámetros que requieren los mandatos **keytool** y **openSSL**. Los valores se describen en los comentarios dentro del script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

Los mandatos del script `cleancert.bat` suprimen el gestor de colas del cliente MQTT para asegurarse de que el almacén de certificados de servidor no está bloqueado y después suprimen todos los almacenes de claves y los certificados que han creado los scripts de seguridad de ejemplo.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
```

```
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

Los mandatos del script `genkeys.bat` crean pares de claves para su entidad emisora de certificados privada, el servidor y un cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

Los mandatos del script `sscerts.bat` exportan los certificados autofirmados de cliente y servidor de sus almacenes de claves e importan el certificado de servidor en el almacén de confianza del cliente, y el certificado de cliente en el almacén de claves del servidor. El servidor no tiene un almacén de confianza. Los mandatos crean un almacén de confianza de cliente en formato PEM a partir del almacén de confianza JKS del cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
```

```
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Este script importa el certificado raíz de la entidad emisora de certificados en los almacenes de claves privados. El certificado raíz de CA es necesario para crear la cadena de claves entre el certificado raíz y el certificado firmado. El script `cacerts.bat` exporta las solicitudes de certificados de cliente y servidor de sus almacenes de claves. El script firma las solicitudes de certificados con la clave de la entidad emisora de certificados privada en el almacén de claves `cajkskeystore.jks`, y después importa los certificados firmados otra vez en los mismos almacenes de claves de donde procedían. La importación crea la cadena de certificados con el certificado raíz de CA. El script crea un almacén de confianza del cliente en formato PEM a partir del almacén de claves del cliente en formato JKS.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

El script enumera los almacenes de claves y los certificados en el directorio de certificados. A continuación, crea el gestor de colas de ejemplo de MQTT y configura los canales de telemetría seguros.

```
@echo ____________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

V 7.5.0.1

```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Generación de claves y certificados

Siga este procedimiento para generar claves y certificados para clientes Java y C, incluyendo las aplicaciones de Android y iOS, y los servidores de IBM WebSphere MQ y IBM MessageSight.

Antes de empezar

1. Debe tener una copia del mandato **keytool**. No todas las versiones de **keytool** dan soporte a la conversión de almacenes de claves de almacén de claves Java (JKS) a sistema criptográfico de clave pública (PKCS) o a la firma de solicitudes de certificado. En este ejemplo se utiliza el mandato **keytool** de JDK versión 7.0, que permite las dos funciones.
2. Si tiene la intención de generar claves y certificados para el cliente para C, que se encuentran en formato PEM (Privacy-Enhanced Mail), debe tener una copia del mandato **openSSL**. Siga los pasos que se indican en “Creación de las bibliotecas de cliente MQTT para C” en la página 31 para crear el paquete openSSL.
3. Cambie los valores de los parámetros en el script `initcert.bat` según sus necesidades. Concretamente, puede optar por omitir los parámetros de contraseña para evitar escribir contraseñas. El mandato **keytool** solicita las contraseñas que faltan.

Acerca de esta tarea

Se requieren claves y certificados para crear conexiones SSL seguras entre clientes y servidores de MQTT. Esta tarea muestra dos formas diferentes de crear las claves y certificados que necesite: autofirmado y firmado por la entidad emisora de certificados propio. El método a seguir dependerá de cómo tenga previsto gestionar los almacenes de claves y los certificados.

Para utilizar certificados firmados por una entidad emisora de certificados externa, sustituya el paso de firma en `cacerts.bat` con el envío de las solicitudes de certificado a una autoridad de certificación externa. La entidad emisora de certificados puede devolver un certificado intermedio y un certificado raíz además del certificado firmado. Siga las instrucciones de la entidad emisora de certificados externa sobre dónde instalar los certificados devueltos.

El servidor de IBM WebSphere MQ busca certificados únicamente en el almacén de certificados que especifique en los parámetros de configuración del canal de telemetría. No busca además en el almacén `cacerts` de JSE. Un cliente Java busca certificados en el almacén de confianza que especifique. Si no especifica un almacén de confianza, busca en el almacén `cacerts` en el directorio JSE `jre\lib\security`. Los clientes Android buscan certificados en el almacén de certificados predefinido en el dispositivo Android. Las aplicaciones cliente de C e iOS buscan únicamente en los almacenes de certificados que especifique la aplicación.

Los clientes de Android y Java buscan los certificados de confianza en un almacén de confianza preconfigurado. Los certificados raíz de CA se almacenan en el almacén de certificados de confianza de Android y en el almacén de JSE `jre\lib\security\cacerts` . Si el certificado raíz de la entidad emisora de certificados que ha certificado el certificado del servidor ya está instalado en el almacén preconfigurado, no defina un almacén de confianza del cliente. La única configuración necesaria es definir el puerto TCP/IP para el canal servidor MQTT seguro.

Las herramientas para crear claves y certificados y gestionar todos los distintos formatos no son fáciles de usar. Tienen múltiples parámetros que hay que saber utilizar y **openSSL** requiere un archivo de configuración, `openssl.cnf`, y parámetros de línea de mandatos. No Hay ninguna herramienta que

proporcione todas las funciones necesarias para gestionar las claves y los certificados de las aplicaciones que funcionan tanto en C como en Java. Los canales de telemetría de IBM WebSphere MQ requieren un almacén de claves JKS así que los ejemplos utilizan básicamente las herramientas de certificados de Java, **ikeyman** y **keytool**. Sin embargo, las herramientas de Java no admiten el formato PEM, que es obligatorio en las aplicaciones cliente de C. Para crear almacenes de claves en formato PEM, ejecute la herramienta **openSSL**. La herramienta **openSSL** convierte almacenes de claves del formato PKCS12 al formato PEM, y **keytool** convierte almacenes de claves del formato JKS a PKCS12 y viceversa. No se requiere ningún archivo `openssl.cnf` para la conversión de almacenes de claves. Sólo necesitará **openSSL** si tiene previsto crear aplicaciones cliente de C o aplicaciones iOS. Si prefiere trabajar con **openSSL**, puede utilizarlo para firmar certificados en lugar de firmarlos con **keytool**.

Procedimiento

1. Abra una ventana de mandatos para ejecutar los scripts siguientes.
2. Cree y ejecute el script `initcert.bat` para establecer los parámetros necesarios para ejecutar los clientes de ejemplo seguros de MQTT .
3. Cree y ejecute el script `cleancert.bat` para limpiar el entorno y dejarlo listo para crear nuevos almacenes de claves y certificados.
4. Cree y ejecute el script `genkeys.bat` para generar los pares de claves que necesite.
5. Realice una de estas opciones:
   - Cree y ejecute el script `sscerts.bat` para crear certificados autofirmados.
   - Cree y ejecute el script `cacerts.bat` para crear cadenas de certificados firmados por una entidad emisora de certificados.
6. Cree y ejecute el script `mqcerts.bat` para crear el gestor de colas `MQXR_SAMPLE_QM` y configurar sus canales de telemetría.

Tareas relacionadas

“Creación y ejecución de Aplicación C de ejemplo de cliente MQTT seguro” en la página 87
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura C de ejemplo en cualquier sistema operativo en el que pueda compilar el código C fuente. Verifique que puede ejecutar la aplicación C de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

“Creación y ejecución de Aplicación Java de ejemplo de cliente MQTT seguro” en la página 56
Basándose en un ejemplo de Windows, puede empezar a trabajar con la aplicación segura Java de ejemplo en IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.Puede ejecutar una aplicación Cliente MQTT para Java en cualquier plataforma con JSE 1.5 o superior que sea compatible con "Java"

Scripts de ejemplo para configurar certificados SSL para Windows

Ejemplo

Estos archivos de mandato de ejemplo crean los certificados y los almacenes de certificados según se describe en los diversos pasos de la tarea. Además el ejemplo configura el gestor de colas de cliente MQTT para que utilice el almacén de certificados del servidor. El ejemplo suprime y vuelve a crear el gestor de colas llamando al script `SampleMQM.bat` que se proporciona en IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` define los nombres y vías de acceso a los certificados y otros parámetros que requieren los mandatos **keytool** y **openSSL**. Los valores se describen en los comentarios dentro del script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

```
@rem Set the paths to Version 7 of the JDK
@rem and to the directory where you built the openSSL package.
@rem Set short cuts to the tools.
set javapath=C:\Program Files\IBM\Java70
set keytool="%javapath%\jre\bin\keytool.exe"
set ikeyman="%javapath%\jre\bin\ikeyman.exe"
set openssl=%SDKRoot%\openSSL
set runopenssl="%openssl%\bin\openssl"
```

```
@rem Set the path to where certificates are to be stored,
@rem and set global security parameters.
@rem Omit set password, and the security tools prompt you for passwords.
@rem Validity is the expiry time of the certificates in days.
set certpath=%SDKRoot%\Certificates
set password=password
set validity=5000
set algorithm=RSA
```

```
@rem Set the certificate authority (CA) jks keystore and certificate parameters.
@rem Omit this step, unless you are defining your own certificate authority.
@rem The CA keystore contains the key-pair for your own certificate authority.
@rem You must protect the CA keystore.
@rem The CA certificate is the self-signed certificate authority public certificate.
@rem It is commonly known as the CA root certificate.
set caalias=caalias
set cadname="CN=mqttca.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

```
@rem Set the server jks keystore and certificate parameters.
@rem The server keystore contains the key-pair for the server.
@rem You must protect the server keystore.
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When you now export the server certificate,
@rem the exported certificate includes the certificate chain.
set srvalias=srvalias
set srvdname="CN=mqttserver.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set srvkeypass=%password%
@rem server key stores
set srvjkskeystore=%certpath%\srvkeystore.jks
set srvjkskeystorepass=%password%
@rem server certificates
set srvcertreq=%certpath%\srvcertreq.csr
set srvcertcasigned=%certpath%\srvcertcasigned.cer
set srvcertselfsigned=%certpath%\srvcertselfsigned.cer
```

```
@rem Set the client jks keystore and certificate parameters
@rem Omit this step, unless you are authenticating clients.
@rem The client keystore contains the key-pair for the client.
@rem You must protect the client keystore.
@rem If you then export the client certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the client keystore for the client key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the client certificate to the CA.
@rem When you now export the client certificate,
@rem the exported certificate includes the certificate chain.
set cltalias=cltalias
set cltdname="CN=mqttclient.ibm.id.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
set cltkeypass=%password%
@rem client key stores
set cltjkskeystore=%certpath%\cltkeystore.jks
set cltjkskeystorepass=%password%
set cltcertreq=%certpath%\cltcertreq.csr
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

```
@rem Set the paths to the client truststores signed by CA and signed by server key.
@rem You only need to define one of the trust stores.
@rem A trust store holds certificates that you trust,
@rem which are used to authenticate untrusted certificates.
@rem In this example, when the client authenticates the MQTT server it connects to,
@rem it authenticates the certificate it is sent by the server
@rem with the certificates in its trust store.
@rem For example, the MQTT server sends its server certificate,
@rem and the client authenticates it with either the same server certificate
@rem that you have stored in the cltsrvtruststore.jks trust store,
@rem or against the CA certificate, if the server certificate is signed by the CA.
set cltcajkstruststore=%certpath%\cltcatruststore.jks
set cltcajkstruststorepass=%password%
set cltsrvjkstruststore=%certpath%\cltsrvtruststore.jks
set cltsrvjkstruststorepass=%password%
```

```
@rem Set the paths to the client PKCS12 and PEM key and trust stores.
@rem Omit this step, unless you are configuring a C or iOS client.
@rem You only need to define either one of the trust stores for storing CA
@rem or server signed server certificates.
set cltp12keystore=%certpath%\cltkeystore.p12
set cltp12keystorepass=%password%
set cltpemkeystore=%certpath%\cltkeystore.pem
set cltpemkeystorepass=%password%
set cltcap12truststore=%certpath%\cltcatruststore.p12
set cltcap12truststorepass=%password%
set cltcapemtruststore=%certpath%\cltcatruststore.pem
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

```
@rem set WMQ Variables
set authopt=NEVER
set authreq=REQUIRED
set qm=MQXR_SAMPLE_QM
set host=localhost
set mcauser='Guest'
set portsslopt=8884
set chlopt=SSLOPT
set portsslreq=8885
set chlreq=SSLREQ
V 7.5.0.1 set portws=1886
set chlws=PLAINWS
set chlssloptws=SSLOPTWS
set portssloptws=8886
set chlsslreqws=SSLREQWS
set portsslreqws=8887
set mqlog=%certpath%\wmq.log
```

cleancert.bat

Los mandatos del script `cleancert.bat` suprimen el gestor de colas del cliente MQTT para asegurarse de que el almacén de certificados de servidor no está bloqueado y después suprimen todos los almacenes de claves y los certificados que han creado los scripts de seguridad de ejemplo.

```
@rem Delete the MQTT sample queue manager, MQXR_SAMPLE_QM
call "%MQ_FILE_PATH%\bin\setmqenv" -s
endmqm -i %qm%
dltmqm %qm%
```

```
@rem Erase all the certificates and key stores created by the sample scripts.
erase %cajkskeystore%
erase %cacert%
erase %srvjkskeystore%
erase %srvcertreq%
erase %srvcertcasigned%
erase %srvcertselfsigned%
erase %cltjkskeystore%
erase %cltp12keystore%
erase %cltpemkeystore%
erase %cltcertreq%
erase %cltcertcasigned%
erase %cltcertselfsigned%
erase %cltcajkstruststore%
```

```
erase %cltcap12truststore%
erase %cltcapemtruststore%
erase %cltsrvjkstruststore%
erase %cltsrvp12truststore%
erase %cltsrvpemtruststore%
erase %mqlog%
@echo Cleared all certificates
dir %certpath%\*.* /b
```

genkeys.bat

Los mandatos del script genkeys.bat crean pares de claves para su entidad emisora de certificados privada, el servidor y un cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Generate %caalias%, %srvalias%, and %cltalias% key-pairs in %cajkskeystore%,
%srvjkskeystore%, and %cltjkskeystore%
@rem
@rem -- Generate a client certificate and a private key pair
@rem Omit this step, unless you are authenticating clients.
%keytool% -genkeypair -noprompt -alias %cltalias% -dname %cltdname% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass% -keypass %cltkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

```
@rem Create CA, client and server key-pairs
@rem  -- Generate a CA certificate and private key pair - The extension asserts this is a
certificate authority certificate, which is required to import into firefox
%keytool% -genkeypair -noprompt -ext bc=ca:true -alias %caalias%  -dname %cadname%
-keystore %cajkskeystore% -storepass %cajkskeystorepass%  -keypass %cakeypass% -keyalg
%algorithm% -validity %validity%
```

sscerts.bat

Los mandatos del script sscerts.bat exportan los certificados autofirmados de cliente y servidor de sus almacenes de claves e importan el certificado de servidor en el almacén de confianza del cliente, y el certificado de cliente en el almacén de claves del servidor. El servidor no tiene un almacén de confianza. Los mandatos crean un almacén de confianza de cliente en formato PEM a partir del almacén de confianza JKS del cliente.

```
@rem
@echo ____________________________________________________________________________
@echo Export self-signed certificates:  %srvcertselfsigned% and %cltcertselfsigned%
@rem Export Server public certificate
%keytool% -exportcert -noprompt -rfc -alias %srvalias% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass% -file %srvcertselfsigned%
@rem Export Client public certificate
@rem Omit this step, unless you are authenticating clients.
%keytool% -exportcert -noprompt -rfc -alias %cltalias% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass% -file %cltcertselfsigned%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned server certificate %srvcertselfsigned% to client trust store:
%cltsrvjkstruststore%
@rem Import the server certificate into the client-server trust store (for server self-
signed authentication)
%keytool% -import -noprompt -alias %srvalias% -file %srvcertselfsigned% -keystore
%cltsrvjkstruststore% -storepass %cltsrvjkstruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Add selfsigned client certificate %cltcertselfsigned% to server trust store:
%srvjkskeystore%
@rem Import the client certificate into the server trust store (for client self-signed
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
```

```
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client-server trust store from the jks client-server trust store:
%cltsrvpemtruststore%
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltsrvjkstruststore% -destkeystore
%cltsrvp12truststore% -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltsrvjkstruststorepass% -deststorepass %cltsrvp12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltsrvp12truststore% -out %cltsrvpemtruststore% -passin
pass:%cltsrvp12truststorepass% -passout pass:%cltsrvpemtruststorepass%@rem
@rem
@echo ------------------------------------------------------------------------------
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

cacerts.bat

Este script importa el certificado raíz de la entidad emisora de certificados en los almacenes de claves privados. El certificado raíz de CA es necesario para crear la cadena de claves entre el certificado raíz y el certificado firmado. El script `cacerts.bat` exporta las solicitudes de certificados de cliente y servidor de sus almacenes de claves. El script firma las solicitudes de certificados con la clave de la entidad emisora de certificados privada en el almacén de claves `cajkskeystore.jks`, y después importa los certificados firmados otra vez en los mismos almacenes de claves de donde procedían. La importación crea la cadena de certificados con el certificado raíz de CA. El script crea un almacén de confianza del cliente en formato PEM a partir del almacén de claves del cliente en formato JKS.

```
@rem
@echo ------------------------------------------------------------------------------
@echo Export self-signed certificates: %cacert%
@rem
@rem Export CA public certificate
%keytool% -exportcert -noprompt -rfc -alias %caalias%  -keystore %cajkskeystore% -storepass
%cajkskeystorepass%  -file %cacert%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Add CA to server key and client key and trust stores: %srvjkskeystore%,
%cltjkskeystore%, %cltcajkstruststore%,
@rem The CA certificate is necessary to create key chains in the client and server key
stores,
@rem and to certify key chains in the server key store and the client trust store
@rem
@rem Import the CA root certificate into the server key store
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %srvjkskeystore%
-storepass %srvjkskeystorepass%
@rem Import the CA root certificate into the client key store
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltjkskeystore%
-storepass %cltjkskeystorepass%
@rem Import the CA root certificate into the client ca-trust store (for ca chained
authentication)
%keytool% -import -noprompt -alias %caalias% -file %cacert% -keystore %cltcajkstruststore%
-storepass %cltcajkstruststorepass%
```

```
@rem
@echo ------------------------------------------------------------------------------
@echo Create certificate signing requests: %srvcertreq% and %cltcertreq%
@rem
@rem Create a certificate signing request (CSR) for the server key
%keytool% -certreq -alias %srvalias% -file %srvcertreq% -keypass %srvkeypass% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Create a certificate signing request (CSR) for the client key
%keytool% -certreq -alias %cltalias% -file %cltcertreq% -keypass %cltkeypass% -keystore
%cltjkskeystore% -storepass %cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Sign certificate requests: %srvcertcasigned% and %cltcertcasigned%
@rem The requests are signed with the ca key in the cajkskeystore.jks keystore
@rem
@rem Sign server certificate request
%keytool% -gencert -infile %srvcertreq% -outfile %srvcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
@rem Sign client certificate request
@rem Omit this step, unless you are authenticating clients.
%keytool% -gencert -infile %cltcertreq% -outfile %cltcertcasigned% -alias %caalias%
-keystore %cajkskeystore% -storepass %cajkskeystorepass% -keypass %cakeypass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Import the signed certificates back into the key stores to create the key chain:
%srvjkskeystore% and %cltjkskeystore%
@rem
@rem Import the signed server certificate
%keytool% -import -noprompt -alias %srvalias% -file %srvcertcasigned% -keypass %srvkeypass%
-keystore %srvjkskeystore% -storepass %srvjkskeystorepass%
@rem Import the signed client certificate and key chain back into the client keystore
%keytool% -import -noprompt -alias %cltalias% -file %cltcertcasigned% -keypass %cltkeypass%
-keystore %cltjkskeystore% -storepass %cltjkskeystorepass%
@rem
@rem The CA certificate is needed in the server key store, and the client trust store
@rem to verify the key chain sent from the client or server
@echo Delete the CA certificate from %cltjkskeystore%: it causes a problem in converting
keystore to pem
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client-ca trust store from the jks client-ca trust store:
%cltcapemtruststore%
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltcajkstruststore%  -destkeystore
%cltcap12truststore%  -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltcajkstruststorepass%  -deststorepass %cltcap12truststorepass%
%openssl%\bin\openssl pkcs12 -in %cltcap12truststore%  -out %cltcapemtruststore%  -passin
pass:%cltcap12truststorepass%  -passout pass:%cltpemtruststorepass%
```

```
@rem
@echo ____________________________________________________________________________
@echo Create a pem client key store from the jks client keystore
@rem Omit this step, unless you are configuring a C or iOS client.
@rem
%keytool% -importkeystore -noprompt -srckeystore %cltjkskeystore%      -destkeystore
%cltp12keystore%     -srcstoretype jks -deststoretype pkcs12 -srcstorepass
%cltjkskeystorepass%     -deststorepass %cltp12keystorepass%
%openssl%\bin\openssl pkcs12 -in %cltp12keystore%     -out %cltpemkeystore%      -passin
pass:%cltp12keystorepass% -passout pass:%cltpemkeystorepass%
```

mqcerts.bat

El script enumera los almacenes de claves y los certificados en el directorio de certificados. A continuación, crea el gestor de colas de ejemplo de MQTT y configura los canales de telemetría seguros.

```
@echo ____________________________________________________________________________
@echo List keystores and certificates
dir %certpath%\*.* /b
```

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```

V 7.5.0.1

```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Identificación, autorización y autenticación de clientes MQTT

El servicio de telemetría (MQXR) publica temas de WebSphere MQ, o se suscribe a ellos, en nombre de los clientes MQTT, a través de los canales MQTT. El administrador de WebSphere MQ configura la identidad de canal MQTT que se utiliza para la autorización de WebSphere MQ. El administrador debe definir una identidad común para el canal o utilizar el `Username` o `ClientIdentifier` de un cliente conectado al canal.

El servicio de telemetría (MQXR) puede autenticar el cliente utilizando el valor `Username` que proporciona el cliente o bien utilizando un certificado de cliente. El valor de `Username` se autentica utilizando una contraseña que proporciona el cliente.

En resumen: la identificación de cliente es la selección de la identidad de cliente. En función del contexto, el cliente se identifica con el valor `ClientIdentifier`, el valor `Username`, una identidad común de cliente que crea el administrador o un certificado de cliente. El identificador de cliente utilizado para la comprobación de la autenticidad no tiene que ser el mismo identificador que se utiliza para la autorización.

Los programas de cliente MQTT establecen el valor de `Username` y de `Password` que se envían al servidor utilizando un canal MQTT. También pueden establecer las propiedades SSL que se necesitan para cifrar y autenticar la conexión. El administrador decide si autenticar el canal MQTT y cómo autenticarlo.

Para autorizar a un cliente MQTT acceder a los objetos de IBM WebSphere MQ, autorice al valor `ClientIdentifier`, o al valor `Username` del cliente, o autorice una identidad común del cliente. Para permitir que un cliente se conecte a IBM WebSphere MQ, autentique el valor de `Username`, o utilice un certificado de cliente. Configurar JAAS para autenticar el valor de `Username`, y configure SSL para autenticar un certificado de cliente.

Si define un valor `Password` en el cliente, cifre la conexión utilizando VPN, o configure el canal MQTT para que utilice SSL y mantenga la contraseña privada.

Resulta difícil gestionar certificados de clientes. Por este motivo, si los riesgos que conlleva la autenticación por contraseña resultan aceptables, ésta se utiliza a menudo para autenticar a los clientes.

Si existe una manera segura de gestionar y almacenar el certificado de cliente, se puede confiar en la autenticación con certificados. Sin embargo los certificados no suelen gestionarse de forma segura en los entornos en los que se utiliza la telemetría. En su lugar la autenticación de dispositivos que utilizan certificados de cliente se complementa con la autenticación de las contraseñas de cliente en el servidor. Debido a la complejidad adicional, la utilización de certificados de cliente se limita a aplicaciones altamente confidenciales. El uso de formas de autenticación se conoce como autenticación por dos factores. Debe conocer uno de los factores, por ejemplo, una contraseña, y tener otro, por ejemplo, un certificado.

En una aplicación en la que se necesite mucha confidencialidad como, por ejemplo, los dispositivos que utilicen la tecnología "chip and pin", el dispositivo se bloquea durante la fabricación para evitar intrusiones no autorizadas en el hardware y software internos. En el dispositivo se copia un certificado de cliente de confianza y duración limitada. El dispositivo se despliega en la ubicación en la que va a utilizarse. Cada vez que se utiliza el dispositivo, se realiza más autenticación, usando una contraseña u otro certificado de una tarjeta inteligente.

Identidad y autorización de cliente MQTT

Utilice el valor `ClientIdentifier`, `Username`, o una identidad de cliente común para autorizar a acceder a objetos de WebSphere MQ.

El administrador de IBM WebSphere MQ tiene tres opciones para seleccionar la identidad del canal MQTT. El administrador elige cuándo definir o modificar el canal MQTT que utiliza el cliente. La identidad se utiliza para autorizar el acceso a temas de IBM WebSphere MQ. Las opciones son las siguientes:

1. El identificador de cliente.
2. Una identidad que el administrador proporciona al canal.
3. El valor de `Username` que se ha pasado del cliente MQTT.

`claseUsername` es un atributo de la clase MqttConnectOptions. Debe establecerse antes de que el cliente se conecte al servicio. Su valor predeterminado es null.

Utilice el mandato IBM WebSphere MQ **setmqaut** para seleccionar qué objetos, y qué acciones, están autorizados para ser utilizados por la identidad asociada con el canal MQTT. Por ejemplo, para autorizar a una identidad de canal, `MQTTClient`, que proporciona el administrador del gestor de colas, `QM1`:

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

Información relacionada

Autorización a clientes MQTT a acceder a objetos de WebSphere MQ

Autenticación de cliente MQTT mediante contraseña

Autentique el valor de `Username` utilizando la contraseña del cliente. Puede autenticar el cliente utilizando una identidad diferente a la utilizada para autorizar al cliente a publicar temas y a suscribirse a los mismos.

El servicio de telemetría (MQXR) utiliza JAAS para autenticar el valor `Username` del cliente. JAAS utiliza el valor `Password` que proporciona el cliente MQTT.

El administrador de IBM WebSphere MQ decide si se debe autenticar el valor de `Username`, o no autenticar nada, configurando el canal MQTT al que se conecta un cliente. Los clientes pueden asignarse a diferentes canales y cada canal puede configurarse para que autentique sus clientes de formas diferentes. Mediante JAAS, puede configurar qué métodos deben autenticar el cliente y cuáles pueden hacerlo de forma opcional.

La opción que elija para autenticar la identidad no afecta a la opción que elija para autorizar la identidad. Es posible que desee configurar una identidad común para la autorización para facilitar las tareas administrativas, pero que se autentique cada usuario para que utilice dicha identidad. En el procedimiento siguiente se describen los pasos para autenticar los usuarios individuales para que utilice una identidad común:

1. El administrador de IBM WebSphere MQ establece la identidad del canal MQTT en un nombre cualquiera como, por ejemplo, `MQTTClientUser`, mediante IBM WebSphere MQ Explorer.
2. El administrador de IBM WebSphere MQ autoriza a `MQTTClient` publicar y suscribirse a cualquier tema:

   ```
   setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
   setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
   ```

3. El desarrollador de aplicaciones de cliente MQTT crea un objeto `MqttConnectOptions` y establece los valores de `Username` y `Password` antes de conectarse al servidor.
4. El desarrollador de seguridad crea un `LoginModule` JAAS para autenticar el valor `Username` con el valor `Password` y lo incluye en el archivo de configuración JAAS.
5. El administrador de IBM WebSphere MQ configura el canal MQTT para autenticar el valor `UserName` del cliente utilizando JAAS.

Autenticación de cliente MQTT mediante SSL

Las conexiones entre el cliente MQTT y el gestor de colas las inicia siempre el cliente MQTT. El cliente MQTT es siempre el cliente SSL. Tanto la autenticación de cliente del servidor como la autenticación de servidor del cliente MQTT son opcionales.

Al proporcionar al cliente un certificado digital firmado privado, puede autenticar el cliente MQTT en IBM WebSphere MQ. El administrador de IBM WebSphere MQ MQ puede obligar a los clientes MQTT a autenticarse ellos mismos en el gestor de colas mediante SSL. Sólo puede solicitar la autenticación de cliente como parte de una autenticación mutua.

Algunos tipos de redes privadas virtuales (VPN) como, por ejemplo, IPsec, autentican los puntos finales de una conexión TCP/IP, como alternativa a utilizar SSL. VPN cifra cada paquete IP que se transmite por la red. Una vez establecida la conexión VPN, se ha establecido la red de confianza. Puede conectar clientes MQTT a los canales de telemetría utilizando TCP/IP en la red VPN.

La autenticación de cliente utilizando SSL depende de que el cliente tenga un dato secreto. Este dato secreto es la clave privada del cliente en el caso de un certificado autofirmado o una clave que proporciona una entidad emisora de certificados. La clave se utiliza para firmar el certificado digital del cliente. Todas las personas que tengan la correspondiente clave pública pueden verificar el certificado digital. Los certificados pueden ser de confianza o, si están en cadena, se les puede realizar un seguimiento a través de una cadena de certificados a un certificado raíz de confianza. La verificación de clientes envía todos los certificados de la cadena de certificados que proporciona el cliente al servidor. El servidor comprueba la cadena de certificados hasta que encuentra un certificado en el que confía. El certificado de confianza puede ser un certificado público generado a partir de un certificado autofirmado, o un certificado raíz que normalmente ha emitido una entidad emisora de certificados. Un último paso, que es opcional, es la comparación del certificado de confianza con una lista de revocación de certificados "activa".

El certificado de confianza puede haberlo emitido una entidad emisora de certificados y puede estar incluido en el almacén de certificados JRE. Puede ser un certificado autofirmado o cualquiera que se haya añadido al almacén de claves del canal de telemetría como un certificado de confianza.

Nota: El canal de telemetría tiene un almacén de claves/almacén de confianza combinado que contiene tanto las claves privadas de uno o varios canales de telemetría, y todos los certificados públicos necesarios para autenticar clientes. Puesto que un canal SSL debe tener un almacén de claves, y es el mismo archivo que el almacén de confianza, nunca se hace referencia al almacén de certificados JRE. La implicación es que si la autenticación de un cliente requiere un certificado raíz de CA, debe colocar el certificado raíz en el almacén de claves del canal, aunque el certificado raíz de CA ya esté en el almacén de certificados JRE. Nunca se hace referencia alguna al almacén de certificados JRE.

Considere las amenazas a las que la autenticación del cliente pretende hacer frente y qué papel juega el cliente y el servidor en esa situación. La autenticación del certificado de cliente sola no es suficiente para evitar accesos no autorizados al sistema. Si otra persona ha tomado posesión del dispositivo del cliente, éste no tiene necesariamente que estar actuando con la autoridad del titular del certificado. No confíe nunca en una única defensa frente a ataques no deseados. Utilice al menos dos factores para la autenticación, además de la posesión complementaria de un certificado con conocimiento de información privada. Por ejemplo, utilice JAAS y autentique el cliente utilizando una contraseña que emita el servidor.

La principal amenaza a la que se enfrenta un certificado de cliente es que caiga en las manos equivocadas. El certificado se encuentra en un almacén de claves protegido con contraseña en el cliente. ¿Cómo se coloca en el almacén de claves? ¿Cómo consigue el cliente MQTT la contraseña para el almacén de claves? ¿Qué nivel de seguridad tiene la protección con contraseña? Los dispositivos de telemetría a menudo son fáciles de eliminar y pueden ser pirateados en privado. ¿Debe el hardware del dispositivo estar protegido contra manipulación no autorizada? La distribución y protección de certificados del lado del cliente se reconoce como difícil; se denomina el problema de la gestión de claves.

Una amenaza secundaria es que el dispositivo se utilice de forma incorrecta para acceder a servidores de forma imprevista. Por ejemplo, si la aplicación MQTT está amenazada, es posible utilizar algún punto débil de la configuración del servidor mediante la identidad de cliente autenticada.

Para autenticar un cliente MQTT mediante SSL, configure el canal de telemetría y el cliente.

•
•

Configuración del cliente MQTT para la autenticación de cliente mediante SSL

Para autenticar el cliente MQTT mediante SSL, el cliente se conecta a un canal de telemetría utilizando SSL. Debe especificar un puerto TCP que corresponda a un canal de telemetría que se haya configurado para autenticar clientes SSL.

Por ejemplo, en el cliente:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

La JVM del cliente debe utilizar la fábrica de sockets estándar de JSSE. Si está utilizando Java ME, debe asegurarse de que el paquete JSSE está cargado. Si utiliza Java SE, JSSE se ha incluido con el JRE desde la versión de Java 1.4.1.

La conexión SSL requiere que se establezcan una serie de propiedades SSL antes de la conexión. Puede establecer las propiedades pasándolas a la JVM utilizando el conmutador `-D` o puede establecer las propiedades utilizando el método `MqttConnectionOptions.setSSLProperties` .

Si carga una fábrica de sockets no estándar, llamando al método `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, la forma en que se pasan los valores SSL al socket de red es definida por la aplicación.

Añada el certificado digital al cliente, firmado mediante la clave privada del cliente, o mediante una CA, al almacén de claves protegido con contraseña del cliente. Si el certificado tiene una cadena de claves, puede añadir los certificados de la cadena de claves al almacén. Cuando el servidor verifica el certificado del cliente, utiliza los certificados que envía el cliente para que coincidan con los certificados de su almacén de claves. Busca la primera coincidencia en la cadena de claves con un certificado que ya tenga. El resto de la cadena de claves se ignora.

El cliente MQTT envía todos los certificados de su almacén de claves al servidor. Si el servidor autentica alguna de las cadenas de claves que el cliente envía, el cliente se autentica.

También puede utilizar las suites de cifrado SSL para la autenticación de cliente.A continuación se muestra una lista alfabética de las suites de cifrado SSL que se admiten actualmente:

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`

- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 Si tiene previsto utilizar suites de cifrado SHA-2, consulte el apartado "Requisitos del sistema para utilizar las suites de cifrado de clientes SHA-2 con clientes MQTT" en la página 176.

Conceptos relacionados

"Configuración del cliente de MQTT para la autenticación de canal mediante SSL" en la página 109
Para autenticar el canal de telemetría mediante SSL, el cliente debe conectarse al canal de telemetría utilizando SSL. Debe especificar un puerto que corresponda a un canal de telemetría que esté configurado para SSL. La configuración debe incluir un almacén de claves protegido con frase de contraseña que contenga el certificado digital firmado en privado del servidor.

Autenticación de canal de telemetría mediante SSL

Las conexiones entre el cliente MQTT y el gestor de colas las inicia siempre el cliente MQTT. El cliente MQTT es siempre el cliente SSL. Tanto la autenticación de cliente del servidor como la autenticación de servidor del cliente MQTT son opcionales.

El cliente intenta siempre autenticar el servidor, a menos que esté configurado para utilizar una CipherSpec que dé soporte a la conexión anónima. Si la autenticación falla, la conexión no se establece.

Algunos tipos de redes privadas virtuales (VPN) como, por ejemplo, IPsec, autentican los puntos finales de una conexión TCP/IP, como alternativa a utilizar SSL. VPN cifra cada paquete IP que se transmite por la red. Una vez establecida la conexión VPN, se ha establecido la red de confianza. Puede conectar clientes MQTT a los canales de telemetría utilizando TCP/IP en la red VPN.

La autenticación de servidor mediante SSL autentica el servidor al que va a enviar información confidencial. El cliente realiza las comprobaciones que coinciden con los certificados enviados desde el servidor, con los certificados colocados en su almacén de confianza o en su almacén `cacerts` de JRE.

El almacén de certificados JRE es un archivo JKS, `cacerts`. Se encuentra en JRE `InstallPath\lib\security\`. Se instala con la contraseña predeterminada `changeit`. Puede almacenar los certificados en que confíe en el almacén de certificados de JRE, o en el almacén de confianza del cliente. No puede utilizar ambos almacenes. Utilice el almacén de confianza del cliente si desea mantener los certificados públicos en los que el cliente confía separados de los certificados que utilicen otras aplicaciones Java. Utilice el almacén de certificados JRE si desea utilizar un almacén de certificados común para todas las aplicaciones Java que ejecuten en el cliente. Si decide utilizar el almacén de certificados de JRE, revise los certificados que contenga, para asegurarse de que confía en ellos.

Puede modificar la configuración JSSE indicando un proveedor de confianza diferente. Puede personalizar un proveedor de confianza para que realice diferentes comprobaciones en un certificado. En algunos entornos OGSi que han utilizado el cliente MQTT, el entorno proporciona un proveedor de confianza diferente.

Para autenticar el canal de telemetría utilizando SSL, configure el servidor y el cliente.

•

Conceptos relacionados

"Configuración del cliente de MQTT para la autenticación de canal mediante SSL" en la página 109
Para autenticar el canal de telemetría mediante SSL, el cliente debe conectarse al canal de telemetría utilizando SSL. Debe especificar un puerto que corresponda a un canal de telemetría que esté configurado para SSL. La configuración debe incluir un almacén de claves protegido con frase de contraseña que contenga el certificado digital firmado en privado del servidor.

Configuración del cliente de MQTT para la autenticación de canal mediante SSL

Para autenticar el canal de telemetría mediante SSL, el cliente debe conectarse al canal de telemetría utilizando SSL. Debe especificar un puerto que corresponda a un canal de telemetría que esté configurado para SSL. La configuración debe incluir un almacén de claves protegido con frase de contraseña que contenga el certificado digital firmado en privado del servidor.

Por ejemplo, en el cliente:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

La JVM del cliente debe utilizar la fábrica de sockets estándar de JSSE. Si está utilizando Java ME, debe asegurarse de que el paquete JSSE está cargado. Si utiliza Java SE, JSSE se ha incluido con el JRE desde la versión de Java 1.4.1.

La conexión SSL requiere que se establezcan una serie de propiedades SSL antes de la conexión. Puede establecer las propiedades pasándolas a la JVM utilizando el conmutador `-D` o puede establecer las propiedades utilizando el método `MqttConnectionOptions.setSSLProperties` .

Si carga una fábrica de sockets no estándar, llamando al método `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, la forma en que se pasan los valores SSL al socket de red es definida por la aplicación.

Codifique el cliente para que se conecte al canal de telemetría utilizando SSL, y configure el cliente para que confíe en un certificado de servidor de una de las tres formas siguientes:

Utilizando un certificado de servidor firmado por una entidad emisora de certificados conocida en el almacén `cacerts`.

Sin configuración adicional, si el servidor envía todas las claves intermedias en la cadena de certificados. Se aconseja revisar los certificados en el almacén `cacerts` del JRE de cliente y cambiar la contraseña para el almacén `cacerts`.

Otros certificados

Almacene los certificados que sean de confianza en el almacén de confianza del cliente. Debe almacenar al menos uno de los certificados de la cadena de certificados en el almacén de confianza. Establezca los parámetros del almacén de confianza en `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Utilización de un gestor de confianza personalizado

Implemente un proveedor de confianza y pásele el nombre del algoritmo que se utilice. Establezca el nombre de la clase de proveedor y el algoritmo que se vaya a utilizar en `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

También puede utilizar las suites de cifrado SSL para la autenticación de canal.A continuación se muestra una lista alfabética de las suites de cifrado SSL que se admiten actualmente:

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`
- `SSL_KRB5_EXPORT_WITH_RC4_40_SHA`
- `SSL_KRB5_WITH_3DES_EDE_CBC_MD5`
- `SSL_KRB5_WITH_3DES_EDE_CBC_SHA`
- `SSL_KRB5_WITH_DES_CBC_MD5`
- `SSL_KRB5_WITH_DES_CBC_SHA`
- `SSL_KRB5_WITH_RC4_128_MD5`
- `SSL_KRB5_WITH_RC4_128_SHA`
- `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_RSA_EXPORT_WITH_RC4_40_MD5`
- `SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA`
- V 7.5.0.2 `SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256`

- V 7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V 7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V 7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V 7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V 7.5.0.2 Si tiene previsto utilizar suites de cifrado SHA-2, consulte el apartado "Requisitos del sistema para utilizar las suites de cifrado de clientes SHA-2 con clientes MQTT" en la página 176.

Conceptos relacionados

"Configuración del cliente MQTT para la autenticación de cliente mediante SSL" en la página 107
Para autenticar el cliente MQTT mediante SSL, el cliente se conecta a un canal de telemetría utilizando SSL. Debe especificar un puerto TCP que corresponda a un canal de telemetría que se haya configurado para autenticar clientes SSL.

Privacidad de las publicaciones en los canales de telemetría

La privacidad de las publicaciones MQTT enviadas en cualquier dirección mediante los canales de telemetría está protegida mediante SSL, para cifrar las transmisiones a través de la conexión.

Los clientes MQTT que se conecten a los canales de telemetría utilizan SSL para proteger la privacidad de las publicaciones transmitidas en el canal, mediante el cifrado de claves simétricas. Puesto que los puntos finales no se autentican, no se puede confiar en un canal de cifrado solo. Combine la protección de privacidad con la autenticación del servidor o mutua.

Algunos tipos de redes privadas virtuales (VPN) como, por ejemplo, IPsec, autentican los puntos finales de una conexión TCP/IP, como alternativa a utilizar SSL. VPN cifra cada paquete IP que se transmite por la red. Una vez establecida la conexión VPN, se ha establecido la red de confianza. Puede conectar clientes MQTT a los canales de telemetría utilizando TCP/IP en la red VPN.

Para obtener información sobre una configuración típica, que cifre el canal y autentique el servidor, consulte "Autenticación de canal de telemetría mediante SSL" en la página 108.

El cifrado de las conexiones SSL sin autenticar el servidor expone la conexión frente a ataques de terceros. Aunque la información que se intercambia está protegida contra escuchas no autorizadas, no sabe con quién se está intercambiando. A menos que controle la red, estará expuesto a que alguien intercepte las transmisiones IP, haciéndose pasar por el punto final.

Puede crear una conexión SSL cifrada, sin autenticar el servidor, mediante un intercambio de claves Diffie-Hellman CipherSpec que dé soporte a SSL anónima. El secreto maestro, compartido entre el cliente y el servidor, y que se utiliza para cifrar transmisiones SSL, se establece sin intercambiar ningún certificado de servidor firmado en privado.

Puesto que las conexiones anónimas son inseguras, la mayoría de las implementaciones SSL no toman como valor predeterminado la utilización de las CipherSpecs anónimas. Si un canal de telemetría acepta una conexión SSL que solicita un cliente, el canal deberá tener un almacén de claves protegido mediante una frase de contraseña. De forma predeterminada, puesto que las implementaciones SSL no utilizan las CipherSpecs anónimas, el almacén de claves debe contener un certificado de firma privada que demuestre que el cliente puede autenticarse.

Si utiliza las CipherSpecs anónima, el almacén de claves del servidor debe existir, pero no es necesario que contenga ningún certificado firmado en privado.

Otra forma de establecer una conexión cifrada es sustituir el proveedor de confianza en el cliente por su propia implementación. El proveedor de confianza no ha podría autenticar el certificado de servidor, pero la conexión se cifraría.

Configuración SSL de clientes MQTT y canales de telemetría

Los clientes MQTT y el servicio WebSphere MQ Telemetry (MQXR) utilizan JSSE (Java Secure Socket Extension) para conectar canales de telemetría utilizando SSL. Los clientes MQTT C y el daemon de WebSphere MQ Telemetry para dispositivos no dan soporte a SSL.

Configure SSL para autenticar el canal de telemetría y el canal MQTT y cifre la transferencia de mensajes entre los clientes y el canal de telemetría.

Algunos tipos de redes privadas virtuales (VPN) como, por ejemplo, IPsec, autentican los puntos finales de una conexión TCP/IP, como alternativa a utilizar SSL. VPN cifra cada paquete IP que se transmite por la red. Una vez establecida la conexión VPN, se ha establecido la red de confianza. Puede conectar clientes MQTT a los canales de telemetría utilizando TCP/IP en la red VPN.

Puede configurar la conexión entre un cliente MQTT de Java y un canal de telemetría para utilizar el protocolo SSL sobre TCP/IP. Lo que se asegura depende de cómo configure SSL para que utilice JSSE. Empezando con la configuración más segura, puede configurar tres niveles de seguridad diferentes:

1. Permita que sólo se conecten los clientes MQTT de confianza. Conecte un cliente MQTT sólo a un canal de telemetría de confianza. Cifre los mensajes entre el cliente y el gestor de colas; consulte “Autenticación de cliente MQTT mediante SSL” en la página 106.
2. Conecte un cliente MQTT sólo a un canal de telemetría de confianza. Cifre los mensajes entre el cliente y el gestor de colas; consulte “Autenticación de canal de telemetría mediante SSL” en la página 108.
3. Cifre los mensajes entre el cliente y el gestor de colas; consulte “Privacidad de las publicaciones en los canales de telemetría” en la página 111.

Parámetros de configuración JSSE

Modifique los parámetros JSSE para cambiar la forma en la que se configura una conexión SSL. Los parámetros de configuración de JSSE se organizan en tres conjuntos:

1. Canal de IBM WebSphere MQ Telemetry
2. Cliente Java MQTT
3. JRE

Configure los parámetros del canal de telemetría mediante IBM WebSphere MQ Explorer. Establezca los parámetros del cliente Java MQTT en el atributo `MqttConnectionOptions.SSLProperties` . Modifique los parámetros de seguridad JRE editando los archivos en el directorio de seguridad JRE en el cliente y en el servidor.

Canal de IBM WebSphere MQ Telemetry

Establezca todos los parámetros SSL del canal de telemetría utilizando WebSphere MQ Explorer.

`ChannelName`

`ChannelName` es un parámetro necesario en todos los canales.

El nombre del canal identifica el canal asociado a un número de puerto particular. Ponga nombre a los canales; le servirá de ayuda para administrar los conjuntos de clientes MQTT.

`PortNumber`

`PortNumber` es un parámetro opcional en todos los canales. El valor predeterminado es 1883 para canales TCP, y 8883 para los canales SSL.

El número de puerto TCP/IP asociado a este canal. Los clientes MQTT se conectan a un canal especificando el puerto definido para el mismo. Si el canal tiene propiedades SSL, el cliente debe conectarse utilizando el protocolo SSL; por ejemplo:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

KeyFileName es un parámetro necesario para los canales SSL. Con los canales TCP, debe omitirse.

KeyFileNombre es la vía de acceso al almacén de claves Java que contiene los certificados digitales que proporciona. Utilice JKS, JCEKS o PKCS12 como el tipo de almacén de claves en el servidor.

Identifique el tipo de almacén de claves utilizando una de las extensiones de archivo siguientes:

.jks
.jceks
.p12
.pkcs12

Un almacén de claves que tenga cualquier otra extensión de archivo se supone que es un almacén de claves JKS.

Puede combinar un tipo de almacén de claves en el servidor con otros tipos de almacén de claves en el cliente.

Coloque el certificado privado del servidor en el almacén de claves. El certificado se conoce como el certificado del servidor. El certificado puede estar autofirmado o formar parte de una cadena de certificados firmado por una autoridad de firmas.

Si está utilizando una cadena de certificados, coloque los certificados asociados en el almacén de claves del servidor.

El certificado del servidor y cualquier certificado de su cadena de certificados se envían a los clientes para autenticar la identidad del servidor.

Si ha establecido ClientAuth en Required, el almacén de claves debe contener los certificados necesarios para autenticar el cliente. El cliente envía un certificado autofirmado, o una cadena de certificados, y el cliente se autentica mediante la primera verificación de este material contra un certificado del almacén de claves. Con el uso de una cadena de certificados, un certificado puede verificar muchos clientes, incluso si se emiten con certificados de cliente distintos.

PassPhrase

PassPhrase es un parámetro necesario para los canales SSL. Con los canales TCP, debe omitirse.

La frase de contraseña se utiliza para proteger el almacén de claves.

ClientAuth

ClientAuth es un parámetro SSL opcional. Toma como valor predeterminado que no se autentique ningún cliente. Con los canales TCP, debe omitirse.

Establezca ClientAuth si desea que el servicio de telemetría (MQXR) autentique el cliente, antes de permitir que el cliente se conecte al canal de telemetría.

Si establece ClientAuth, el cliente debe conectarse al servidor utilizando SSL y autenticar el servidor. Como respuesta a la definición de ClientAuth, el cliente envía su certificado digital al servidor y cualquier otro certificado de su almacén de claves. El certificado digital se conoce como el certificado de cliente. Estos certificados se autentican con los certificados contenidos en el almacén de claves del canal, y en el almacén cacerts de JRE.

CipherSuite

CipherSuite es un parámetro SSL opcional. El valor predeterminado es intentar todas las CipherSpecs habilitadas. Con los canales TCP, debe omitirse.

Si desea utilizar una CipherSpec particular, establezca CipherSuite en el mismo nombre que la CipherSpec que debe utilizarse para establecer una conexión SSL.

El servicio de telemetría y el cliente MQTT negocian una CipherSpec común a partir de todas las CipherSpecs que están habilitadas en cada extremo. Si se especifica una CipherSpec determinada en uno de los extremos de la conexión, o en ambos, debe coincidir con la CipherSpec del otro extremo.

Instale otros cifrados añadiendo otros proveedores adicionales a JSSE.

Federal Information Processing Standards (FIPS)

FIPS es un parámetro opcional. De forma predeterminada, no está definido.

Bien en el panel de propiedades del gestor de colas, o bien mediante **runmqsc**, establezca SSLFIPS. SSLFIPS especifica si sólo se deben utilizar algoritmos certificados por FIPS.

Revocation namelist

Revocation namelist es un valor opcional. De forma predeterminada, no está definido.

Bien en el panel de propiedades del gestor de colas, o bien mediante **runmqsc**, establezca SSLCRLNL. SSLCRLNL especifica una lista de nombres de objetos de información de autenticación que se utilizan para proporcionar ubicaciones de revocación de certificados.

No se utiliza ningún otro parámetro de gestor de colas que defina propiedades SSL.

Cliente Java MQTT

Establezca las propiedades SSL para el cliente Java en MqttConnectionOptions.SSLProperties; por ejemplo:

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

Los nombres y valores de propiedades específicas se describen en la documentación de la API para MqttConnectOptions.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.

Protocol

Protocol es opcional.

El protocolo se selecciona bajo negociación con el servidor de telemetría. Si necesita un protocolo específico, puede seleccionar uno. Si el servidor de telemetría no da soporte al protocolo, la conexión falla.

ContextProvider

ContextProvider es opcional.

KeyStore

KeyStore es opcional. Configúrelo si ha establecido ClientAuth en el servidor para forzar la autenticación del cliente.

Coloque el certificado digital del cliente, firmado mediante su clave privada, en el almacén de claves. Especifique la vía de acceso del almacén de claves y la contraseña. El tipo y el proveedor son opcionales. JKS es el tipo predeterminado, e IBMJCE el proveedor predeterminado.

Especifique un proveedor de almacén de claves diferente que haga referencia a una clase que añada un nuevo proveedor de almacén de claves. Pase el nombre del algoritmo utilizado

por el proveedor del almacén de claves para crear una instancia de KeyManagerFactory estableciendo el nombre del gestor de claves.

TrustStore

TrustStore es opcional. Puede colocar todos los certificados de confianza en el almacén cacerts de JRE.

Configure el almacén de confianza si desea tener almacén de confianza diferente para el cliente. Es posible que no pueda configurar el almacén de confianza si el servidor utiliza un certificado emitido por una CA bien conocida que ya tenga su certificado raíz almacenado en cacerts.

Añada el certificado firmado públicamente del servidor o el certificado raíz al almacén de confianza, y especifique la vía de acceso de almacén de confianza y la contraseña. JKS es el tipo predeterminado, e IBMJCE el proveedor predeterminado.

Especifique un proveedor de almacén de confianza diferente que haga referencia a una clase que añada un nuevo proveedor de almacén de confianza. Pase el nombre del algoritmo utilizado por el proveedor del almacén de confianza para crear una instancia de TrustManagerFactory estableciendo el nombre del gestor de confianza.

JRE

En el JRE se configuran otros aspectos sobre la seguridad Java que afectan al comportamiento del SSL en el cliente y en el servidor. Los archivos de configuración en Windows se encuentran en *Java Installation Directory*\jre\lib\security. Si va a utilizar el JRE que se proporciona con IBM WebSphere MQ la vía de acceso es la que se indica en la tabla siguiente:

Tabla 3. Vías de acceso de archivo por plataforma para archivos de configuración JRE SSL

Plataforma	**Vía de acceso de archivo**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux para System x de 32 bits	*WMQ Installation Directory*/java/jre/lib/security
Otras plataformas UNIX and Linux	*WMQ Installation Directory*/java/jre64/jre/lib/security

Entidades emisoras de certificados conocidas

El archivo cacerts contiene los certificados raíz de las entidades emisoras de certificados conocidas. El archivo cacerts se utiliza de forma predeterminada, a menos que especifique un almacén de confianza. Si utiliza el almacén cacerts, o no proporciona ningún almacén de confianza, debe revisar y editar la lista de firmantes que aparece en cacerts para que se cumplan sus requisitos de seguridad.

Puede abrir cacerts utilizando el mandato strmqikmde WebSphere MQ que ejecuta el programa de utilidad IBM Key Management. Abra cacerts como un archivo JKS, utilizando la contraseña changeit. Modifique la contraseña para proteger el archivo.

Configuración de clases de seguridad

Utilice el archivo java.security para registrar proveedores de seguridad adicionales y otras propiedades de seguridad predeterminadas.

Permisos

Utilice el archivo java.policy para modificar los permisos otorgados a los recursos. javaws.policy otorga permisos a javaws.jar

Fuerza de cifrado

Algunos JRE se entregan con poca fuerza de cifrado. Si no puede importar claves a los almacenes de claves, la poca fuerza de cifrado puede ser la causa. Intente iniciar **ikeyman** con el mandato

strmqikm, o descargue archivos de jurisdicción fuerte, pero limitada, de IBM developer kits, Security information.

Importante: Es posible que su país tenga restricciones sobre la importación, posesión, utilización y nueva exportación a otro país de software cifrado. Antes de descargar o utilizar archivos de políticas sin restricciones, debe comprobar las leyes existentes en su país. Compruebe sus regulaciones y políticas sobre importación, posesión, utilización y nueva exportación de software cifrado para ver si están permitidas estas acciones.

Modificación del proveedor de confianza para permitir al cliente para conectarse a cualquier servidor

En el siguiente ejemplo se muestra cómo añadir un proveedor de confianza y cómo hacer referencia al mismo desde el código de cliente MQTT. En el ejemplo no se autentican el cliente ni el servidor. La conexión SSL resultante se cifra sin que se haya autenticado.

El fragmento de código que aparece en la Figura 20 en la página 116 establece el proveedor de confianza `AcceptAllProviders` y el gestor de confianza del cliente MQTT.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

Figura 20. Fragmento de código de cliente MQTT

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
                AcceptAllTrustManagerFactory.class.getName());
    }
```

Figura 21. AcceptAllProvider.java

```
    protected static class AcceptAllTrustManagerFactory extends
            javax.net.ssl.TrustManagerFactorySpi {
        public AcceptAllTrustManagerFactory() {}
        protected void engineInit(java.security.KeyStore keystore) {}
        protected void engineInit(
                javax.net.ssl.ManagerFactoryParameters parameters) {}
        protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
            return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
        }
```

Figura 22. AcceptAllTrustManagerFactory.java

```
protected static class AcceptAllX509TrustManager implements
        javax.net.ssl.X509TrustManager {
    public void checkClientTrusted(
            java.security.cert.X509Certificate[] certificateChain,
            String authType) throws java.security.cert.CertificateException {
        report("Client authtype=" + authType);
        for (java.security.cert.X509Certificate certificate : certificateChain) {
            report("Accepting:" + certificate);
        }
    }
    public void checkServerTrusted(
            java.security.cert.X509Certificate[] certificateChain,
            String authType) throws java.security.cert.CertificateException {
        report("Server authtype=" + authType);
        for (java.security.cert.X509Certificate certificate : certificateChain) {
            report("Accepting:" + certificate);
        }
    }
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return new java.security.cert.X509Certificate[0];
    }
private static void report(String string) {
       System.out.println(string);
  }
}
```

Figura 23. AcceptAllX509TrustManager.java

Configuración JAAS del canal de telemetría

Configure JAAS para autenticar el valor de `Username` que envía el cliente.

El administrador de WebSphere MQ configura qué canales MQTT que necesitan autenticación de cliente mediante JAAS. Especifique el nombre de una configuración JAAS para cada canal que vaya a realizar la autenticación JAAS. Los canales pueden utilizar todos la misma configuración JAAS o utilizar configuraciones JAAS diferentes. Las configuraciones se definen en *WMQData directory*`\qmgrs\`*qMgrName*`\mqxr\jaas.config`.

El archivo `jaas.config` está organizado por nombre de configuración JAAS. Bajo cada nombre de configuración aparece una lista de las configuraciones de inicio de sesión; consulte la Figura 24 en la página 118.

JAAS proporciona cuatro módulos de inicio de sesión estándar. Los módulos de inicio de sesión estándar de NT y UNIX son de valor limitado.

JndiLoginModule
Autentica un servicio de directorio configurado con JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Autentica utilizando protocolos Kerberos.

NTLoginModule
Autentica utilizando la información de seguridad de NT para el usuario actual.

UnixLoginModule
Autentica utilizando la información de seguridad UNIX del usuario actual.

El problema al utilizar `NTLoginModule` o `UnixLoginModule` es que el servicio de telemetría (MQXR) se ejecuta con la identidad `mqm` y no con la identidad del canal MQTT. `mqm` es la identidad que se pasa a `NTLoginModule` o a `UnixLoginModule` para la autenticación, y no la identidad del cliente.

Para solucionar este problema, escriba un módulo de inicio de sesión propio o utilice otros módulos de inicio de sesión estándar. Con WebSphere MQ Telemetry se proporciona un archivo `JAASLoginModule.java` de ejemplo. Es una implementación de la interfaz de `javax.security.auth.spi.LoginModule`. Utilícelo para desarrollar un método de autenticación propio.

Las clases nuevas de LoginModule que proporcione deben encontrarse en la vía de acceso de clase del servicio de telemetría (MQXR). No coloque sus clases en los directorios de WebSphere MQ que estén en

la vía de acceso de clases. Cree sus propios directorios y defina la vía de acceso de clase completa del servicio de telemetría (MQXR).

Puede aumentar la vía de acceso de clase que utiliza el servicio de telemetría (MQXR) estableciendo la vía de acceso de clase en el archivo `service.env`. CLASSPATH debe estar en mayúsculas y la sentencia de vía de acceso de clase sólo puede contener literales. No puede utilizar variables en CLASSPATH; por ejemplo CLASSPATH=%CLASSPATH% no es correcto. El servicio de telemetría (MQXR) establece su propia vía de acceso de clases. Se añade el valor CLASSPATH definido en el archivo `service.env`.

El servicio de telemetría (MQXR) proporciona dos devoluciones de llamada que devuelven el valor de `Username` y de `Password` del cliente conectado al canal MQTT. El `Nombre de usuario` y la `Contraseña` se establecen en el objeto `MqttConnectOptions` . Consulte Figura 25 en la página 118 para obtener un ejemplo de cómo acceder al `Nombre de usuario` y la `Contraseña`.

Ejemplos

Ejemplo de un archivo de configuración JAAS con una configuración denominada `MQXRConfig`.

```
MQXRConfig {
  samples.JAASLoginModule required debug=true;
  //com.ibm.security.auth.module.NTLoginModule required;
  //com.ibm.security.auth.module.Krb5LoginModule required
  //                 principal=principal@your_realm
  //                 useDefaultCcache=TRUE
  //                 renewTGT=true;
  //com.sun.security.auth.module.NTLoginModule required;
  //com.sun.security.auth.module.UnixLoginModule required;
  //com.sun.security.auth.module.Krb5LoginModule required
  //                 useTicketCache="true"
  //                 ticketCache="${user.home}${/}tickets";
};
```

Figura 24. Archivo `jaas.config` de ejemplo

Ejemplo de un módulo de inicio de sesión JAAS codificado para recibir el valor de `Username` y de `Password` que proporciona un cliente MQTT.

```
  public boolean login()
      throws javax.security.auth.login.LoginException {
    javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
    callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
    callbacks[1] = new javax.security.auth.callback.PasswordCallback(
        "PasswordCallback", false);
    try {
      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // Accept everything.
      if (true) {
        loggedIn = true;
      } else
        throw new javax.security.auth.login.FailedLoginException("Login failed");

      principal= new JAASPrincipal(username);

    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }

    return loggedIn;
  }
```

Figura 25. Método `JAASLoginModule.Login()` de ejemplo

Conceptos de programación de clientes

Los conceptos que se describen en esta sección le ayudarán a comprender el cliente de Java para la versión 3.1 de MQTT protocol. Estos conceptos complementan la documentación de API que acompaña al paquete `com.ibm.micro.client.mqttv3`.

`com.ibm.micro.client.mqttv3` contiene las clases que proporcionan los métodos públicos para las implementaciones de Java del protocolo MQTT versión 3.1 . El paquete `com.ibm.micro.client.mqttv3` y los paquetes que lo acompañan que implementan el protocolo para Java SE y ME, se proporcionan con la instalación de IBM WebSphere MQ Telemetry.

Para desarrollar y ejecutar un cliente MQTT debe copiar o instalar estos paquetes en el dispositivo cliente. No es necesario que instale un cliente aparte en tiempo de ejecución.

Las condiciones de licencia para los clientes van asociadas al servidor al que conecta los clientes.

El cliente de Java es una implementación de referencia de la versión 3.1 de MQTT protocol. Puede implementar sus propios clientes en los distintos lenguajes adecuados a sus distintas plataformas de dispositivos. Consulte Formato y protocolo de MQ Telemetry Transport para obtener información detallada.

La documentación de la API de cliente para el paquete `com.ibm.micro.client.mqttv3` no hace ninguna presunción sobre a qué servidor se conecta el cliente. Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.El comportamiento del cliente puede diferir ligeramente al conectarlo a servidores distintos. A continuación se describe el comportamiento del cliente al conectarlo al servicio de telemetría (MQXR) de IBM WebSphere MQ.

El Cliente MQTT de mensajería para JavaScript y las aplicaciones web

Hasta hace poco, la programación de aplicaciones web y la creación de aplicaciones de mensajería eran disciplinas distintas. Independientemente de cuál sea su experiencia anterior, hay ventajas importantes en el uso conjunto de JavaScript y la mensajería. Al codificar la aplicación de mensajería como una aplicación web, se puede separar la aplicación y ejecutarla en cualquier navegador actualizado. Si cambia la aplicación, se utiliza la última versión cada vez que se renueva el navegador. El navegador también se ocupa de la seguridad y de la transmisión fiable de los mensajes.

Cómo facilita el despliegue de aplicaciones el hecho de utilizar una aplicación web

Si tiene experiencia en desarrollar y desplegar aplicaciones de mensajería tradicionales en (por ejemplo) IBM WebSphere MQ, estará familiarizado con los siguientes procesos de despliegue:

1. El administrador del sistema instala o incorpora la biblioteca de cliente.
2. El administrador del sistema organiza que la aplicación de mensajería se distribuya a los usuarios finales y se instale en sus sistemas locales.
3. Cuando hay cambios en el código, el administrador del sistema repite los pasos previos (o sea que la gestión de cambios es compleja).

Si codifica su aplicación de mensajería como una aplicación web, el proceso de despliegue es el siguiente:

1. El administrador del sistema envía la aplicación web y la biblioteca de cliente a un URL.
2. El navegador del usuario final une la aplicación web y la biblioteca de cliente.
3. Cuando hay cambios en el código, se escoge la versión actualizada al renovar el navegador (o sea que la gestión de cambios es sencilla).

Por qué puede preferir utilizar la mensajería directamente del navegador en sus aplicaciones web

Si tiene experiencia en la programación de aplicaciones en JavaScript, puede que le interese en conocer las ventajas de los sistemas de mensajería como IBM WebSphere MQ:

- Si envía y recibe mensajes a través de un sistema de mensajería, ese sistema es el responsable de garantizar que los mensajes se entregan.
- Debido a que el sistema de mensajería se encarga de la entrega, la aplicación web puede "transmitir y olvidar". Esto simplifica enormemente la lógica de programación. Si otro se encarga de entregar los mensajes, no es necesario que el código compruebe que han llegado. La aplicación no tiene que gestionar acuses de recibo ni guardar los mensajes no entregados para reintentarlos más tarde.
- Los sistemas de mensajería ofrecen mensajería orientada a eventos. La aplicación de cliente ya no tiene que enviar una solicitud ni estar después comprobando continuamente si hay respuesta. En vez de eso, el servidor de mensajería envía un mensaje a la aplicación cliente cuando se produce algún evento interesante. Esto también significa que la aplicación cliente recibe una alerta tan pronto como se produce el evento, en lugar de tener que esperar hasta la próxima vez que la aplicación sondea al servidor.
- La mensajería orientada a eventos también reduce de forma drástica la carga en el dispositivo que aloja la aplicación cliente, el tráfico de red entre el navegador y el servidor de mensajería y la carga en el servidor de mensajería. Esto es cada vez más importante, ya que hay cada vez más sistemas que se ejecutan en dispositivos móviles y se conectan a través de redes inalámbricas.

Cómo encajan las piezas

El Cliente MQTT de mensajería para JavaScript incluye una biblioteca de cliente y una aplicación web de ejemplo que utiliza esa biblioteca. Puede codificar su propia aplicación web que utilice la biblioteca. La aplicación web y la biblioteca de cliente se ponen después a disposición pública en el URL que usted elija, por ejemplo a través de un gestor de colas de MQ (como en el diagrama siguiente) o a través de un servidor de aplicaciones. El navegador coge la aplicación web y la biblioteca de cliente, y después la aplicación web utiliza el navegador para conectarse e intercambiar mensajes con un servidor MQTT como por ejemplo IBM WebSphere MQ Telemetry o IBM MessageSight.

Los flujos son los siguientes:

1. Cada instancia del navegador renueva su conexión al URL donde está disponible la aplicación web y se carga en el navegador una versión actualizada de la aplicación web y de la biblioteca de cliente.
2. La aplicación web se conecta a un gestor de colas, utilizando MQTT sobre WebSocket protocol y se suscribe a un tema de interés.
3. El gestor de colas utiliza la misma conexión para enviar mensajes que coincidan con la subscripción de vuelta a la aplicación web.

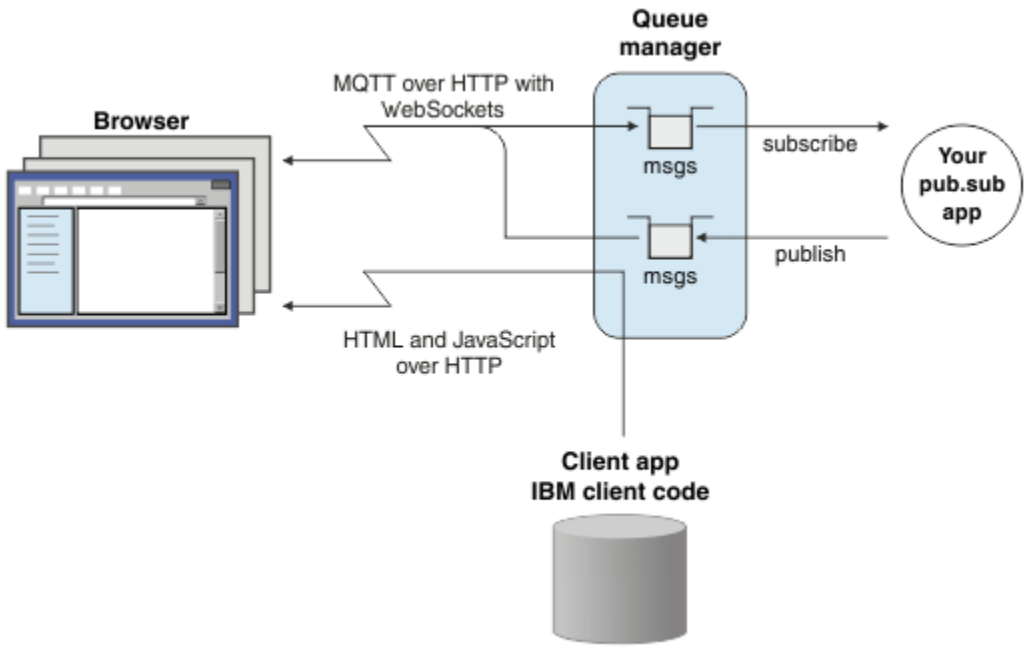


Figura 26. Utilización de Cliente MQTT de mensajería para JavaScript con la mensajería de publicación/suscripción

La aplicación web contiene lógica de aplicaciones y el URL del servidor MQTT. Cuando se abre en un navegador, la aplicación se conecta al servidor MQTT, crea las suscripciones que necesita y, a continuación, espera a recibir alertas orientadas a eventos y actúa sobre ellos.

La aplicación web se conecta utilizando MQTT como protocolo de transporte, sobre WebSockets. Los navegadores más modernos pueden realizar conexiones WebSockets. Utilizando WebSockets, la aplicación web puede pasar mensajes a través de cortafuegos que acepten HTTP y WebSocket protocol, y puede enviar paquetes de datos (conocidos como "frames") igual que utilizando TCP sobre IP.

Cuando un mensaje enviado por la aplicación web llega al servidor MQTT, la aplicación del lado del servidor lo considera sólo un mensaje. No sabe que el mensaje procede de un navegador.

Administración y control de un servidor MQTT

El servidor MQTT maneja la complejidad de la mensajería del lado del servidor. Garantiza la entrega de los mensajes que recibe de la aplicación web y aloja la aplicación de publicación/suscripción que responde a la aplicación web. Para cualquier servidor MQTT, debe completar los siguientes pasos:

- Cree un servidor.
- Elegir un puerto.
- Definir un nuevo canal MQTT.
- Configurar la aplicación web de cliente para que se conecte a los puertos elegidos a través del nuevo canal MQTT.

También tiene que enviar el JavaScript ejecutable de la aplicación web al navegador. Si utiliza IBM WebSphere MQ Telemetry, de forma predeterminada, el servidor MQTT lo hace por usted, utilizando el mismo canal MQTT que la aplicación web utiliza para conectarse al servidor MQTT. Si está haciendo pruebas con MQTT, esto puede ayudarle a empezar a trabajar rápidamente con MQTT. Para uso productivo, en particular en entornos de alto rendimiento, es posible que prefiera enviar el JavaScript ejecutable de la aplicación web en un canal aparte, utilizando un servidor de aplicaciones dedicado, como por ejemplo WebSphere Application Server.

Nota: Puesto que está diseñado para entornos de alto rendimiento, IBM MessageSight espera que lo haga.

Por ejemplo, si utiliza IBM WebSphere MQ Telemetry, utilizará el asistente **Nuevo canal de telemetría** de IBM WebSphere MQ Explorer para completar los pasos siguientes:

1. Cree un servidor.
2. Elegir un puerto (valor predeterminado: 1883).
3. Definir un nuevo canal MQTT.
4. Configurar la aplicación web de cliente para que se conecte a los puertos elegidos a través del nuevo canal MQTT.

El JavaScript ejecutable de la aplicación web (opcionalmente) también se envía a través del gestor de colas por el mismo canal. Para ello, el gestor de colas debe admitir tanto MQTT como HTTP. Si ya tiene un gestor de colas configurado para MQTT, puede utilizar la herramienta de línea de mandatos para cambiar el protocolo en la definición de canal para que admita tanto MQTT como HTTP. Consulte ALTER CHANNEL.

La aplicación web y la biblioteca de cliente Cliente MQTT de mensajería para JavaScript se almacenan en disco en una estructura que define el servidor de aplicaciones o el gestor de colas. Si utiliza IBM WebSphere MQ Telemetry, la aplicación web y la biblioteca de cliente se almacenan en la siguiente estructura de directorios:

```
MQINSTALL
  |
  mqxr
  |  |
  |  SDK
  |    |
  |    WebContent
  |       sample web app (the "Web Messaging Utility" sample HTML pages)
  |       |
  |       WebSocket
  |          IBM client library (the messaging client JavaScript classes)
  |
  qmgrs
     |
     qmgr_name
        |
        mqxr
           |
           WebContent
              |
              your_client_app (your own JavaScript pages)
```

La aplicación web de ejemplo y la biblioteca de clientes se almacenan en el directorio *MQINSTALL*/mqxr/SDK/WebContent. El material de este directorio procede de todos los gestores de colas. SI no desea que sus usuarios vean y utilicen todo este material, deberá crear su propia versión de la aplicación. Para hacer que esta aplicación, o su propia aplicación sustitutiva, esté disponible en gestores de colas específicos, debe poner la aplicación en el directorio *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent. Para seleccionar la app y las clases JavaScript asociadas para servir en un URL, el gestor de colas busca primero en su propio directorio WebContent y, a continuación, en el directorio WebContent global. En el árbol de directorios de ejemplo anterior, el gestor de colas ofrece *su_aplic_cliente* y la copia global de las clases JavaScript.

Para que el gestor de colas deje de ofrecer los archivos ejecutables de la aplicación web, o para modificar dónde debe buscar el gestor de colas los archivos ejecutables, debe configurar la propiedad **webcontentpath** y añadirla al archivo mqxr.properties. Consulte Propiedades de MQXR.

Conceptos relacionados

“Cómo programar aplicaciones de mensajería en JavaScript” en la página 123

Tareas relacionadas

“Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets” en la página 79
Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

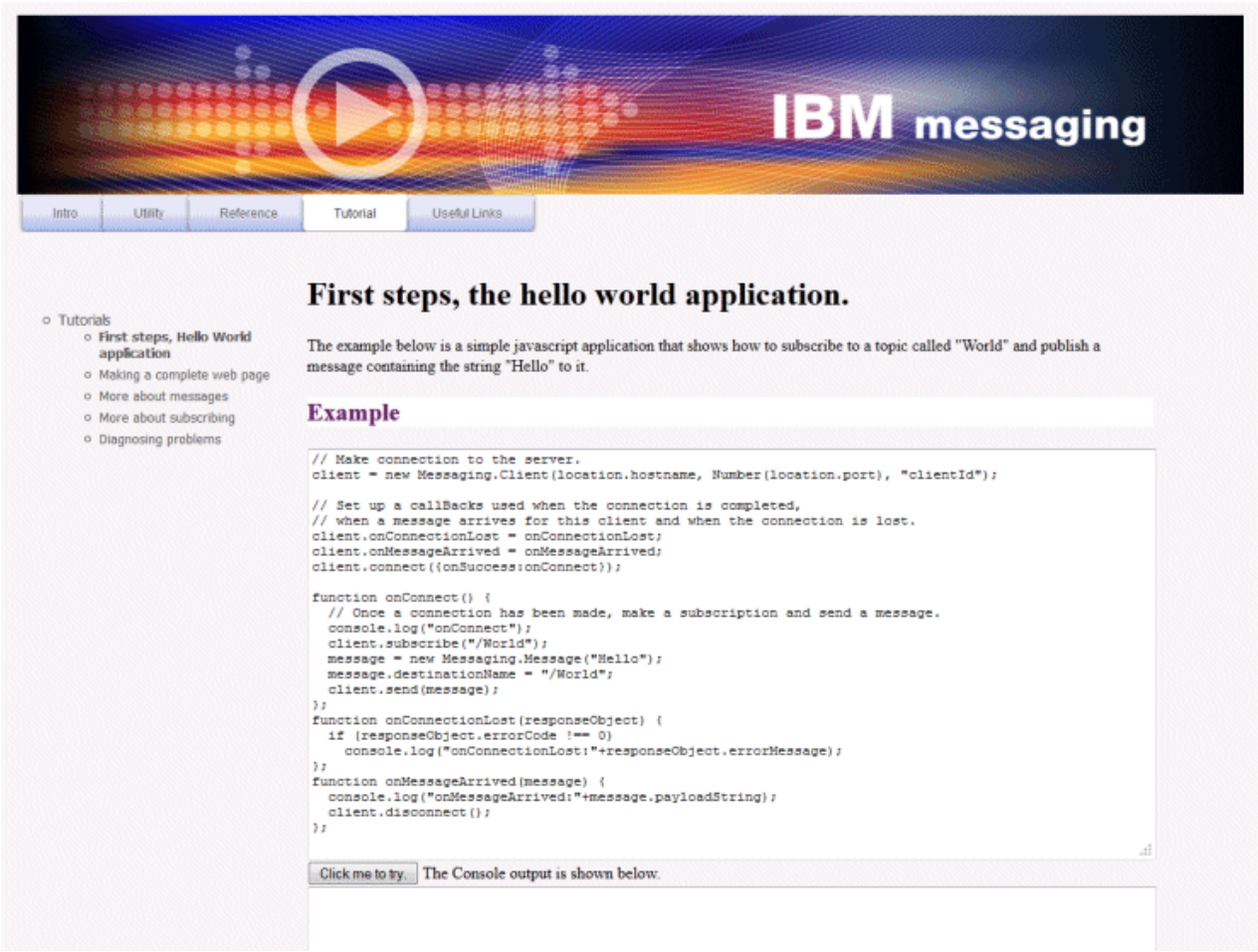
“Iniciación a Cliente MQTT de mensajería para JavaScript” en la página 24

Puede empezar a trabajar con Cliente MQTT de mensajería para JavaScript yendo a la página inicial de ejemplo del cliente de mensajería y examinando los recursos a los que enlaza. Para ver esta página de inicio, debe configurar un servidor MQTT para aceptar conexiones de la Páginas de JavaScript de ejemplo de cliente de mensajería MQTT, a continuación, escriba el URL que ha configurado en el servidor en un navegador web. Cliente MQTT de mensajería para JavaScript se inicia automáticamente en el dispositivo y se muestra la página inicial de ejemplo del cliente de mensajería. Esta página contiene enlaces a programas de utilidad, documentación de la interfaz de programación, una guía de aprendizaje y otra información útil.

Cómo programar aplicaciones de mensajería en JavaScript

Cliente MQTT de mensajería para JavaScript incluye una guía de aprendizaje que muestra cómo crear una aplicación web de publicación y suscripción simple. Explorando el código de las aplicaciones "First steps, Hello world", puede hacerse una idea básica de los mecanismos de la programación de aplicaciones web para mensajería.

Si su experiencia hasta la fecha ha sido principalmente en el desarrollo y despliegue de aplicaciones de mensajería tradicionales, también puede resultarle útil la sección “Consejos para la codificación en JavaScript” en la página 123. Si es usted un desarrollador experimentado pero nuevo en mensajería JavaScript, encontrará una breve introducción a los conceptos clave de mensajería en el apartado “Conceptos básicos de mensajería” en la página 126.



Consejos para la codificación en JavaScript

Si ya ha desarrollado aplicaciones de mensajería anteriormente, pero es nuevo en aplicaciones web, pueden resultarle útiles los siguientes consejos:

Envolver el código de cada evento en una devolución de llamada onSuccess

Cuando codifica una aplicación de mensajería, codifica de los siguientes eventos en el orden siguiente:

1. connect
2. suscribirse
3. publicar
4. receive message

La API de Cliente MQTT de mensajería para JavaScript es totalmente asíncrona, lo que significa que la hebra de la aplicación no se bloquea mientras espera que tengan efecto llamadas como connect o subscribe. En vez de eso, estas llamadas indican que se han completado llamando a una devolución de llamada `onSuccess` o `onFailure`. Para asegurarse de que cada suceso se ha completado antes de que se desencadene el siguiente suceso, debe envolver el código para cada suceso en una llamada `onSuccess`. Por ejemplo, la aplicación JavaScript podría volver de realizar la llamada 'connect' antes de que se hubiera creado la conexión. Para asegurarse de que la conexión ha tenido lugar antes de realizar la suscripción, debe poner el código para suscribirse en una devolución de llamada `onSuccess` de 'connect'.

El código de la aplicación "First steps, Hello world" utiliza este método.

Cómo incrustar el código de aplicación dentro de etiquetas HTML

A continuación se muestra un ejemplo de página JavaScript:

Y este sería el código de esta página, donde puede ver cómo el código de la aplicación está dentro de las etiquetas HTML:

```
<!DOCTYPE html>


<head>
  <script type="text/javascript" src="../WebSocket/mqttws31.js"></script>

  <script type="text/javascript">

  var client;
  var form = document.getElementById("tutorial");

  function doConnect() {
      client = new Messaging.Client("whistler1.hursley.ibm.com", 1883, "ClientId");
      client.onConnect = onConnect;
      client.onMessageArrived = onMessageArrived;
      client.onConnectionLost = onConnectionLost;
      client.connect({onSuccess:onConnect});
  }

  function doSubscribe() {
```

```
        client.subscribe("/World");
    }

    function doSend() {
        message = new Messaging.Message("Hello");
        message.destinationName = "/World";
        client.send(message);
    }

    function doDisconnect() {
        client.disconnect();
    }

    // Web Messaging API callbacks

    function onConnect() {
        var form = document.getElementById("example");
        form.connected.checked= true;
    }

    function onConnectionLost(responseObject) {
        var form = document.getElementById("example");
        form.connected.checked= false;
        if (responseObject.errorCode !== 0)
            alert(client.clientId+"\n"+responseObject.errorCode);
    }

    function onMessageArrived(message) {
        var form = document.getElementById("example");
        form.receiveMsg.value = message.payloadString;
    }

    </script>
</head>

<body>
    <h1>Example Web Messaging web page.</h1>
    <form id="example">
    <fieldset>
    <legend id="Connect" > Connect </legend>
      Make a connection to the server, and set up a call back used if a
      message arrives for this client.
      <br>
      <input type="button" value="Connect" onClick="doConnect(this.form)" name="Connect"/>
      <input type="checkbox" name="connected" disabled="disabled"/>
    </fieldset>

    <fieldset>
    <legend id="Subscribe" > Subscribe </legend>
    Make a subscription to topic "/World".
    <br> <input type="button" value="Subscribe" onClick="doSubscribe(this.form)"/>
    </fieldset>

    <fieldset>
    <legend id="Send" > Send </legend>
      Create a Message object containing the word "Hello" and then publish it at
      the server.
      <br>
      <input type="button" value="Send" onClick="doSend(this.form)"/>
    </fieldset>

    <fieldset>
    <legend id="Receive" > Receive </legend>
      A copy of the published Message is received in the callback we created earlier.
    <textarea name="receiveMsg" rows="1" cols="40" disabled="disabled"></textarea>
    </fieldset>

    <fieldset>
    <legend id="Disconnect" > Disconnect </legend>
      Now disconnect this client from the server.
    <br> <input type="button" value="Disconnect" onClick="doDisconnect()"/>
    </fieldset>
    </form>
</body>
</html>
```

Conceptos básicos de mensajería

A continuación se proporcionan información básicas sobre mensajería para desarrolladores de aplicaciones web que empiezan con la mensajería:

Mensajería asíncrona y de 'dispara y olvida'.

El protocolo MQTT da soporte a la entrega asegurada y a transferencias 'dispara y olvida'. En este protocolo, la entrega de mensajes es asíncrona: la aplicación pasa el mensaje a la API de cliente y no realiza ninguna acción adicional para asegurarse de que el mensaje se entregue. Este método se conoce como *dispara y olvida*. Cuando hay una respuesta disponible, se envía automáticamente a la aplicación.

La entrega asíncrona libera la aplicación de tener que mantener conexión con un servidor y de tener que estar a la espera de mensajes. El modelo de interacción es como en el correo electrónico, pero optimizado para la programación de aplicaciones.

Consulte también la sección "protocolo MQTT" de “Introducción a MQTT” en la página 5

Visión general de la mensajería de publicación/suscripción.

El proveedor de la información recibe el nombre de *publicador*. Un publicador proporciona información sobre un tema, sin necesidad de saber nada acerca de las aplicaciones que están interesadas en esa información. Un publicador elige un *tema*, que es un contenedor de mensajes sobre un asunto específico. A continuación, el publicador genera cada una de los ítems de información sobre ese asunto en forma de mensaje, que se denomina *publicación*, y lo publica en el tema asociado.

El consumidor de información se denomina *suscriptor*. Un suscriptor crea una *suscripción* a un tema en el que está interesado. Cuando se publica un mensaje nuevo en ese tema, se reenvía a todos los suscriptores al tema. Los suscriptores pueden hacer varias suscripciones y pueden recibir información de muchos publicadores diferentes.

Consulte también Introducción a la mensajería de publicación/suscripción de IBM WebSphere MQ.

Cómo se emparejan las suscripciones y los temas.

Si utiliza IBM WebSphere MQ como servidor MQTT, debe comprender cómo IBM WebSphere MQ especifica los temas. En IBM WebSphere MQ, un publicador crea un mensaje y lo publica con la serie de tema que mejor se adecua al asunto de la publicación. Para recibir publicaciones, un suscriptor crea una suscripción con una serie de tema que coincide con un patrón para seleccionar temas de publicación. El gestor de colas entrega publicaciones a los suscriptores que tienen suscripciones que coinciden con el tema de publicación y tienen autorización para recibir las publicaciones.

Los asuntos se organizan normalmente de forma jerárquica, en árboles de temas, utilizando el carácter '/' para crear subtemas en la serie de tema. Los temas son nodos del árbol de temas. Los temas pueden ser nodos hoja sin subtemas, o nodos intermedios con subtemas. Los suscriptores pueden utilizar caracteres comodín para suscribirse a más de un tema a la vez. Por ejemplo, una suscripción a `/sport/tennis` sólo obtiene mensajes publicados en el subtema de tenis, mientras que una suscripción a `/sport/#` obtiene mensajes publicados en cualquier subtema de `/sport`.

Consulte también Temas, Árboles de temas y Esquemas de comodín.

Conceptos relacionados

“El Cliente MQTT de mensajería para JavaScript y las aplicaciones web” en la página 119

Tareas relacionadas

“Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets” en la página 79
Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

“Iniciación a Cliente MQTT de mensajería para JavaScript” en la página 24
Puede empezar a trabajar con Cliente MQTT de mensajería para JavaScript yendo a la página inicial de ejemplo del cliente de mensajería y examinando los recursos a los que enlaza. Para ver esta página de inicio, debe configurar un servidor MQTT para aceptar conexiones de la Páginas de JavaScript de ejemplo de cliente de mensajería MQTT, a continuación, escriba el URL que ha configurado en el servidor en un navegador web. Cliente MQTT de mensajería para JavaScript se inicia automáticamente en el dispositivo y se muestra la página inicial de ejemplo del cliente de mensajería. Esta página contiene enlaces a

programas de utilidad, documentación de la interfaz de programación, una guía de aprendizaje y otra información útil.

Devoluciones de llamada y sincronización en aplicaciones cliente de MQTT

El modelo de programación de cliente MQTT utiliza hebras de forma extensiva. Las hebras desacoplan una aplicación cliente de MQTT , tanto como pueden, de los retardos en la transmisión de mensajes a y desde el servidor. Las publicaciones, las señales de entrega y los sucesos de pérdida de conexión se entregan a los métodos de una clase de devolución de llamada que implementa `MqttCallback`.

Devoluciones de llamadas

La interfaz de `MqttCallback` tiene tres métodos de devolución de llamada; consulte alguna implementación de ejemplo en Callback.java.

`connectionLost(java.lang.Throwable cause)`

Se invoca `connectionLost` cuando la conexión se cierra por un error de comunicaciones. También se invoca si el servidor cierra la conexión como resultado de un error en el servidor después de establecerse la conexión. Los errores del servidor se registran en el registro de errores del gestor de colas. El servidor cierra la conexión con el cliente y el cliente invoca `MqttCallback.connectionLost`.

Los únicos errores remotos que se lanzan como excepciones en la misma hebra que la aplicación cliente son excepciones de `MqttClient.connect`. Los errores que el servidor detecta una vez establecida la conexión se notifican al método callback `MqttCallback.connectionLost` como `throwables`.

Los errores de servidor habituales dan como resultado la invocación de `connectionLost` son los errores de autorización. Por ejemplo, el servidor de telemetría intenta publicar en un tema en nombre de un cliente que no tiene autorización para publicar en el tema. Todo lo que produzca la devolución de un código de condición `MQCC_FAIL` al servidor de telemetría puede dar como resultado el cierre de la conexión.

`deliveryComplete(MqttDeliveryToken token)`

El cliente de MQTT llama a `deliveryComplete` para devolver una señal de entrega a la aplicación cliente; consulte “Señales de entrega” en la página 131. Cuando se utiliza una señal de entrega, la devolución de llamada puede acceder al mensaje publicado mediante el método `token.getMessage`.

Cuando la devolución de llamada de la aplicación devuelve el control al cliente de MQTT después de ser invocada por el método `deliveryComplete`, la entrega se ha completado. Hasta que no se completa la entrega, la clase de persistencia retiene los mensajes con QoS 1 ó 2.

La llamada `deliveryComplete` constituye un punto de sincronización entre la aplicación y la clase de persistencia. Nunca se llama al método `deliveryComplete` dos veces para el mismo mensaje.

Cuando la devolución de llamada de aplicación vuelve de `deliveryComplete` al cliente MQTT , el cliente llama a `MqttClientPersistence.remove` para mensajes con QoS 1 o 2. `MqttClientPersistence.remove` suprime la copia almacenada localmente del mensaje publicado.

Desde una perspectiva de proceso de transacción, la llamada a `deliveryComplete` es una transacción de una sola fase que confirma la entrega. Si falla el proceso durante la devolución de llamada, al reiniciar el cliente se invoca de nuevo `MqttClientPersistence.remove` para suprimir la copia local del mensaje publicado. La devolución de llamada no se invoca de nuevo. Si utiliza la devolución de llamada para almacenar un registro de los mensajes entregados, puede sincronizar el registro con el cliente MQTT. Si desea almacenar un registro de forma segura, actualice el registro en la clase `MqttClientPersistence`.

La hebra principal de la aplicación y el cliente MQTT hacen referencia a la señal de entrega y al mensaje. El cliente MQTT anula la referencia al objeto `MqttMessage` cuando se ha completado la entrega, y al objeto de señal de entrega cuando el cliente se desconecta. El objeto `MqttMessage` puede recuperarse de la basura una vez que la entrega se ha completado si la aplicación cliente

anula la referencia al mismo. La señal de entrega puede ser eliminada por el recolector de basura una vez que se ha desconectado la sesión.

Puede obtener los atributos `MqttDeliveryToken` y `MqttMessage` una vez que se haya publicado un mensaje. Si intenta definir cualquier atributo `MqttMessage` después de que el mensaje se haya publicado, no puede definirse el resultado.

El cliente MQTT continúa procesando los acuses de recibo si el cliente se reconecta a la sesión anterior con el mismo `ClientIdentifier`; consulte "Limpiar sesiones" en la página 130. La aplicación cliente de MQTT debe establecer `MqttClient.CleanSession` en `false` para la sesión anterior y establecerla en `false` en la nueva sesión. El cliente de MQTT crea nuevas señales de entrega y objetos de mensaje en la nueva sesión para las entregas pendientes. Recupera los objetos utilizando la clase `MqttClientPersistence`. Si el cliente de la aplicación todavía tiene referencias a señales de entrega y mensajes antiguos, elimine las referencias a ellos. La devolución de llamada de aplicación se invoca en la nueva sesión para todas las entregas iniciadas en la sesión anterior y completadas en la sesión actual.

La devolución de llamada de aplicación se invoca después de que el cliente de aplicación se conecte, cuando se completa una entrega pendiente. Antes de que se conecte el cliente de la aplicación, puede recuperar entregas pendientes con el método `MqttClient.getPendingDeliveryTokens`.

Observe que la aplicación cliente originalmente ha creado el objeto de mensaje que se publica, y su matriz de bytes de carga. El cliente de MQTT hace referencia a estos objetos. El objeto de mensaje devuelto por la señal de entrega en el método `token.getMessage` no es necesariamente el mismo objeto de mensaje que creó el cliente. Si una nueva instancia de cliente MQTT vuelve a crear la señal de entrega, la clase `MqttClientPersistence` vuelve a crear el objeto `MqttMessage` . Por razones de coherencia, `token.getMessage` devuelve `null` si `token.isCompleted` está definido en `true`, independientemente de si el objeto de mensaje lo creó el cliente de la aplicación o la clase `MqttClientPersistence`.

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` se llama cuando una publicación llega al cliente que coincide con un tema de suscripción. `topic` es el tema de publicación, no el filtro de suscripción. Pueden ser diferentes si el filtro contiene comodines.

Si el tema coincide con varias suscripciones creadas por el cliente, este recibe varias copias de la publicación. Si un cliente publica en un tema al que también está suscrito, recibe una copia de su propia publicación.

Si se envía un mensaje con QoS igual a 1 o 2, el mensaje se almacena en la clase `MqttClientPersistence` antes de que el cliente MQTT llame a `messageArrived`. `messageArrived` se comporta como `deliveryComplete`: es invocado una sola vez para una publicación y `MqttClientPersistence.remove` elimina la copia local de la publicación cuando `messageArrived` devuelve el control al cliente MQTT. El cliente MQTT elimina sus referencias al tema y mensaje cuando `messageArrived` devuelve el control al cliente MQTT. Los objetos de tema y mensaje son eliminados por el recolector de basura si el cliente de aplicación no ha retenido una referencia a los objetos.

Devoluciones de llamada, hebras, y sincronización de aplicaciones cliente

El cliente MQTT llama a un método de devolución de llamada en una hebra distinta de la hebra de aplicación principal. La aplicación cliente no crea una hebra para la devolución de llamada, la crea el cliente de MQTT .

El cliente MQTT sincroniza los métodos de devolución de llamada. Se ejecuta una sola instancia del método de devolución de llamada cada vez. La sincronización permite actualizar fácilmente un objeto que cuenta las publicaciones que se han entregado. Se ejecuta una sola instancia de `MqttCallback.deliveryComplete` cada vez, por lo que es seguro actualizar el recuento sin realizar más sincronización. Es el mismo caso que cuando llega una sola publicación cada vez. El código del método `messageArrived` puede actualizar un objeto sin sincronizarlo. Si va a hacer una referencia al recuento o al objeto que se está actualizando en otra hebra, sincronice el recuento o el objeto.

La señal de entrega proporciona un mecanismo de sincronización entre la hebra de aplicación principal y la entrega de una publicación. El método `token.waitForCompletion` espera hasta que se completa una publicación específica o hasta que finaliza un tiempo de espera opcional. Puede utilizar `token.waitForCompletion` de dos formas sencillas para procesar una publicación en un momento dado:

1. Para detener el cliente de aplicaciones hasta que se haya completado la entrega de la publicación, consulte PubSync.java.
2. Para sincronizar con el método `MqttCallback.deliveryComplete`. Sólo cuando `MqttCallback.deliveryComplete` vuelve al cliente MQTT reanuda `token.waitForCompletion`. Utilizando este mecanismo puede sincronizar el código de ejecución en `MqttCallback.deliveryComplete` antes de que se ejecute el código en la hebra de la aplicación principal.

¿Qué ocurre si desea publicar sin esperar que se entregue cada publicación, pero desea confirmación cuando se hayan entregado todas las publicaciones? Si realiza la publicación en una única hebra, la última publicación que se envía es también la última publicación que se entrega.

Sincronización de solicitudes enviadas al servidor

Tabla 4. Comportamiento de la sincronización de métodos que tienen como resultado la creación de solicitudes para el servidor.

Esta tabla lista los métodos del cliente Java de MQTT que envían una solicitud al servidor. Para cada método, en la tabla se describen las condiciones bajo las cuales el método espera o vuelve, y el tiempo que el método espera.

Método	Sincronización	Intervalo de tiempo de espera
`MqttClient.Connect`	Espera a que se establezca una conexión con el servidor.	30 segundos de forma predeterminada, o según se haya establecido en un parámetro.
`MqttClient.Disconnect`	Espera a que el cliente MQTT finalice el trabajo que debe hacer y que la sesión TCP/IP se desconecte.	30 segundos de forma predeterminada, o según se haya establecido en un parámetro.
`MqttClient.Subscribe`	Espera hasta que se completa la solicitud de suscripción.	30 segundos de forma predeterminada, o según se haya establecido en un parámetro.
`MqttClient.UnSubscribe`	Espera hasta que se completa la solicitud de anulación de suscripción.	30 segundos de forma predeterminada, o según se haya establecido en un parámetro.
`MqttClient.Publish`	Devuelve inmediatamente el control a la hebra de aplicación después de pasar la solicitud al cliente MQTT.	Ninguno.
`MqttDeliveryToken.waitForCompletion`	Espera a que se devuelva la señal de entrega.	Indefinido de forma predeterminada, o según se haya establecido en un parámetro.

Limpiar sesiones

El cliente MQTT y el servicio de telemetría (MQXR) mantienen la información del estado de la sesión. La información de estado se utiliza para garantizar la entrega "al menos una vez" y "exactamente una vez" y la recepción "exactamente una vez" de publicaciones. El estado de la sesión también incluye las suscripciones que ha creado un cliente MQTT. Puede elegir ejecutar un cliente MQTT manteniendo, o sin mantener, la información de estado entre sesiones. Cambie la modalidad de sesión limpia estableciendo `MqttConnectOptions.cleanSession` antes de conectarse.

Cuando conecte una aplicación cliente de MQTT mediante el método `MqttClient.connect`, el cliente identifica la conexión utilizando el identificador de cliente y la dirección del servidor. El servidor comprueba si se ha guardado información de sesión procedente de una conexión anterior con el servidor. Si todavía existe una sesión anterior y se ha especificado `cleanSession=true`, se borra la información de la sesión anterior en el cliente y en el servidor. Si `cleanSession=false`, se reanuda la sesión anterior. Si no existe ninguna sesión anterior, se inicia una nueva.

Nota: el administrador de WebSphere MQ puede forzar el cierre de una sesión abierta y suprimir toda la información sobre la misma. Si el cliente reabre la sesión con la opción `cleanSession=false`, se inicia una sesión nueva.

Publicaciones

Si utiliza el valor predeterminado `MqttConnectOptions`, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar con el cliente, se eliminan todas las entregas de publicaciones pendientes para el cliente cuando éste se conecta.

El valor de limpiar sesión no afecta a las publicaciones enviadas con `QoS=0`. Para `QoS=1` y `QoS=2`, la utilización de `cleanSession=true` puede provocar la pérdida de una publicación.

Suscripciones

Si utiliza el `MqttConnectOptions`predeterminado, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar el cliente, las suscripciones antiguas para el cliente se eliminan cuando se conecta el cliente. Las suscripciones nuevas que el cliente realice durante la sesión se eliminan cuando se desconecta.

Si establece `MqttConnectOptions.cleanSession` en `false` antes de conectarse, las suscripciones que crea el cliente se añaden a todas las suscripciones que existían para el cliente antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otro modo de entender la forma en la que el atributo `cleanSession` afecta a las suscripciones es pensar en él como un atributo modal. Con la modalidad predeterminada, `cleanSession=true`, el cliente crea suscripciones y recibe publicaciones sólo dentro del ámbito de la sesión. En la modalidad alternativa, `cleanSession=false`, las suscripciones son duraderas. El cliente puede conectarse y desconectarse y las suscripciones permanecen activas. Cuando el cliente vuelve a conectarse, recibe las publicaciones que no se hayan entregado. Mientras está conectado, puede modificar el conjunto de suscripciones que estén activas en su nombre.

Debe establecer la modalidad `cleanSession` antes de conectarse; la modalidad dura toda la sesión. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de utilización de `cleanSession=false` a `cleanSession=true`, se descartarán todas las suscripciones anteriores para el cliente y las publicaciones que no se hayan recibido.

Identificador de cliente

El identificador de cliente es una serie de 23 bytes que identifica un cliente MQTT . El identificador de cliente debe ser exclusivo entre todos los clientes que se conectan al servidor y no debe ser el mismo que el nombre del gestor de colas en el servidor. Dentro de estas restricciones, puede utilizar cualquier serie de identificación. Es importante contar con un procedimiento para asignar identificadores de cliente y un medio de configurar un cliente con el identificador que se haya elegido para el mismo.

El identificador de cliente se utiliza en la administración de un sistema MQTT. Con cientos de miles de clientes potenciales que administrar, es necesario poder identificar un cliente concreto rápidamente. Supongamos por ejemplo, que un dispositivo no ha funcionado bien y se lo notifica al usuario un cliente que llama al centro de atención al cliente. ¿Cómo identifica el cliente el dispositivo y cómo correlaciona el usuario dicha identificación con el servidor que está conectado al cliente normalmente? ¿Debe consultar una base de datos en la que cada dispositivo está relacionado con un identificador de cliente y un servidor? ¿El nombre del dispositivo identifica a qué servidor está conectado? Cuando examina conexiones de clientes MQTT, todas las conexiones están etiquetadas con el identificador de cliente. ¿Necesita mirar una tabla para correlacionar un identificador de cliente con un dispositivo físico?

¿El identificador de cliente identifica a un dispositivo específico, a un usuario o a una aplicación que se ejecuta en el cliente? Si un cliente reemplaza un dispositivo defectuoso por uno nuevo, ¿el nuevo tiene el mismo identificador que el antiguo? ¿Asigna un identificador nuevo? Si cambia un dispositivo físico, pero mantiene el mismo identificador, las publicaciones pendientes y las suscripciones activas se transfieren automáticamente al nuevo dispositivo.

¿Cómo se garantiza la exclusividad de los identificadores de cliente? Al igual que ocurre con un sistema que genera identificadores exclusivos, el usuario debe contar con un proceso fiable para establecer el identificador en el cliente. Es posible que el dispositivo del cliente sea una "caja negra" sin interfaz de usuario. ¿Se fabrica el dispositivo con un identificador de cliente como, por ejemplo, utilizando la dirección MAC? ¿O cuenta con un proceso de configuración e instalación de software que configura el dispositivo antes de que se active?

Puede crear un identificador de cliente a partir de una dirección MAC de dispositivo de 48 bits para que el identificador siga siendo corto y exclusivo. Si el tamaño de la transmisión no es algo que resulte crítico, puede utilizar los 17 bytes restantes para hacer que la dirección sea más fácil de administrar.

Señales de entrega

Cuando un cliente publica en un tema se crea una nueva señal de entrega. Utilice la señal de entrega para supervisar la entrega de una publicación o bloquear la aplicación cliente hasta que se complete la entrega.

La señal es un objeto `MqttDeliveryToken`. Se crea al llamar al método MqttTopic.publish() y la retiene el cliente MQTT hasta que la sesión de cliente se desconecta y se completa la entrega.

La señal se utiliza normalmente para comprobar si se ha completado la entrega. Bloquee la aplicación cliente hasta que se complete la entrega utilizando la señal que se devuelve para llamar a `token.waitForCompletion`. Como alternativa, puede proporcionar un controlador `MqttCallBack`. Cuando el cliente MQTT recibe todos los acuses de recibo que espera como parte de la entrega de una publicación, llama al método `MqttCallBack.deliveryComplete`, pasando la señal de entrega como parámetro.

Hasta que se complete la entrega, puede examinar la publicación mediante la señal de entrega devuelta llamando a `token.getMessage`.

Entregas completadas

La finalización de las entregas es asíncrona, y depende de la calidad de servicio asociada a la publicación.

Como máximo una vez

`QoS=0`

La entrega se completa inmediatamente tras la devolución por parte de `MqttTopic.publish`. Se llama inmediatamente a `MqttCallback.deliveryComplete`.

Al menos una vez

`QoS=1`

La entrega se completa cuando se recibe en la publicación un acuse de recibo proceden del gestor de colas. Cuando se recibe el acuse de recibo, se llama inmediatamente a `MqttCallback.deliveryComplete`. Es posible que el mensaje se entregue más de una vez

antes de que se llame a `MqttCallback.deliveryComplete`, si la comunicación es lenta o no es fiable.

Exactamente una vez

QoS=2
La entrega se completa cuando el cliente recibe un mensaje de finalización indicando que la publicación se ha publicado para los suscriptores. Se llama a `MqttCallback.deliveryComplete` tan pronto como se recibe el mensaje de publicación. No espera al mensaje que indica la finalización.

En circunstancias excepcionales, es posible que la app de cliente no vuelva al cliente de MQTT desde `MqttCallback.deliveryComplete` normalmente. Sabrá que la entrega se ha completado porque se ha invocado `MqttCallback.deliveryComplete`. Si el cliente reinicia la misma sesión, no se llama de nuevo a `MqttCallback.deliveryComplete`.

Entregas incompletas

Si la entrega no se ha completado después de que la sesión de cliente se desconecte, puede conectar el cliente de nuevo y completar la entrega. Sólo puede completar la entrega de un mensaje si éste se publica en una sesión con el atributo `MqttConnectionOptions` establecido en `false`.

Cree el cliente utilizando el mismo identificador de cliente y la dirección de servidor, y conéctelo entonces estableciendo el atributo `cleanSession` `MqttConnectionOptions` de nuevo en `false`. Si establece `cleanSession` en `true`, se descartan las señales de entrega pendientes.

Puede comprobar si existe alguna entrega pendiente llamando a `MqttClient.getPendingDeliveryTokens`. Puede llamar a `MqttClient.getPendingDeliveryTokens` antes de conectar el cliente.

Publicación de última voluntad y testamento

Si una conexión de cliente MQTT finaliza de forma inesperada, puede configurar WebSphere MQ Telemetry para enviar una publicación de "última voluntad y testamento". Defina de antemano el contenido de la publicación y el tema al que enviarla. La "última voluntad y testamento" es una propiedad de la conexión. Créela antes de conectar el cliente.

Cree un tema para la última voluntad y testamento. Puede crear un tema como, por ejemplo, `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Configure una "última voluntad y testamento" utilizando el método `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Considere la posibilidad de crear una indicación de la hora en el mensaje `lastWillPayload`. Incluya otra información de cliente que ayude a identificar el cliente y las circunstancias de la conexión. Pase el objeto `MqttConnectionOptions` al constructor `MqttClient`.

Establezca `lastWillQos` en 1 o 2, para hacer que el mensaje sea persistente en IBM WebSphere MQ y para asegurar su entrega. Para que se conserve la información de la última conexión perdida, establezca `lastWillRetained` en `true`.

La publicación "última voluntad y testamento" se envía a los suscriptores si la conexión finaliza de forma inesperada. Se envía si la conexión finaliza sin que el cliente llame al método `MqttClient.disconnect`.

Para supervisar las conexiones, complemente la publicación "última voluntad y testamento" con otras publicaciones para registrar las conexiones y desconexiones programadas.

Persistencia de mensajes en clientes MQTT

Los mensajes de publicaciones son persistentes si se envían con una calidad de servicio "al menos una vez" o "exactamente una vez". Puede implementar su propio mecanismo de persistencia en el cliente

o utilizar el mecanismo de persistencia predeterminado proporcionado por el cliente. La persistencia funciona en ambas direcciones, para publicaciones enviadas a o desde el cliente.

En MQTT, la persistencia de mensajes tiene dos aspectos; el modo en que se transfiere el mensaje y si se pone en cola en el servidor de MQTT como un mensaje persistente.

1. El cliente MQTT asocia la persistencia de mensaje con la calidad de servicio. Dependiendo de la calidad de servicio que elija para un mensaje, el mensaje pasa a ser persistente. La persistencia de mensaje es necesaria para implementar la calidad de servicio necesaria.

   Si se especifica "como máximo una vez", `QoS=0`, el cliente rechaza el mensaje tan pronto como se publique. Si existe algún error en las comunicaciones en sentido ascendente, el mensaje no se vuelve a enviar. Incluso aunque el cliente permanezca activo, el mensaje no se vuelve a enviar. El comportamiento de los mensajes con `QoS=0` es el mismo que para lo mensajes no persistentes rápidos de IBM WebSphere MQ.

   Si un cliente publica un mensaje con `QoS` establecido en a 1 ó 2, el mensaje pasa a ser persistente. El mensaje se almacena localmente y sólo se descarta del cliente cuando ya no es necesario para garantizar la entrega "al menos una vez", `QoS=1`,o "exactamente una vez", `QoS=2`.
2. Si un mensaje está marcado como `QoS` 1 ó 2, se pone en cola como un mensaje persistente. Si está marcado como `QoS=0`, se pone en cola como un mensaje no persistente. En IBM WebSphere MQ , los mensajes no persistentes se transfieren entre gestores de colas "exactamente una vez", a menos que el canal de mensajes tenga el atributo `NPMSPEED` establecido en `FAST`.

Una publicación persistente se almacena en el cliente hasta que la recibe una aplicación cliente. Para `QoS=2`, la publicación se descarta del cliente cuando la devolución de llamada de la aplicación devuelve el control. Para `QoS=1` la aplicación puede volver a recibir la publicación si se produce un error. Para `QoS=0`, la devolución de llamada recibe la publicación no más de una vez. Es posible que no reciba la publicación si existe un error o si el cliente se desconecta en el momento de la publicación.

Cuando se suscribe a un tema, puede reducir la `QoS` con la que el suscriptor recibe mensajes para que la calidad de servicio sea conforme con los recursos de persistencia del suscriptor. Las publicaciones que se crean con una `QoS` mayor se envían con la `QoS` más alta que solicita el suscriptor.

Almacenamiento de mensajes

La implementación del almacenamiento de datos en dispositivos pequeños varía mucho. El método para guardar temporalmente mensajes persistentes en un espacio de almacenamiento gestionado por el cliente MQTT puede ser demasiado lento o exigir demasiado espacio de almacenamiento. En los dispositivos móviles, el sistema operativo para dispositivos móviles puede proporcionar un servicio de almacenamiento que es ideal para los mensajes de MQTT.

Para proporcionar flexibilidad y tener en cuenta las limitaciones de los dispositivos pequeños, el cliente MQTT tiene dos interfaces de persistencia. Las interfaces definen las operaciones que intervienen en el almacenamiento de mensajes persistentes. Las interfaces se describen en la documentación de la API para el Cliente MQTT para Java. Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT. Puede implementar las interfaces para que se adapten a un dispositivo. El cliente MQTT que se ejecuta en Java SE tiene una implementación predeterminada de las interfaces que almacenan mensajes persistentes en el sistema de archivos. Esta implementación utiliza el paquete `java.io`. El cliente también tiene una implementación predeterminada para Java ME, `MqttDefaultMIDPPersistence`.

Clases de persistencia

`MqttClientPersistence`
: Esta clase pasa una instancia de la implementación de `MqttClientPersistence` al cliente MQTT como parámetro del constructor `MqttClient`. Si omite el parámetro `MqttClientPersistence` del constructor `MqttClient` , el cliente MQTT almacena mensajes persistentes utilizando la clase `MqttDefaultFilePersistence` o `MqttDefaultMIDPPersistence`.

MqttPersistable
MqttClientPersistence obtiene y coloca objetos MqttPersistable mediante una clave de almacenamiento. Debe proporcionar una implementación de MqttPersistable, así como la implementación de MqttClientPersistence si no va a utilizar MqttDefaultFilePersistence ni MqttDefaultMIDPPersistence.

MqttDefaultFilePersistence

El cliente MQTT proporciona la clase MqttDefaultFilePersistence. Si crea una instancia de MqttDefaultFilePersistence en su aplicación cliente, puede proporcionar el directorio para almacenar mensajes persistentes como un parámetro del constructor MqttDefaultFilePersistence.

Como alternativa, el cliente MQTT puede crear una instancia de MqttDefaultFilePersistence y colocar archivos en un directorio predeterminado. El nombre del directorio es *client identifier*-tcp *hostname portnumber*. "\", "\\", "/", ":" y " " se eliminan de la serie de nombre de directorio.

La vía de acceso del directorio es el valor de la propiedad del sistema rcp.data. Si no se ha establecido rcp.data, la vía de acceso es el valor de la propiedad del sistema usr.data.

rcp.data es una propiedad asociada a la instalación de OSGi o Eclipse Rich Client Platform (RCP).

usr.data es el directorio en el que se ha iniciado el mandato Java que ha iniciado la aplicación.

MqttDefaultMIDPPersistence
MqttDefaultMIDPPersistence tiene un constructor predeterminado y ningún parámetro. Utiliza el paquete javax.microedition.rms.RecordStore para almacenar mensajes.

Publicaciones

Las publicaciones son instancias de MqttMessage que están asociadas con una serie de tema. El cliente de MQTT puede crear publicaciones para enviar a IBM WebSphere MQy suscribirse a temas en IBM WebSphere MQ MQ para recibir publicaciones.

Un MqttMessage tiene una matriz de bytes como carga útil. Intente mantener los mensajes lo más pequeños posible. La longitud máxima de mensaje que permite el protocolo de MQTT es 250 MB.

Generalmente, un programa cliente MQTT utiliza java.lang.String o java.lang.StringBuffer para manipular el contenido de los mensajes. Para su comodidad, MqttMessage utiliza un método toString para convertir la carga en una serie. Para crear una carga útil de matriz de bytes a partir de un java.lang.String o java.lang.StringBuffer, utilice el método getBytes.

El método getBytes convierte una serie al conjunto de caracteres predeterminado de la plataforma. El conjunto de caracteres predeterminado es normalmente UTF-8. La publicaciones de MQTT que contienen sólo texto, generalmente están en UTF-8. Utilice el método getBytes("UTF8") para anular temporalmente el conjunto de caracteres predeterminado.

En IBM WebSphere MQ, una publicación de MQTT se recibe como un mensaje de jms-bytes. El mensaje incluye una carpeta MQRFH2 que contiene una carpeta <mqtt> y una carpeta <mqps>. La carpeta <mqtt> contiene clientId y qos, pero este contenido puede cambiar en el futuro.

Un MqttMessage tiene tres atributos adicionales: la calidad de servicio, un indicador de retención y un indicador de duplicado. El indicador de duplicado se define sólo si la calidad de servicio es "al menos una vez" o "exactamente una vez". Si el mensaje se ha enviado anteriormente y el cliente MQTT no lo ha reconocido lo suficientemente rápido, el mensaje se vuelve a enviar con el atributo de duplicado definido en true.

En publicación

Para crear una publicación en una aplicación cliente de MQTT , cree un MqttMessage. Defina la carga útil, la calidad de servicio y un indicador de retención para el mensaje e invoque el método MqttTopic.publish(MqttMessage message); se devuelve MqttDeliveryToken y la finalización de la publicación es asíncrona.

Como alternativa, el cliente MQTT puede crear automáticamente un objeto de mensaje temporal a partir de los parámetros del método `MqttTopic.publish(byte [] payload, int qos, boolean retained)` cuando crea una publicación.

Si la calidad de servicio de la publicación es "al menos una vez" o "exactamente una vez", `QoS=1` o `QoS=2`, el cliente MQTT invoca la interfaz `MqttClientPersistence`. Llama a `MqttClientPersistence` para almacenar el mensaje antes de devolver una señal de entrega a la aplicación.

La aplicación puede elegir bloquear hasta que el mensaje se haya entregado al servidor, utilizando el método `MqttDeliveryToken.waitForCompletion`. Como alternativa, la aplicación puede continuar sin realizar bloqueo. Si desea comprobar que las publicaciones se entregan, sin realizar un bloqueo, registre una instancia de una clase de devolución de llamada que implemente `MqttCallback` con el cliente MQTT. El cliente MQTT llama al método `MqttCallback.deliveryComplete` tan pronto como se entrega la publicación. Según la calidad de servicio, la entrega puede ser casi inmediata con `QoS=0` o puede tardar un poco con `QoS=2`.

Utilice el método `MqttDeliveryToken.isComplete` para averiguar si la entrega se ha completado. Mientras el valor de `MqttDeliveryToken.isComplete` sea `false`, puede llamar a `MqttDeliveryToken.getMessage` para obtener el contenido del mensaje. Si el resultado al llamar a `MqttDeliveryToken.isComplete` es `true`, se ha rechazado el mensaje y al llamar a `MqttDeliveryToken.getMessage` debe aparecer una excepción de puntero nulo. No existe sincronización generada entre `MqttDeliveryToken.getMessage` y `MqttDeliveryToken.isComplete`.

Si el cliente se desconecta antes de recibir todas las señales de entrega pendientes, con una nueva instancia de cliente se puede consultar las señales de entrega pendientes antes de conectarse. Hasta que el cliente se conecte, no se completa ninguna nueva entrega y es seguro llamar a `MqttDeliveryToken.getMessage`. Utilice el método `MqttDeliveryToken.getMessage` para averiguar qué publicaciones no se han entregad aún. Las señales de entrega pendientes se descartan si se conecta con `MqttConnectOptions.cleanSession` definido en `true`, su valor predeterminado.

Suscripción

Un gestor de colas o un IBM MessageSight es el responsable de crear las publicaciones para enviar a un suscriptor de MQTT. El gestor de colas comprueba si el filtro de temas de una suscripción creada por un cliente MQTT coincide con la serie de tema de una publicación. La coincidencia puede ser exacta o incluir comodines. Antes de reenviar la publicación al suscriptor con el gestor de colas, este comprueba los atributos de temas asociados a la publicación. Se sigue el procedimiento de búsqueda que se describe en Suscripción utilizando una serie de tema que contiene caracteres comodín para identificar si un objeto de tema administrativo otorga al usuario la autorización para suscribirse.

Cuando el cliente MQTT recibe una publicación con una calidad de servicio de "al menos una vez", llama al método `MqttCallback.messageArrived` para procesar la publicación. Si la calidad de servicio de la publicación es "exactamente una vez", `QoS=2`, el cliente MQTT llama a la interfaz `MqttClientPersistence` para almacenar el mensaje cuando se reciba. A continuación llama a `MqttCallback.messageArrived`.

Calidades de servicio proporcionadas por un cliente MQTT

Un cliente MQTT proporciona tres calidades de servicio para entregar publicaciones a WebSphere MQ y al cliente de MQTT : "como máximo una vez", "como mínimo una vez" y "exactamente una vez". Cuando un cliente MQTT envía una solicitud a IBM WebSphere MQ para crear una suscripción, la solicitud se envía con la calidad de servicio "al menos una vez".

La calidad de servicio de una publicación es un atributo de `MqttMessage`. Es establecido por el método `MqttMessage.setQos`.

El método `MqttClient.subscribe` puede reducir la calidad de servicio que se aplica a las publicaciones enviadas a un cliente sobre un tema. La calidad de servicio de una publicación que se reenvía a un suscriptor puede ser diferente a la de la publicación. Para reenviar una publicación se utiliza el valor más bajo de los dos.

Como máximo una vez

QoS=0

El mensaje se entrega como máximo una vez, si no, no se entrega. No se efectúa acuse de recibo la entrega del mensaje por la red.

El mensaje no se almacena. El mensaje puede perderse si se desconecta el cliente o si falla el servidor.

QoS=0 es la modalidad de transferencia más rápida. Se denomina a veces "transmitir y olvidar".

El protocolo MQTT no necesitan servidores para reenviar publicaciones con un valor de QoS=0 a un cliente. Si el cliente está desconectado cuando el servidor recibe la publicación, es posible que se descarte la publicación, dependiendo del servidor. El servicio de telemetría (MQXR) no descarta los mensajes enviados con QoS=0. Se almacenan como mensajes no persistentes y sólo se rechazan si se detiene el gestor de colas.

Al menos una vez

QoS=1

QoS=1 es el valor predeterminado la modalidad de transferencia.

El mensaje siempre se entrega, como mínimo, una vez. Si el emisor no recibe un acuse de recibo, el mensaje se envía de nuevo con el distintivo DUP establecido hasta que se reciba un acuse de recibo. Como consecuencia, se puede enviar al receptor el mismo elemento varias veces, y que se procese varias veces.

El mensaje debe almacenarse localmente en el emisor y el receptor hasta que se procese.

El mensaje se suprime del receptor después de que se haya procesado. Si el receptor es un intermediario, el mensaje se publica a sus suscriptores. Si el receptor es un cliente, el mensaje se entrega a la aplicación de suscriptor. Una vez suprimido el mensaje, el receptor envía un acuse de recibo al emisor.

El mensaje se suprime del emisor después de que se haya recibido el acuse de recibo por parte del receptor.

Exactamente una vez

QoS=2

El mensaje se entrega siempre exactamente una vez.

El mensaje debe almacenarse localmente en el emisor y el receptor hasta que se procese.

QoS=2 es la modalidad de transferencia más segura, pero la más lenta. Deben realizarse como mínimo dos pares de transmisiones entre el emisor y el receptor antes de que el mensaje pueda suprimirse de la parte del emisor. El mensaje puede procesarse en el receptor tras la primera transmisión.

En el primer par de transmisiones, el emisor transmite el mensaje y obtiene acuse de recibo del receptor que ha almacenado el mensaje. Si el emisor no recibe un acuse de recibo, el mensaje se envía de nuevo con el distintivo DUP establecido hasta que se reciba un acuse de recibo.

En el segundo par de transmisiones, el emisor comunica al receptor que puede completar el proceso del mensaje, "PUBREL". Si el emisor no recibe acuse de recibo del mensaje "PUBREL", el mensaje "PUBREL" se envía de nuevo hasta que se recibe aquél. El emisor suprime el mensaje que guardó al recibir el acuse de recibo para el mensaje "PUBREL".

El receptor puede procesar el mensaje proporcionado en la primera o segunda fase, no tiene que volver a procesarlo. Si el receptor es un intermediario, publica el mensaje a los suscriptores. Si el receptor es un cliente, el mensaje se entrega a la aplicación de suscriptor. El receptor devuelve un mensaje de finalización al emisor para comunicarle que ha terminado de procesar el mensaje.

Publicaciones retenidas y clientes MQTT

Si crea una suscripción a un tema que tenga una publicación retenida, se le reenviará inmediatamente la publicación retenida más reciente en el tema.

Utilice el método `MqttMessage.setRetained` para especificar si se retiene o no una publicación en un tema.

Para suprimir una publicación retenida en IBM WebSphere MQ, ejecute el mandato MQSC CLEAR CLEAR TOPICSTR**CLEAR TOPICSTR**.

Si crea una publicación con una carga útil nula, la publicación vacía se envía a los suscriptores. Es posible que otros intermediarios de MQTT no reenvíen una publicación vacía a los suscriptores.

Si publica una publicación no retenida en un tema que tiene una publicación retenida, la publicación retenida no se ve afectada. Los suscriptores actuales reciben la nueva publicación. Los suscriptores nuevos reciben la publicación retenida primero y luego las nuevas.

Cuando cree o actualice una publicación retenida, envíe la publicación con una QoS o 1 o 2. Si lo envía con una QoS de 0, IBM WebSphere MQ crea una publicación retenida no persistente. La publicación no se retiene si se detiene el gestor de colas.

Utilice publicaciones retenidas para registrar el valor más reciente de una medida. Los nuevos suscriptores al tema retenido reciben inmediatamente el valor más reciente de la medida. Si no se toman nuevas medidas desde que el suscriptor se suscribió por última vez al tema de publicación, y si el suscriptor vuelve a suscribirse, el suscriptor recibe de nuevo la publicación retenida más reciente sobre el tema.

Suscripciones

Cree suscripciones para registrar un interés en temas de publicación mediante un filtro de tema. Un cliente puede crear varias suscripciones o una suscripción que contenga un filtro de tema con asteriscos, para registrarse en varios temas. Las publicaciones en temas que coinciden con los filtros se envían al cliente. Las suscripciones pueden permanecer activas mientras un cliente está desconectado. Las publicaciones se envían al cliente cuando vuelve a conectarse.

Cree suscripciones utilizando los métodos `MqttClient.subscribe`, pasando uno o varios filtros de temas y parámetros de calidades de servicio. El parámetro de calidad de servicio establece la calidad de servicio máxima que el suscriptor puede utilizar para recibir un mensaje. Los mensajes enviados a este cliente no pueden entregarse con una calidad de servicio mayor. La calidad de servicio se establece en el valor original cuando el mensaje se publicó o en el nivel especificado para la suscripción, el valor que sea más bajo de los dos. La calidad de servicio predeterminada para recibir mensajes es `QoS=1`, al menos una vez.

La propia solicitud de suscripción se envía con `QoS=1`.

Un suscriptor recibe las publicaciones cuando el cliente MQTT llama al método `MqttCallback.messageArrived`. El método `messageArrived` también pasa la serie del tema con la que el mensaje se ha publicado al suscriptor.

Puede eliminar una suscripción o conjunto de definiciones utilizando los métodos `MqttClient.unsubscribe`.

Un mandato de WebSphere MQ puede eliminar una suscripción.Listar suscripciones utilizando WebSphere MQ Explorer, o utilizando **runmqsc** o mandatos PCF. A todas las suscripciones de cliente MQTT se les asigna un nombre. Se les asigna un nombre con el formato: *`ClientIdentifier`*`:`*Topic name*

Si utiliza el `MqttConnectOptions`predeterminado, o establece `MqttConnectOptions.cleanSession` en `true` antes de conectar el cliente, las suscripciones antiguas para el cliente se eliminan cuando se conecta el cliente. Las suscripciones nuevas que el cliente realice durante la sesión se eliminan cuando se desconecta.

Si establece `MqttConnectOptions.cleanSession` en `false` antes de conectarse, las suscripciones que crea el cliente se añaden a todas las suscripciones que existían para el cliente antes de conectarse. Cuando el cliente se desconecta, permanecen activas todas las suscripciones.

Otro modo de entender la forma en la que el atributo `cleanSession` afecta a las suscripciones es pensar en él como un atributo modal. Con la modalidad predeterminada, `cleanSession=true`, el cliente crea suscripciones y recibe publicaciones sólo dentro del ámbito de la sesión. En la modalidad alternativa, `cleanSession=false`, las suscripciones son duraderas. El cliente puede conectarse y desconectarse y las suscripciones permanecen activas. Cuando el cliente vuelve a conectarse, recibe las publicaciones

que no se hayan entregado. Mientras está conectado, puede modificar el conjunto de suscripciones que estén activas en su nombre.

Debe establecer la modalidad `cleanSession` antes de conectarse; la modalidad dura toda la sesión. Para cambiar este valor, debe desconectar el cliente y volverlo a conectar. Si cambia las modalidades de utilización de `cleanSession=false` a `cleanSession=true`, se descartarán todas las suscripciones anteriores para el cliente y las publicaciones que no se hayan recibido.

Las publicaciones que coinciden con las suscripciones activas se envían al cliente tan pronto como se publiquen. Si el cliente está desconectado, se envían al cliente si se vuelve a conectar al mismo servidor con el mismo identificador de cliente y `MqttConnectOptions.cleanSession` establecido en `false`.

Las suscripciones de un cliente determinado se identifican por el identificador de cliente. Puede volver a conectar el cliente desde un dispositivo del cliente diferente al mismo servidor y continuar con las mismas suscripciones y recibir publicaciones que no se hayan entregado.

Series de tema y filtros de tema en clientes MQTT

Las series de temas y los filtros de temas se utilizan para las publicaciones y suscripciones. La sintaxis de los series de tema y los filtros en los clientes MQTT es en gran medida la misma que la de las series de temas en IBM WebSphere MQ.

Las series de tema se utilizan para enviar publicaciones a suscriptores. Cree una serie de tema utilizando el método, `MqttClient.getTopic(java.lang.String topicString)`.

Los filtros de tema se utilizan para suscribirse a temas y recibir publicaciones. Los filtros de tema pueden contener comodines. Los comodines le permiten suscribirse a varios temas. Puede crear un filtro de tema utilizando un método de suscripción; por ejemplo, `MqttClient.subscribe(java.lang.String topicFilter)`.

Series de tema

La sintaxis de una serie de tema de IBM WebSphere MQ se describe en Series de tema. La sintaxis de las series de tema de MQTT se describe en la clase `MqttClient` en la documentación de la API para Cliente MQTT para Java. Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT..

La sintaxis de cada tipo de serie de tema es casi la misma. Existen cuatro diferencias menores:

1. Las series de tema enviadas a IBM WebSphere MQ por clientes de MQTT deben seguir el convenio para los nombres de gestores de colas. Concretamente, las series de temas no pueden contener guiones.
2. Las longitudes máximas difieren. Las series de tema de IBM WebSphere MQ están limitadas a 10.240 caracteres. Un cliente MQTT puede crear series de tema de hasta 65535 bytes.
3. Una serie de tema creada por un cliente MQTT no puede contener ningún carácter nulo.
4. En WebSphere Message Broker, un nivel de tema nulo, `'...//...'` no era válido. Los niveles de tema nulos están permitidos en IBM WebSphere MQ.

A diferencia de la publicación/suscripción en IBM WebSphere MQ, en el protocolo `mqttv3` no existe el concepto de objeto de tema administrativo. No puede construir una serie de tema a partir de un objeto de tema y una serie de tema. Sin embargo, en WebSphere MQ la serie de tema se correlaciona con un tema administrativo. El control de accesos asociado al tema administrativo determina si una publicación se publica en el tema o se rechaza. Los atributos que se aplican a una publicación cuando se reenvía a los suscriptores están afectados por los atributos del tema administrativo.

Filtros de tema

La sintaxis de un filtro de tema de IBM WebSphere MQ se describe en Esquema de comodín basado en temas. La sintaxis de los filtros de tema que puede construir con un cliente de MQTT se describe en la clase `MqttClient` en la documentación de la API del Cliente MQTT para Java.Para obtener enlaces

a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.

La sintaxis de cada tipo de filtro de temas es casi la misma. La única diferencia es la forma en que los distintos intermediarios MQTT interpretan un filtro de temas. En WebSphere Message Broker V6, sólo se podía utilizar un comodín multinivel al final de un filtro de temas. En WebSphere MQ, se puede utilizar un comodín de varios niveles en cualquier nivel del árbol de temas; por ejemplo, USA/#/Dutchess County.

Referencia de programación del cliente MQTT

Estos son los enlaces a Mobile Messaging and M2M Paquete de clientes y a la documentación de la API del cliente asociado.

En el Mobile Messaging and M2M Paquete de clientes, se empaquetan las bibliotecas de cliente MQTT con la documentación de la API generada. Puede descargar el paquete de cliente de Descargas de comunidad de mensajería de IBM.

Puede ver copias en línea de la documentación de API más reciente siguiendo estos enlaces al proyecto de Eclipse Paho:

- Cliente MQTT para clases Java
- Biblioteca de clientes MQTT para C
- Biblioteca de clientes MQTT asíncrona para C

Nota:

1. Enlace las aplicaciones de MQTT Java al paquete `org.eclipse.paho.client.mqttv3` en lugar de `com.ibm.micro.client.mqttv3`. :NONE. El paquete `com.ibm.micro.client.mqttv3` se proporciona para dar soporte a las aplicaciones de MQTT Java existentes.
2. V7.5.0.1 Enlace las aplicaciones cliente de MQTT para C a la biblioteca `MQTTAsync` en lugar de a la biblioteca `MQTTClient` . `MQTTClient` se proporciona para dar soporte a las aplicaciones MQTT existentes para C.
3. Cliente MQTT de mensajería para JavaScript necesita un servidor de MQTT que sea compatible con WebSockets. Por ejemplo, IBM WebSphere MQ Version 7.5 y las versiones posteriores lo son.

Iniciación a los servidores MQTT

Los servidores de mensajería que dan soporte al protocolo de transporte MQTT disponibles en IBM y otros. El servidor MQTT más básico habilita aplicaciones y dispositivos móviles, con el soporte de bibliotecas de cliente MQTT, para intercambiar mensajes. IBM WebSphere MQ e IBM MessageSight son servidores MQTT de IBM. Además de actuar como servidores MQTT básicos, también intercambian mensajes entre las aplicaciones cliente MQTT y las aplicaciones de empresa. Todos los servidores MQTT de IBM admiten el protocolo MQTT version 3.1 y MQTT sobre el WebSocket protocol.

Servidores MQTT actuales de IBM

IBM WebSphere MQ

- IBM WebSphere MQ proporciona mensajería de grado empresarial. El componente de telemetría permite a IBM WebSphere MQ actuar también como servidor MQTT.
- Esto da soporte a sus aplicaciones móviles, de máquina a máquina (M2M - machine-to-machine) y basadas en dispositivos y les permite intercambiar mensajes con aplicaciones de mensajería empresarial tales como aplicaciones IBM WebSphere MQ y JMS.
- La instalación de IBM WebSphere MQ incluye una copia del SDK de MQTT de IBM. Este SDK proporciona aplicaciones cliente MQTT de ejemplo y bibliotecas de cliente MQTT que dan soporte a estas aplicaciones.

Nota: Para obtener la versión más actualizada de este SDK, descargue Mobile Messaging and M2M Paquete de clientes. Si desea ver más información, consulte "Iniciación a los clientes MQTT" en la página 11.

- El soporte para MQTT se introdujo por primera vez en IBM WebSphere MQ Version 7.0.1. Para obtener información completa para cada release de IBM WebSphere MQ, consulte la siguiente documentación del producto:
  - WebSphere MQ Telemetry versión 7.5
  - WebSphere MQ Telemetry versión 7.1

  Para obtener una breve introducción a IBM WebSphere MQ y los pasos para comenzar con el componente de IBM WebSphere MQ Telemetry, consulte "IBM WebSphere MQ como servidor MQTT" en la página 141.

IBM MessageSight

- IBM MessageSight es un servidor MQTT basado en dispositivos que puede conectar un gran número de clientes MQTT a la vez y ofrecer el rendimiento y la escalabilidad necesarios para ajustarse al siempre creciente volumen de dispositivos y sensores móviles. Da soporte al protocolo MQTT version 3.1, y a MQTT sobre WebSocket protocol.

- Las principales características y ventajas de IBM MessageSight como servidor MQTT son las siguientes:
  - Alto rendimiento, fiabilidad y mensajería escalable.
  - Diseñado específicamente para escenarios M2M (machine-to-machine) e Internet of Things, admite puntos finales conectados de forma concurrente en grandes comunidades.
  - Facilidad de instalación y uso. Puede estar listo y en funcionamiento en menos de 30 minutos.
  - Soporte para aplicaciones móviles nativas incluyendo Android y iOS.
  - Integración con IBM WebSphere MQ como intermediario de publicación/suscripción.
- Para obtener una introducción rápida a IBM MessageSight, consulte introducción a MessageSight en YouTube y anuncio de MessageSight. Para información técnica detallada, consulte Documentación del producto MessageSight.

IBM WebSphere MQ Telemetry daemon for devices

- También se conoce como IBM WebSphere MQ Telemetry advanced client for C. Es un servidor MQTT de poco espacio que generalmente se ejecuta en ubicaciones satélites o en dispositivos cercanos al extremo de la red; por ejemplo en decodificadores, unidades remotas de telemetría o terminales de punto de venta.
- Se utiliza frecuentemente para concentrar muchas conexiones de cliente MQTT, que a su vez se conectan a IBM WebSphere MQ por Internet en una única conexión MQTT. Por ejemplo, puede instalar un gran número de sensores en un edificio, conectarlos al IBM WebSphere MQ Telemetry daemon for devices y conectar el daemon a IBM WebSphere MQ.

- El IBM WebSphere MQ Telemetry daemon for devices se incluye con IBM WebSphere MQ. Es necesaria una licencia distinta para conectarlo a IBM WebSphere MQ. Consulte IBM United States Software Announcement 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) es una versión del IBM WebSphere MQ Telemetry daemon for devices. La diferencia principal estriba en el uso. Es un servidor de pruebas pequeño, disponible en IBM alphaWorks y pensado para utilizar al evaluar o realizar experimentos con soluciones basadas en MQTT. RSMB da soporte a MQTT en diversas plataformas Linux, en Windows XP, en Apple Mac OS X Leopard y en Unslung (Linksys NSLU12)

Servidores MQTT anteriores de IBM

WebSphere Message Broker ***(ahora conocido como IBM Integration Bus)***

- WebSphere Message Broker versión 6 proporcionaba su propio servidor MQTT. El soporte fue sustituido en WebSphere Message Broker versión 7 por el componente de telemetría de IBM WebSphere MQ.

Otros servidores MQTT

MQTT.org mantiene una lista de servidores e intermediarios MQTT en su página Software, incluyendo los servidores de código abierto.

Tareas relacionadas

"Iniciación a los clientes MQTT" en la página 11
Puede empezar a desarrollar una aplicación móvil o de máquina a máquina (M2M - machine-to-machine) compilando y ejecutando una aplicación cliente MQTT de ejemplo que utilice una biblioteca de cliente MQTT. Las aplicaciones de ejemplo y las bibliotecas de cliente asociadas están disponibles en el Mobile Messaging and M2M Paquete de clientes de IBM. Hay versiones de las aplicaciones y bibliotecas de cliente escritas en Java, en JavaScripty en C. Puede ejecutar estas aplicaciones en la mayoría de plataformas y dispositivos, incluidos los dispositivos y productos de Android desde Apple.

IBM WebSphere MQ como servidor MQTT

Introducción a cómo utilizar el servidor MQTT que se incluye en IBM WebSphere MQ.

Para empezar, siga los pasos que se describen en los siguientes artículos:

- "Instalación de IBM WebSphere MQ" en la página 142
- "Configuración del servicio de MQTT desde la línea de mandatos" en la página 143
- "Configuración del servicio de MQTT con IBM WebSphere MQ Explorer" en la página 146

Nota: Puede iniciarse rápidamente utilizando el ejemplo de la interfaz de línea de mandatos. Sin embargo, si la configuración es significativamente distinta de la del ejemplo necesitará más conocimientos para utilizar la interfaz de línea de mandatos con eficacia. Utilice la interfaz de IBM WebSphere MQ Explorer tanto para iniciarse como para realizar tareas de configuración estándar fácilmente.

Para obtener información conceptual clave sobre el componente IBM WebSphere MQ Telemetry, consulte los artículos siguientes en la documentación del producto IBM WebSphere MQ:

- Conexión de dispositivos de telemetría a un gestor de colas
- Servicio de telemetría (MQXR)
- Canales de telemetría

Información relacionada

Configuración de un gestor de colas para telemetría en Linux y AIX
Configuración de un gestor de colas para la telemetría en Windows
Configurar la cola distribuida para enviar mensajes a clientes MQTT
Administración de WebSphere MQ Telemetry

Instalación de IBM WebSphere MQ

Siga estas instrucciones para obtener e instalar IBM WebSphere MQ y configurar IBM WebSphere MQ Telemetry sobre Windows o Linux.

Antes de empezar

Para los sistemas operativos que están soportados por el servicio MQTT que se ejecuta en IBM WebSphere MQ, consulte Requisitos de sistema de telemetría de IBM WebSphere MQ.

Obtenga una copia del material de instalación de IBM WebSphere MQ y una licencia de una de las formas siguientes:

1. Pida a su administrador de IBM WebSphere MQ el material de instalación y confirme con él que puede aceptar el acuerdo de licencia.
2. Obtenga una copia de evaluación de 90 días de IBM WebSphere MQ. Consulte Evaluar: IBM WebSphere MQ.
3. Adquiera IBM WebSphere MQ. Consulte IBM WebSphere MQ página de producto.

Acerca de esta tarea

Instale IBM WebSphere MQ como `root` en Linux y como administrador en Windows. Durante la instalación, seleccione las opciones adicionales `Servicio de telemetría` y `Clientes de telemetría` para instalar el componente IBM WebSphere MQ Telemetry. Cree un ID de usuario para administrar IBM WebSphere MQ y compruebe que el ID de usuario invitado está definido. El ID de usuario invitado que se utiliza en la configuración del servicio MQTT de ejemplo para autorizar el acceso MQTT a IBM WebSphere MQ.

Tras instalar IBM WebSphere MQ, inicie el servicio MQTT siguiendo los pasos que se detallan en “Configuración del servicio de MQTT desde la línea de mandatos” en la página 143 o en “Configuración del servicio de MQTT con IBM WebSphere MQ Explorer” en la página 146.

Procedimiento

1. Inicie la sesión como `root` en Linux, o como administrador en Windows.
2. Instale IBM WebSphere MQ.

   Siga las instrucciones de Instalación del servidor WebSphere MQ en Linux o Instalación del servidor WebSphere MQ en Windows. Seleccione `Servicio de telemetría` y `Clientes de telemetría` para instalar el componente IBM WebSphere MQ Telemetry.

   En Linux, siga las instrucciones de la sección "Qué hacer a continuación" para realizar la instalación primaria. Incluso si esta instalación es la única instalación de IBM WebSphere MQ en la estación de trabajo, configúrela como primaria. Consulte Instalación sencilla de WebSphere MQ versión 7.1 o posterior configurada como la instalación primaria.

   Para seguir exactamente las instrucciones de configuración de ejemplo, debe configurar la instalación como primaria.

   Varias instalaciones: Si desea trabajar con una instalación no primaria, ejecute el mandato setmqenv. Se configura el entorno IBM WebSphere MQ en una ventana de mandatos en la estación de trabajo. Consulte Varias instalaciones.

   Suponiendo que haya aceptado la ubicación de instalación predeterminada que le ofrecía el programa de instalación, IBM WebSphere MQ se habrá instalado en los siguientes directorios:

   Linux de 64 bits

   ```
   /opt/mqm
   ```

Windows de 32 bits

```
C:\Program Files\IBM\WebSphere MQ
```

Windows de 64 bits

```
C:\Program Files (x86)\IBM\WebSphere MQ
```

El directorio de instalación se muestra como *MQ_INSTALLATION_PATH*

3. Opcional: Añada el usuario con el que vaya a administrar IBM WebSphere MQ en el grupo `mqm` en esta estación de trabajo.

   Este paso es opcional en Windows porque puede administrar IBM WebSphere MQ como administrador de Windows . Consulte Authorización para administrar WebSphere MQ en sistemas UNIX y Windows.

   Si la estación de trabajo de Windows es un miembro de un dominio, consulte Dominio de Windows 2000 con permisos de seguridad no predeterminados o Windows 2003 y Windows Server 2008 con permisos de seguridad predeterminados.

   En Linux, el programa de instalación crea un usuario `mqm`, como miembro del grupo `mqm`. Dé a este usuario una contraseña o cree otro usuario que tenga `mqm` como grupo primario.
4. Opcional: Inicie sesión con el usuario al que ha hecho miembro del grupo `mqm`.

   Este paso es opcional en Windows porque puede administrar IBM WebSphere MQ como administrador de Windows .
5. Compruebe que el ID de usuario invitado está definido en la estación de trabajo.

   El ID de usuario invitado es "`guest`" en Windows y "`nobody`" en Linux. El ID de usuario invitado no requiere permisos ni derechos de sistema operativo.

Resultados

Ha instalado IBM WebSphere MQ en su estación de trabajo como instalación primaria IBM WebSphere MQ y ha creado el grupo `mqm`. La instalación da permiso para administrar IBM WebSphere MQ a los miembros del grupo `mqm` . Los miembros del grupo de administradores en Windows también tienen autoridad para administrar IBM WebSphere MQ.

Qué hacer a continuación

1. Configure el servicio MQTT desde la línea de mandatos a IBM WebSphere MQ Explorer; consulte “Configuración del servicio de MQTT con IBM WebSphere MQ Explorer” en la página 146 o “Configuración del servicio de MQTT desde la línea de mandatos” en la página 143.
2. Pruebe sus clientes Android, iOS, WebSockets, Java y "C" MQTT.
3. Cuando termine la prueba, elimine el gestor de colas y el servicio MQTT ejecutando el mandato *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` en Windows y *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` en Linux.

Información relacionada

Instalación de WebSphere MQ Telemetry

Instalación del servidor WebSphere MQ en Linux

Instalación del servidor de WebSphere MQ en Windows

Configuración del servicio de MQTT desde la línea de mandatos

Siga estas instrucciones para configurar IBM WebSphere MQ utilizando la línea de mandatos para ejecutar las aplicaciones ejemplo de IBM WebSphere MQ Telemetry. Los pasos le muestran cómo ejecutar un script para crear un servicio MQTT en un nuevo gestor de colas denominado `MQXR_SAMPLE_QM`.

Antes de empezar

Debe tener acceso administrativo a un gestor de colas de IBM WebSphere MQ para configurar el servicio de MQTT. Puede acceder a un gestor de colas de diversas formas:

1. Obtener una copia de IBM WebSphere MQ y crear un gestor de colas en su propia estación de trabajo Linux o Windows. Siga las instrucciones de "Instalación de IBM WebSphere MQ" en la página 142 para obtener e instalar IBM WebSphere MQ. Tenga en cuenta que también debe seleccionar `Servicio de telemetría` y `Clientes de telemetría` durante la instalación. También puede modificar una instalación existente para añadir estas opciones.
2. Ponerse en contacto con un administrador de IBM WebSphere MQ y solicitar acceso administrativo a un gestor de colas en un servidor que tenga la opción de IBM WebSphere MQ Telemetry instalada.
   V 7.5.0.1 Además del nombre de gestor de colas, necesitará al menos dos puertos TCP/IP para MQTT y para MQTT sobre WebSockets. Si tiene previsto conectarse a clientes seguros, necesitará al menos dos puertos más.

Para realizar los pasos de la tarea exactamente tal como se describen, debe poder crear un gestor de colas denominado `MQXR_SAMPLE_QM`, y el puerto TCP/IP 1883 debe estar sin utilizar.

Acerca de esta tarea

En esta tarea se ejecuta un script que crea un gestor de colas y luego configura el servicio de MQTT para estar a la escucha de conexiones de clientes MQTT V3.1 en el puerto 1883.La configuración permite a todo el mundo publicar y suscribirse a cualquier tema. La configuración de la seguridad y el control de accesos es mínima y está pensada únicamente para un gestor de colas que se encuentre en una red segura con acceso restringido. Para ejecutar IBM WebSphere MQ y MQTT en un entorno no seguro, debe configurar la seguridad. Para configurar la seguridad para IBM WebSphere MQ y MQTT, consulte los enlaces relacionados al final de esta tarea.

Procedimiento

1. Inicie sesión con un ID de usuario que tenga autorización administrativa para IBM WebSphere MQ.

   Para definir un ID de usuario con autorización administrativa para IBM WebSphere MQ, consulte el paso 3 en "Instalación de IBM WebSphere MQ" en la página 142.
2. Abra una ventana de mandatos y ejecute el script de mandatos de ejemplo para crear e iniciar el gestor de colas de ejemplo denominado servicio de `MQXR_SAMPLE_QM` y la MQTT.

   La vía de acceso al script de ejemplo es `%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat` en Windows y *`MQ_INSTALLATION_PATH`*`/mqxr/samples/SampleMQM.sh` en Linux.

   Escriba el mandato siguiente para crear y configurar el gestor de colas:

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Resultados

El ejemplo crea un canal MQTT denominado `PlainText` con estas propiedades en Windows:

```
com.ibm.mq.MQXR.channel/PlainText: \
com.ibm.mq.MQXR.Protocol=MQTT;\
com.ibm.mq.MQXR.Port=1883;\
com.ibm.mq.MQXR.Backlog=4096;\
com.ibm.mq.MQXR.UserName=Guest;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Las propiedades del canal en Linux son las mismas que en Windows, excepto `com.ibm.mq.MQXR.UserName=nobody`.

Los clientes MQTT V3.1 que se conectan al puerto 1883 acceden a IBM WebSphere MQ con el ID de usuario definido en la variable *com.ibm.mq.MQXR.UserName*. El script de ejemplo autoriza al ID de usuario con los siguientes mandatos de IBM WebSphere MQ:

```
setmqaut -m MQXR_SAMPLE_QM -t topic -n SYSTEM.BASE.TOPIC -p com.ibm.mq.MQXR.UserName -all +pub
+sub
setmqaut -m MQXR_SAMPLE_QM -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p com.ibm.mq.MQXR.UserName -all
+put
```

El primer mandato otorga al usuario autorización para publicar y suscribirse a temas que heredan sus permisos del tema base. El segundo mandato otorga al usuario autorización para poner mensajes en la cola de transmisión `SYSTEM.MQTT.TRANSMIT.QUEUE`. El servicio MQTT envía mensajes a la cola `SYSTEM.MQTT.TRANSMIT.QUEUE` como publicaciones a los suscriptores de MQTT.

El script inicia el servicio de MQTT en el gestor de colas para estar a la escucha en el puerto 1883.

Qué hacer a continuación

Siga estos pasos para probar la conexión ejecutando la aplicación Java de ejemplo de MQTT V3.1.

El origen de la aplicación Java de ejemplo se encuentra en el archivo `MQTTV3Sample.java` .

Se requieren dos ventanas de mandatos para ejecutar el ejemplo. En una ventana ejecute el ejemplo como suscriptor y en la otra como publicador.

- Windows Para iniciar el suscriptor, ejecute el mandato:

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  Para publicar, ejecute el mandato:

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux Para iniciar el suscriptor, ejecute el mandato:

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  Para publicar, ejecute el mandato:

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

El publicador y el suscriptor escriben la salida en sus ventanas de mandatos:

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 27. Salida del publicador

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
Topic:          MQTTV3Sample/Java/v3
Message:        Message from MQTTv3 Java client
QoS:            2
```

Figura 28. Salida del suscriptor

El servidor está ahora preparado para probar la aplicación MQTT V3.1 .

Tareas relacionadas

Configuración del servicio de MQTT con WebSphere MQ Explorer
Siga estas instrucciones para configurar IBM WebSphere MQ utilizando IBM WebSphere MQ Explorer para ejecutar los clientes de IBM WebSphere MQ Telemetry de ejemplo. Los pasos muestran cómo crear un servicio de MQTT ejecutando el asistente de configuración de `Define sample` .

Información relacionada

WebSphere MQ Telemetry
Desarrollo de aplicaciones para WebSphere MQ Telemetry
Administración de WebSphere MQ Telemetry
Seguridad en WebSphere MQ Telemetry

Configuración del servicio de MQTT con IBM WebSphere MQ Explorer

Siga estas instrucciones para configurar IBM WebSphere MQ utilizando IBM WebSphere MQ Explorer para ejecutar los clientes de IBM WebSphere MQ Telemetry de ejemplo. Los pasos muestran cómo crear un servicio de MQTT ejecutando el asistente de configuración de `Define sample` .

Antes de empezar

Debe tener acceso administrativo a un gestor de colas de IBM WebSphere MQ para configurar el servicio de MQTT. Puede acceder a un gestor de colas de diversas formas:

1. Obtener una copia de IBM WebSphere MQ y crear un gestor de colas en su propia estación de trabajo Linux o Windows. Siga las instrucciones de "Instalación de IBM WebSphere MQ" en la página 142 para obtener e instalar IBM WebSphere MQ. Tenga en cuenta que también debe seleccionar `Servicio de telemetría` y `Clientes de telemetría` durante la instalación. También puede modificar una instalación existente para añadir estas opciones.
2. Ponerse en contacto con un administrador de IBM WebSphere MQ y solicitar acceso administrativo a un gestor de colas en un servidor que tenga la opción de IBM WebSphere MQ Telemetry instalada.
   V 7.5.0.1 Además del nombre de gestor de colas, necesitará al menos dos puertos TCP/IP para MQTT y para MQTT sobre WebSockets. Si tiene previsto conectarse a clientes seguros, necesitará al menos dos puertos más.

Para realizar los pasos de la tarea exactamente tal como se describen, debe poder crear un gestor de colas denominado `MQXR_SAMPLE_QM`, y el puerto TCP/IP 1883 debe estar sin utilizar.

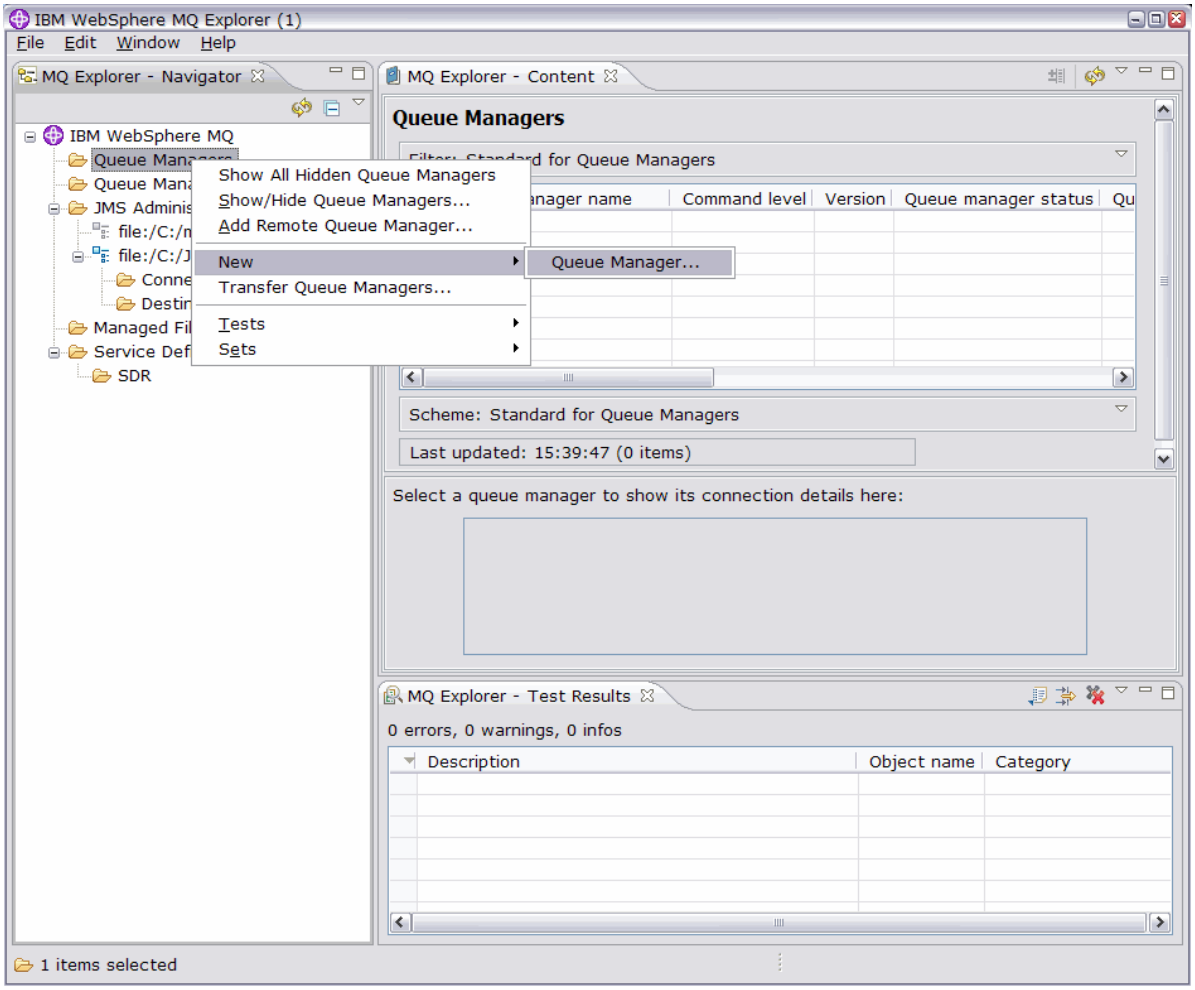
Acerca de esta tarea

En esta tarea, ejecute el asistente de configuración de IBM WebSphere MQ Explorer `Define sample` para crear un servicio de MQTT para escuchar las conexiones de cliente de MQTT V3.1 en el puerto 1883. La configuración permite a todo el mundo publicar y suscribirse a cualquier tema. La configuración de la seguridad y el control de accesos es mínima y está pensada únicamente para un gestor de colas que se encuentre en una red segura con acceso restringido. Para ejecutar IBM WebSphere MQ y MQTT en un entorno no seguro, debe configurar la seguridad. Para configurar la seguridad para IBM WebSphere MQ y MQTT, consulte los enlaces relacionados al final de esta tarea.

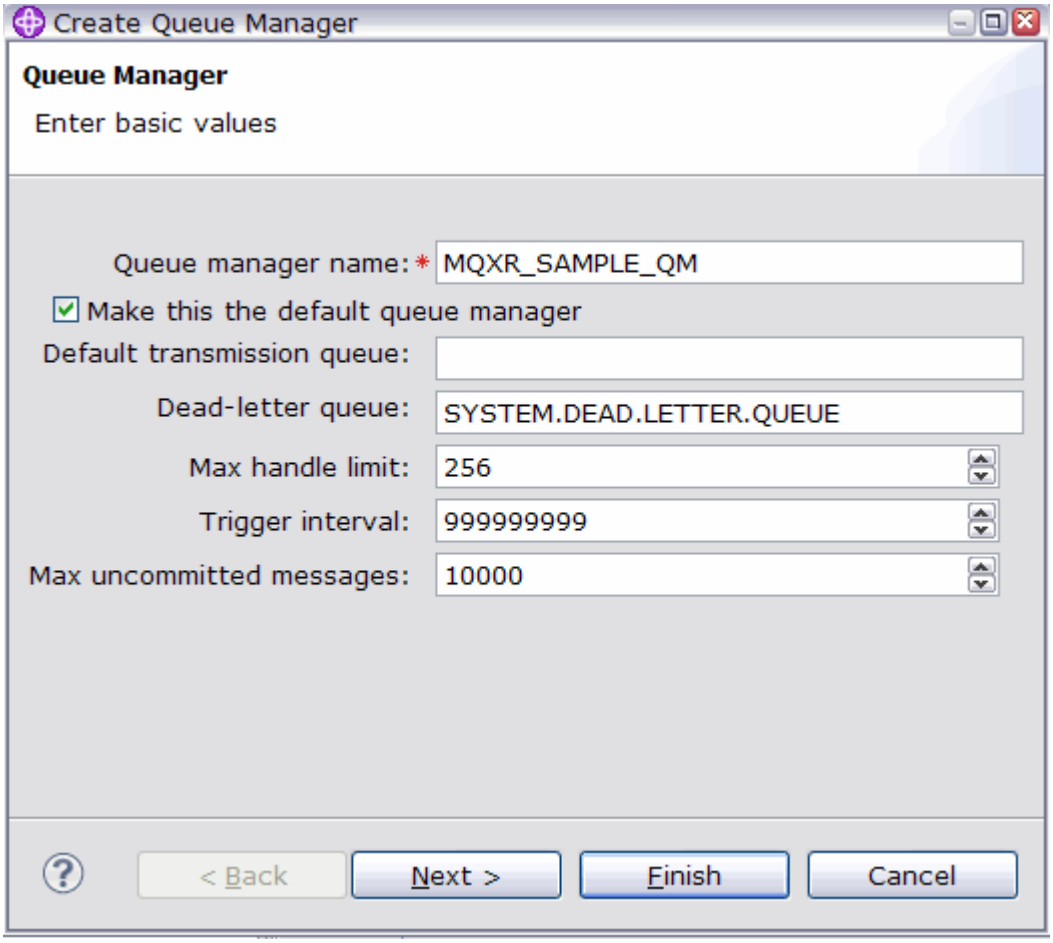
Procedimiento

1. Inicie sesión con un ID de usuario que tenga autorización administrativa para IBM WebSphere MQ.

   Para definir un ID de usuario con autorización administrativa para IBM WebSphere MQ, consulte el paso 3 en "Instalación de IBM WebSphere MQ" en la página 142.
2. Abra una ventana de mandatos y ejecute el mandato IBM WebSphere MQ Explorer **strmqcfg** para iniciar IBM WebSphere MQ Explorer.
3. Crear un gestor de colas
   a) Inicie el asistente **Nuevo gestor de colas**

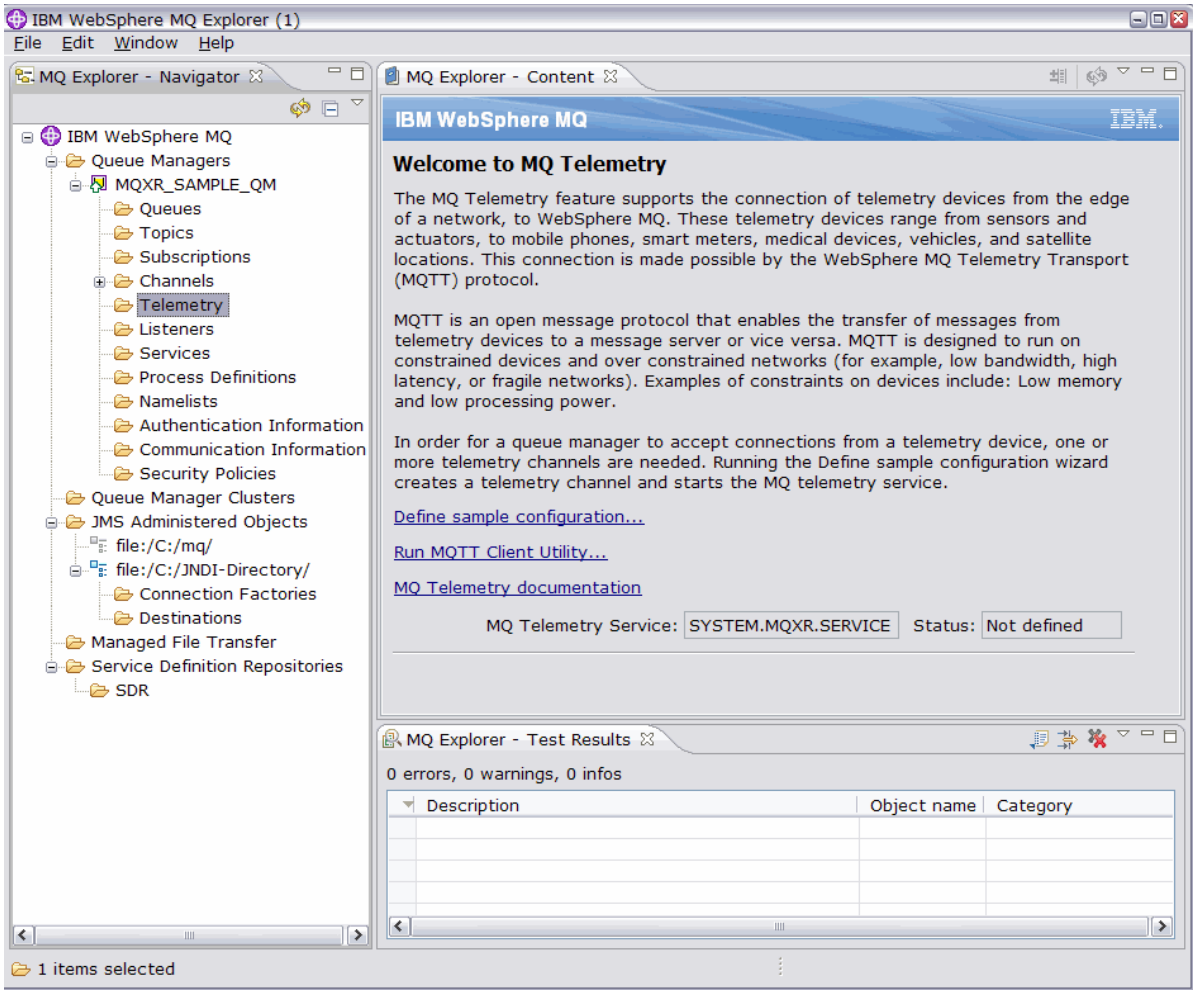


b) Escriba un nombre para el **gestor de colas** y el nombre de la **cola de mensajes no entregados**. Para su comodidad, márquelo como el gestor de colas predeterminado. Pulse **Finalizar**.



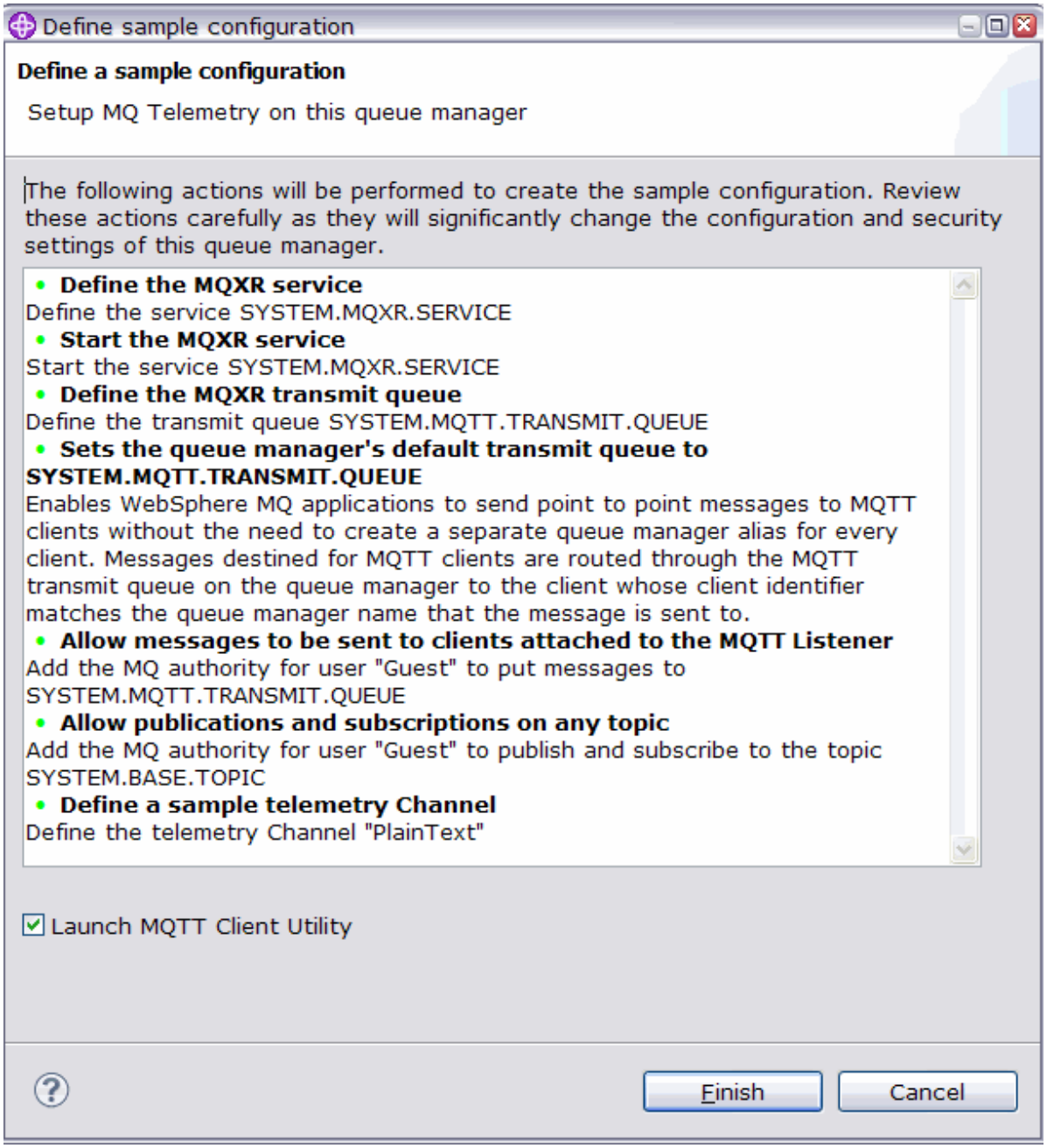
IBM WebSphere MQ Explorer crea el gestor de colas y lo inicia.

4. Ejecute el asistente de telemetría **Definir configuración de ejemplo**.
   a) Abra la carpeta Telemetría del gestor de colas.



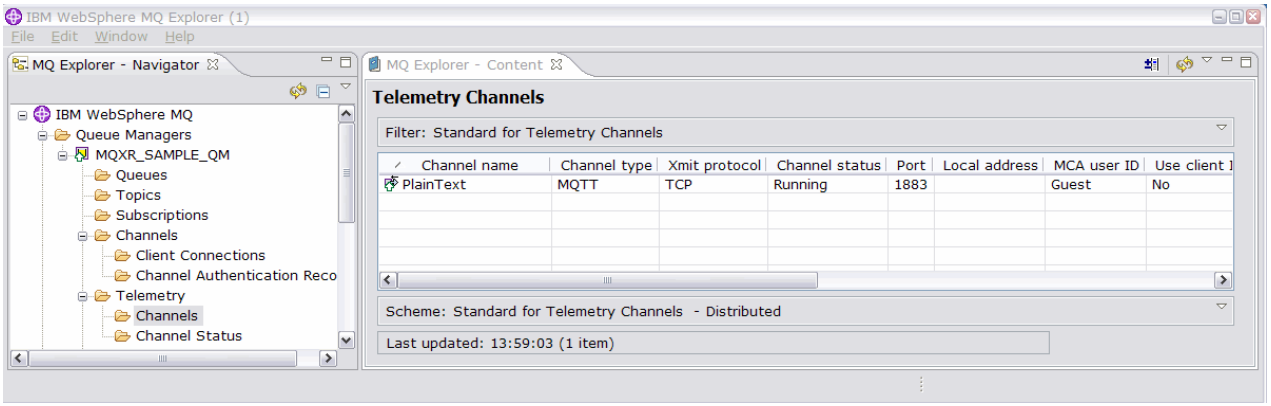
b) Pulse en **Definir configuración de ejemplo** para iniciar el asistente.

c) Pulse **Finalizar** para crear el servicio de telemetría y ejecutar el programa de utilidad de cliente MQTT



Resultados

Abra la carpeta Canales de telemetría para listar los canales de ejemplo.



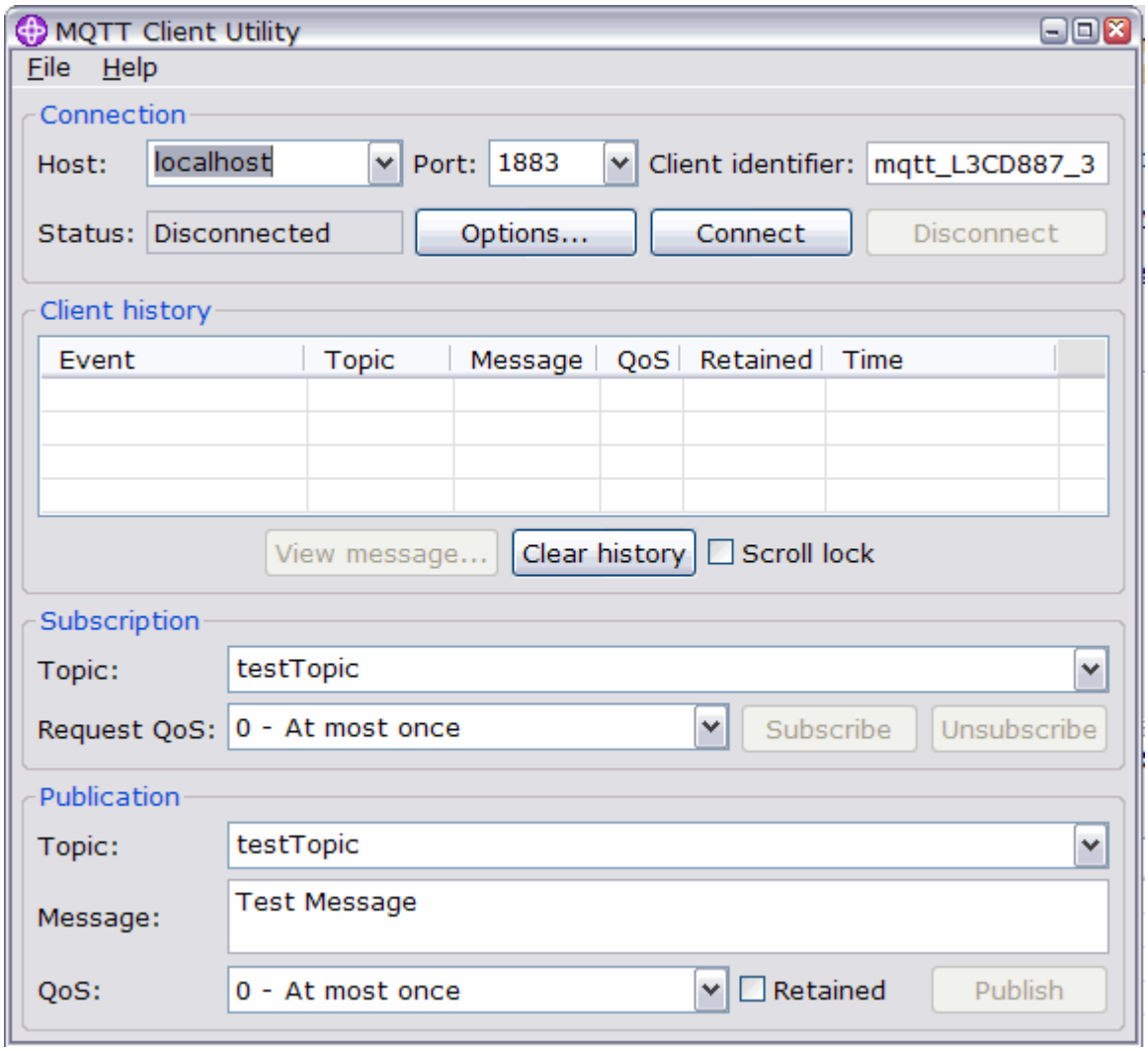
Puede modificar las propiedades de este canal y añadir y eliminar canales en esta ventana.

Qué hacer a continuación

Puede probar la conexión ejecutando el programa de utilidad de cliente MQTT.

1. Para iniciar el programa de utilidad de cliente, abra la carpeta **Telemetría** y pulse **Ejecutar el programa de utilidad de cliente MQTT** dos veces.

   Se abrirán dos ventanas **Programa de utilidad de cliente MQTT**, idénticas pero para distintos identificadores de cliente.



2. Pulse **Conectar** en ambas ventanas.
3. Pulse **Suscribir** en ambas ventanas.
4. Pulse **Publicar** en cualquiera de las dos ventanas. Los resultados se muestran en Figura 29 en la página 152

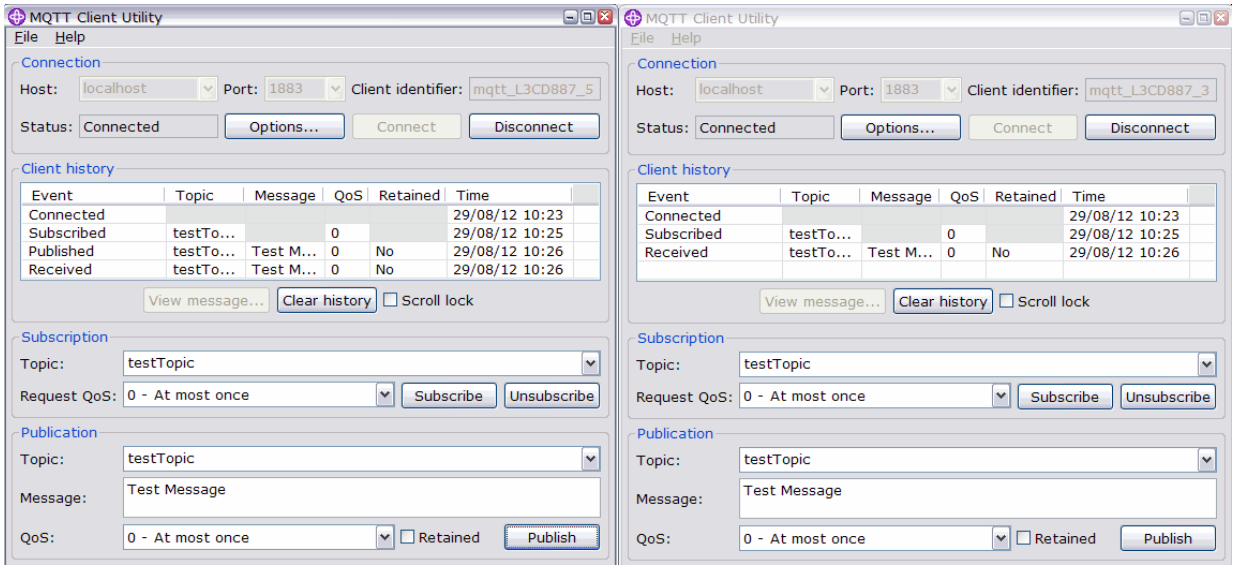


Figura 29. Resultados

5. Pulse **Desconectar** en ambas ventanas.

El servidor está ahora preparado para probar la aplicación MQTT V3.1 .

Tareas relacionadas

Configuración del servicio de MQTT desde la línea de mandatos

Siga estas instrucciones para configurar IBM WebSphere MQ utilizando la línea de mandatos para ejecutar las aplicaciones ejemplo de IBM WebSphere MQ Telemetry. Los pasos le muestran cómo ejecutar un script para crear un servicio MQTT en un nuevo gestor de colas denominado `MQXR_SAMPLE_QM`.

Administración de WebSphere MQ Telemetry

Información relacionada

WebSphere MQ Telemetry

Administración de WebSphere MQ Telemetry con WebSphere MQ Explorer

Desarrollo de aplicaciones para WebSphere MQ Telemetry

Seguridad

Seguridad en WebSphere MQ Telemetry

Daemon de IBM WebSphere MQ Telemetry para conceptos de dispositivos

El daemon de IBM WebSphere MQ Telemetry para dispositivos es una aplicación cliente MQTT V3 avanzada. Utilícelo para almacenar y reenviar mensajes desde otros clientes MQTT. Se conecta a IBM WebSphere MQ como un cliente MQTT, pero también puede conectarle otros clientes MQTT.

El daemon es un intermediario de publicación/suscripción. Los clientes MQTT V3 se conectan al él para publicar y suscribirse a temas, utilizando las series de tema para publicar y los filtros de temas para suscribirse. La serie de tema es jerárquica, con niveles de tema divididos por /. Los filtros de temas son series de tema que pueden incluir comodines + de un solo nivel y un comodín # multinivel como la última parte de la serie de tema.

Nota: Los comodines del daemon siguen las reglas más restrictivas de WebSphere Message Broker, v6. IBM WebSphere MQ es diferente. Da soporte a varios comodines multinivel; puede representar cualquier número de niveles de la jerarquía, en cualquier lugar de la serie de tema.

Varios clientes MQTT v3 se conectan al daemon utilizando un puerto de escucha. El puerto de escucha predeterminado puede modificarse. Puede definir varios puertos de escucha y asignar espacios de nombres distintos a los mismos, consulte “Puertos de escucha del daemon de WebSphere MQ Telemetry para dispositivos” en la página 160. El daemon, en sí mismo, es un cliente MQTT v3. Configure una conexión de puente de daemon para conectar el daemon al puerto de escucha de otro daemon,o a un servicio de WebSphere MQ Telemetry (MQXR).

Puede configurar varios puentes para el daemon de WebSphere MQ Telemetry para dispositivos. Utilice los puentes para interconectar una red de daemons que puedan intercambiar publicaciones.

Cada puente puede publicar y suscribirse a temas en su daemon local. También puede publicar y suscribirse a temas en otro daemon, en un intermediario de publicación/suscripción de WebSphere MQ o en cualquier otro intermediario MQTT v3 al que se haya conectado. Al utilizar un filtro de temas, puede seleccionar las publicaciones para propagar de un intermediario a otro. Puede propagar las publicaciones en cualquier dirección. Puede propagar las publicaciones del daemon local a cada uno de sus intermediarios remotos conectados, o de cualquiera de los intermediarios conectados al daemon local; consulte “Daemon de IBM WebSphere MQ Telemetry para puentes de dispositivos” en la página 153.

Daemon de IBM WebSphere MQ Telemetry para puentes de dispositivos

Un puente de IBM WebSphere MQ Telemetry para el puente de dispositivos conecta dos intermediarios de publicación/suscripción utilizando el protocolo MQTT v3. El puente propaga las publicaciones de un intermediario a otro, en cualquier dirección. En un extremo hay una conexión de puente del daemon de WebSphere MQ Telemetry para dispositivos, y en el otro puede haber un gestor de colas, u otro daemon. Hay un gestor de colas conectado a la conexión de puente mediante un canal de telemetría. Hay un daemon conectado a la conexión de puente mediante un escucha de daemon.

El daemon IBM WebSphere MQ Telemetry para dispositivos admite una o más conexiones simultáneas a otros intermediarios. Las conexiones procedentes del daemon se llaman puentes, y se definen mediante las entradas de conexión del archivo de configuración del daemon. Las conexiones a IBM WebSphere MQ se establecen mediante los canales de telemetría de IBM WebSphere MQ, tal como se muestra en la figura siguiente:

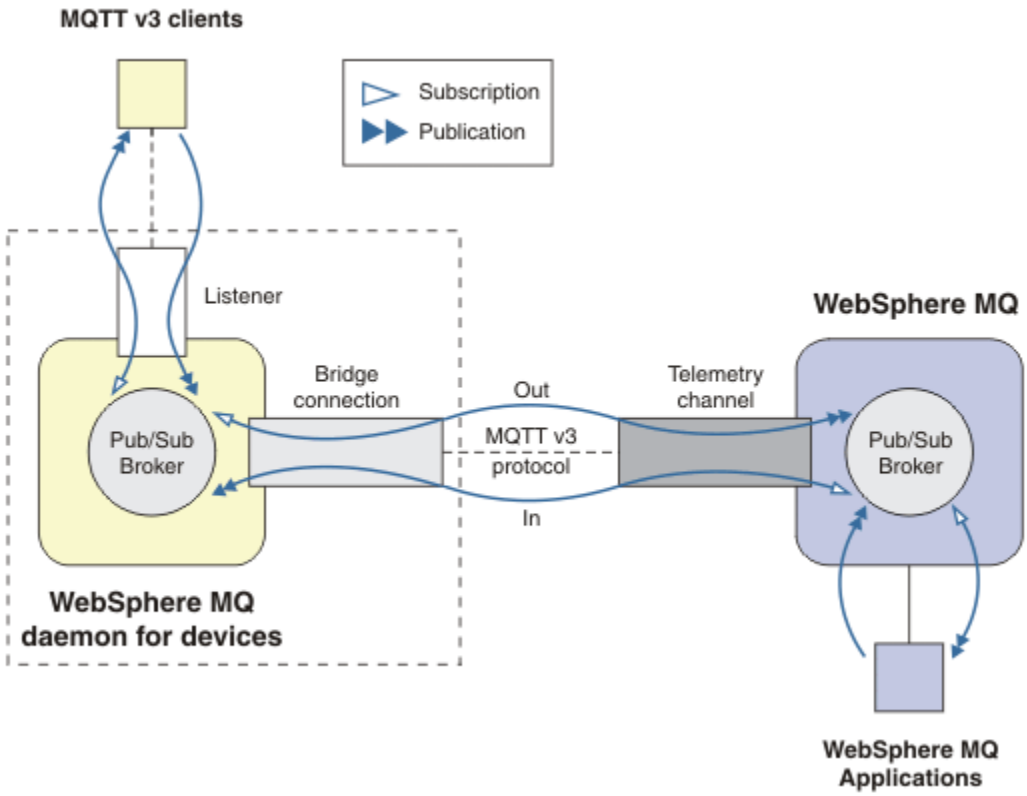


Figura 30. Conexión de IBM WebSphere MQ Telemetry daemon for devices a IBM WebSphere MQ

Un puente conecta el daemon a otro intermediario como un cliente MQTT v3. Los parámetros del puente duplican los atributos de un cliente MQTT v3.

Un puente es más que una conexión. Actúa como un agente de publicación y suscripción, situado entre dos intermediarios de publicación/suscripción. El intermediario local es el daemon IBM WebSphere MQ Telemetry para dispositivos, y el intermediario es cualquier intermediario de publicación/suscripción remoto que dé soporte al protocolo MQTT v3. Normalmente, el intermediario remoto es otro daemon, o IBM WebSphere MQ.

La función del puente es a propagar publicaciones entre ambos intermediarios. El puente es bidireccional. Propaga las publicaciones en cualquier dirección. En la Figura 30 en la página 153 se ilustra el modo en que el puente conecta el daemon de IBM WebSphere MQ Telemetry para dispositivos a IBM WebSphere MQ. En “Valores de tema de ejemplo para el puente” en la página 154 se utilizan ejemplos que ilustran cómo utilizar el parámetro topic para configurar el puente.

Las flechas In (entrada) y Out (salida) que aparecen en la Figura 30 en la página 153 indican la bidireccionalidad del puente. En un extremo de la flecha se crea una suscripción. Las publicaciones que coincidan con la suscripción se publican en el intermediario situado en el extremo opuesto de la flecha. La etiqueta de la flecha depende del flujo de las publicaciones. El flujo de las publicaciones entra (In) en el daemon, y sale (Out) del daemon. La importancia de las etiquetas es que se utilizan en la sintaxis del mandato. Recuerde que In y Out hacen referencia a dónde fluyen las publicaciones, y no a dónde se envía la suscripción.

Otros clientes, otras aplicaciones u otros intermediarios pueden estar conectados a IBM WebSphere MQ o al daemon de WebSphere MQ Telemetry para dispositivos. Publican y se suscriben a temas en el intermediario al que están conectados. Si el intermediario es IBM WebSphere MQ, los temas pueden estar en clúster o distribuidos, y no definidos de forma explícita en el gestor de colas local.

Usos de los puentes

Puede conectar los daemons entre sí, utilizando conexiones de puente y escuchas. Puede conectar los demonios y los gestores de colas entre sí, utilizando conexiones de puente y canales de telemetría. Cuando se conectan varios intermediarios entre sí, es posible que se creen bucles. Atención: las publicaciones podrían circular indefinidamente dentro de un bucle de intermediarios, sin detectarse.

Algunas de las razones para utilizar daemons conectados a IBM WebSphere MQ son las siguientes:

Reducir el número de conexiones de cliente MQTT a WebSphere MQ.
Utilizar una jerarquía de daemons, puede conectar muchos clientes a WebSphere MQ; más clientes que los que un solo gestor de colas puede conectar a la vez.

Almacenar y reenviar los mensajes entre clientes MQTT y WebSphere MQ.
Puede utilizar el almacenamiento y el reenvío para evitar tener que mantener conexiones continuas entre los clientes y el IBM WebSphere MQ, si los clientes no disponen de su propio almacenamiento. Puede utilizar varios tipos de conexiones entre el cliente MQTT y WebSphere MQ; consulte Conceptos y escenarios de telemetría para supervisión y control.

Filtrar las publicaciones intercambiadas entre los clientes MQTT y WebSphere MQ.
Normalmente, las publicaciones se dividen en los mensajes que se procesan localmente y los mensajes en los que intervienen otras aplicaciones. Las publicaciones locales pueden incluir flujos de control entre sensores y actuadores, y las publicaciones remotas incluyen peticiones de mandatos de lecturas, estado y de configuración.

Cambiar los espacio de temas de las publicaciones.
Evite que las series de tema procedentes de clientes conectados a puertos de escucha distintos colisionen entre sí. En el ejemplo, se utiliza el daemon para etiquetar las lecturas de medidor procedentes de edificios diferentes; consulte Separación de espacios de temas de grupos de clientes diferentes.

Valores de tema de ejemplo para el puente

Publicar todo en el intermediario remoto: utilizar los valores predeterminados

La dirección predeterminada se conoce como out, y el puente publica los temas en el intermediario remoto. El parámetro topic controla qué temas se propagan utilizando los filtros de tema.

El puente utiliza el parámetro topic en Figura 31 en la página 155 para suscribirse a todo lo publicado en el daemon local por los clientes MQTT, o por otros intermediarios. El puente publica los temas en el intermediario remoto que se ha conectado mediante el puente.

```
connection Daemon1
topic #
```

Figura 31. Publicar todo en el intermediario remoto

Publicar todo en el intermediario remoto: explicito

El valor de topic en el fragmento de código siguiente proporciona el mismo resultado que utilizar los valores predeterminados. La única diferencia es que el parámetro **direction** es explícito. Utilice dirección `out` para suscribirse al intermediario local, el daemon, y publicar en el intermediario remoto. Las publicaciones creadas en el daemon local al cual el puente se haya suscrito, se publican en el intermediario remoto.

```
connection Daemon1
topic # out
```

Figura 32. Publicar todo en el intermediario remoto: explicito

Publicar todo en el intermediario local

En lugar de utilizar la dirección `out`, puede establecer la dirección contraria, `in`. El fragmento de código siguiente configura el puente para suscribirse a todo lo publicado en el intermediario remoto conectado al mismo. El puente publica los temas en el intermediario local, el daemon.

```
connection Daemon1
topic # in
```

Figura 33. Publicar todo en el intermediario local

Publicar todo el tema export del intermediario local en el tema import del intermediario remoto

Puede utilizar dos parámetros topic adicionales, **local_prefix** y **remote_prefix**, para modificar el filtro de tema, *#* en los ejemplos anteriores. Un parámetro se utiliza para modificar el filtro utilizado en el tema de la suscripción y el otro parámetro se utiliza para modificar el tema en el que se publica la publicación. El efecto que se consigue es sustituir el principio de la serie de tema que se utiliza en un intermediario por otra serie de tema en el otro intermediario.

En función de la dirección del mandato topic, el significado de **local_prefix** y de **remote_prefix** se invierte. Si la dirección es `out`, el valor predeterminado, se utiliza **local_prefix** como parte de la suscripción del tema, y **remote_prefix** sustituye a la parte **local_prefix** de la serie de tema, en la publicación remota. Si la dirección es `in`, **remote_prefix** se convierte en parte de la suscripción remota, y **local_prefix** sustituye a la parte **remote_prefix** de la serie de tema.

La primera parte de una serie de tema se considera, a menudo, como la parte que define un espacio de temas. Utilice los parámetros adicionales para cambiar el espacio de temas en el que se publica un tema. Puede hacer esto para evitar que el tema se propague y colisione con otro tema en el intermediario de destino, o para eliminar una serie de tema de punto de montaje.

Por ejemplo, en el fragmento de código siguiente, todas las publicaciones a la serie de tema `export/#` en el daemon, se vuelven a publicar como `import/#` en el intermediario remoto.

```
topic # out export/ import/
```

Figura 34. Publicar todo el tema export del intermediario local en el tema import del intermediario remoto

Publicar en el tema import del intermediario local todo lo procedente del tema export del intermediario remoto

El fragmento de código siguiente muestra la configuración invertida; el puente se suscribe a todo lo publicado con la serie de tema `export/#` en el intermediario remoto y lo publica en `import/#` en el intermediario local.

```
connection Daemon1
topic # in import/ export/
```

Figura 35. Publicar en el tema import del intermediario local todo lo procedente del tema export del intermediario remoto

Publicar todo desde el punto de montaje 1884/ al intermediario remoto, con las series de tema originales

En el fragmento de código siguiente, el puente se suscribe a todo lo publicado por los clientes conectados al punto de montaje 1884/ en el daemon local. El puente publica en el intermediario remoto todo lo publicado en el punto de montaje. La serie de tema del punto de montaje 1884/ se elimina de los temas publicados en el intermediario remoto. El *local_prefix* es el mismo que la serie del punto de montaje 1884/ y el *remote_prefix* es una serie en blanco.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Figura 36. Publicar todo desde el punto de montaje 1884/ al intermediario remoto, con las series de tema originales.

Separación de espacios de temas de clientes diferentes conectados a daemons diferentes

Supongamos se escribe una aplicación para que medidores de energía eléctrica publiquen sus lecturas de un edificio. Las lecturas se publican utilizando clientes MQTT a un daemon que se aloja en el mismo edificio. El tema seleccionado para que las publicaciones es `power`. La misma aplicación se despliega en varios edificios de un complejo. Para la supervisión y el almacenamiento de datos in situ, se agregan las lecturas de todos los edificios, utilizando las conexiones de puente. Las conexiones enlazan los daemons del edificio a WebSphere MQ, que se encuentra en una ubicación central.

Se utiliza una aplicación cliente idéntica en todos los edificios. Esta aplicación publica en el tema `power`. Sin embargo, los datos deben estar diferenciados por edificio. Esto se realiza mediante el daemon para cada edificio, que añade el número de edificio como prefijo al nombre de tema. El puente del primer edificio del complejo utiliza el prefijo `meters/building01/`; el prefijo del segundo edificio es `meters/building02/`. Las lecturas de los demás edificios siguen el mismo patrón. Por lo tanto, WebSphere MQ recibe las lecturas con temas como `meters/building01/power`.

El archivo de configuración de cada daemon tiene una sentencia de tema que sigue el patrón del fragmento de código siguiente:

```
connection Daemon1
topic power out "" meters/building01/
```

Figura 37. Separar espacios de temas de clientes conectados a daemons diferentes

En el fragmento de código anterior, la serie vacía es un espacio reservado para el parámetro local_prefix no utilizado.

Nota: Este ejemplo es un poco artificial y sólo pretende ser ilustrativo. En la práctica, el espacio de temas en el que la aplicación publica probablemente será configurable.

Separar espacios de temas de clientes conectados al mismo daemon.

Supongamos que se utiliza único daemon para conectar todos los medidores de corriente eléctrica. Suponiendo que la aplicación puede configurarse para conectarse a diferentes puertos, se pueden distinguir los edificios conectando los medidores de edificios diferentes a puertos de escucha diferentes, tal como se muestra en el fragmento de código siguiente. De nuevo, el ejemplo no es real; en él se ilustra cómo se pueden utilizar los puntos de montaje.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Figura 38. Separar espacios de temas de clientes conectados al mismo daemon.

Volver a correlacionar temas diferentes para las publicaciones que fluyen en ambas direcciones.

En la configuración del siguiente fragmento de código, el puente se suscribe al tema único b en el intermediario remoto y reenvía publicaciones sobre b al daemon local, cambiando el tema a a. El puente también se suscribe al tema único x en el intermediario local y reenvía publicaciones sobre x al intermediario remoto, cambiando el tema a y.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Figura 39. Volver a correlacionar temas diferentes para las publicaciones que fluyen en ambas direcciones.

Un punto importante de este ejemplo es que ambos intermediarios están suscritos a diferentes temas y en ambos se publican los diferentes temas. Los espacios de temas en ambos intermediarios están separados.

Volver a correlacionar los mismos temas para las publicaciones que fluyen en ambas direcciones (repetición en bucle).

A diferencia del ejemplo anterior, la configuración de Figura 40 en la página 158 generalmente da como resultado un bucle. En la sentencia topic `topic "" in a b`, el puente se suscribe a b de forma remota, y publica en a de forma local. En la otra sentencia topic, el puente se suscribe a a de forma local, y publica de forma remota en b. La misma configuración puede escribirse tal como se muestra en Figura 41 en la página 158.

El resultado general es que si un cliente publica en b de forma remota, la publicación se transfiere al daemon local como una publicación sobre el tema a. Sin embargo, al ser publicado por el puente en el daemon local en el tema a, la publicación coincide con la suscripción realizada por el puente en el

tema local a. La suscripción es `topic "" out a b`. Como resultado, la publicación se transfiere de nuevo al intermediario remoto como una publicación sobre el tema b. Ahora el puente está suscrito al tema remoto by el ciclo se inicia de nuevo.

Algunos intermediarios implementan la detección de bucles para evitar que se generen bucles. Sin embargo, el mecanismo de detección de bucles debe funcionar cuando diferentes tipos de intermediarios están conectados entre sí mediante un puente. La detección de bucle no funciona si WebSphere MQ tiene un puente que lo conecta al daemon de WebSphere MQ Telemetry para dispositivos. Sí funciona cuando dos daemons IBM WebSphere MQ Telemetry para dispositivos están conectados entre sí mediante un puente. De forma predeterminada, la detección de bucle está activada; consulte try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Figura 40. !Volver a correlacionar los mismos temas para las publicaciones que fluyen en ambas direcciones

```
connection Daemon1
topic "" both a b
```

Figura 41. !Vuelva a correlacionar los mismos temas para las publicaciones que fluyen en ambas direcciones, utilizando `both`.

La configuración que aparece en Figura 39 en la página 157 es la misma que la de Figura 40 en la página 158.

Disponibilidad de las conexiones de puente del IBM WebSphere MQ Telemetry daemon for devices

Configure varias direcciones de conexión de puente del IBM WebSphere MQ Telemetry daemon for devices para poder conectarse al primer intermediario remoto que haya disponible. Si el intermediario es un gestor de colas de varias instancias, proporcione ambas direcciones TCP/IP. Configure una conexión primaria para conectar, o volver a conectar, al servidor primario, cuando éste esté disponible.

El parámetro del puente de la conexión, addresses, es una lista de direcciones de socket de TCP/IP. El puente intentará conectarse a cada dirección por turnos, hasta que se realice una conexión satisfactoria. Los parámetro de conexión round_robin y start_type controlan cómo se utilizan las direcciones una vez que ya se ha efectuado una conexión satisfactoria.

Si start_type es auto, manual o active, y la conexión falla, el puente intenta conectarse de nuevo. Utiliza cada dirección por turnos, con un retardo de 20 segundos entre cada intento de conexión. Si `start_type` es once, y la conexión falla, el puente no intenta conectarse de nuevo automáticamente.

Si round_robin es true, la conexión de puente intenta iniciarse en la primera dirección de la lista, y lo intenta con cada dirección de la lista, una por una. Cuando se agota la lista, vuelve a empezar en la primera dirección. Si sólo hay una dirección en la lista, lo vuelve a intentar cada 20 segundos.

Si round_robin es false, se otorga preferencia a la primera dirección de la lista, que se conoce como el servidor primario. Si el primer intento de conexión al servidor primario falla, el puente sigue intentando conectarse al servidor primario en segundo plano. Al mismo tiempo, el puente intenta conectarse utilizando el resto de direcciones de la lista. Cuando los intentos de conexión al servidor primario que se efectúan en segundo plano resulten satisfactorios, el puente se desconectará de la conexión actual, y conmutará a la conexión del servidor primario.

Si una conexión se ha desconectado de forma voluntaria, por ejemplo mediante la emisión de un mandato **`connection_stop`**, en ese caso, si se reinicia la conexión, ésta intenta utilizar la misma dirección

otra vez. Si la conexión se ha desconectado debido a una anomalía de conexión o se debe a que el intermediario remoto ha finalizado la conexión, el puente espera durante 20 segundos. A continuación, trata de conectarse a la siguiente dirección de la lista, o a la misma dirección, si la lista sólo contiene una dirección.

Conexión a un gestor de colas de varias instancias

En una configuración de gestor de colas de varias instancias, el gestor de colas se ejecuta en dos servidores distintos con direcciones IP distintas. Normalmente, los canales de telemetría se configuran sin una dirección IP específica. Se configuran sólo con un número de puerto. Cuando se inicia el canal de telemetría, éste selecciona, de forma predeterminada, la primera dirección de red disponible que haya en el servidor local.

Configure el parámetro addresses de la conexión de puente con las dos direcciones IP que utilice el gestor de colas. Establezca round_robin en true.

Si la instancia de gestor de colas activo falla, el gestor de colas conmuta a la instancia que está en espera. El daemon detecta que la conexión con la instancia activa ha finalizado, e intenta conectarse de nuevo a la instancia que está en espera. Utiliza la otra dirección IP de la lista de direcciones configuradas para la conexión de puente.

El gestor de colas al cual se conecta el puente, sigue siendo el mismo gestor de colas. El gestor de colas recupera su propio estado. Si cleansession se ha establecido en false, la conexión de puente se restaura al mismo estado que tenía antes de producirse la migración tras error. La conexión se reanuda después de un retardo. Los mensajes con "al menos una vez" o "como máximo una vez" de calidad de servicio no se pierden y las suscripciones siguen funcionando.

El tiempo de reconexión depende del número de canales y de clientes que se reinicien al iniciarse la instancia que está en espera, y de cuántos mensajes había en curso. La conexión de puente puede tratar de volver a conectarse a ambas direcciones IP varias veces, antes de que se restablezca la conexión.

No configure un canal de telemetría de gestor de colas de varias instancias con una dirección IP específica. La dirección IP sólo es válida en un servidor.

Si utiliza una solución de alta disponibilidad alternativa, que gestione la dirección IP, esta opción puede ser correcta para configurar un canal de telemetría con una dirección IP específica.

cleansession

Una conexión de puente es una sesión de cliente MQTT v3. Puede controlar si una conexión se inicia una nueva sesión, o si restaura una sesión existente. Si restaura una sesión existente, la conexión de puente conserva las suscripciones y las publicaciones retenidas de la sesión anterior.

No establezca cleansession en false si figuran varias direcciones IP en addresses y éstas se conectan a canales de telemetría alojados en gestores de colas diferentes o a daemons diferentes. El estado de la sesión no se transfiere entre los gestores de colas ni los daemons. Tratar de reiniciar una sesión existente en un gestor de colas o daemon diferente da como resultado que se inicie una nueva sesión. Los mensajes dudosos se pierden, y es posible que las suscripciones no se comporten de la forma esperada.

notifications

Una aplicación puede hacer un seguimiento de si se está ejecutando la conexión de puente, utilizando las notificaciones. Una notificación es una publicación que tiene el valor `1`, conectada, o `0`, desconectada. Se publica en *SerieTema* que se define mediante el parámetro `notification_topic`. El valor predeterminado de *topicString* es $SYS/`broker`/`connection`/ *clientIdentifier*/`state`. El valor predeterminado de *SerieTema* contiene el prefijo $SYS. Puede suscribirse a los temas que empiecen por $SYS definiendo un filtro de tema que empiece por $SYS. El filtro de tema *#*, suscribirse a todo, no efectúa la suscripción a los temas que empiecen por $SYS en el daemon. Considere el caso de $SYS como si definiera un espacio de tema de sistema especial distinto del espacio de tema de la aplicación.

Las notificaciones permite que el IBM WebSphere MQ Telemetry daemon for devices notifique a los clientes MQTT cuando un puente está conectado o desconectado.

keepalive_interval

El parámetro de conexión de puente keepalive_interval establece el intervalo que el puente utiliza para enviar un ping TCP/IP al servidor remoto. El intervalo predeterminado es de 60 segundos. El envío de pings impide que el servidor remoto, o un cortafuegos, cierre la sesión TCP/IP, si se detecta un período de inactividad en la conexión.

clientid

Una conexión de puente es una sesión de cliente MQTT v3, y tiene un valor `clientIdentifier` que se ha establecido mediante el parámetro de conexión de puente clientid. Si tiene la intención de que las reconexiones reanuden una sesión anterior estableciendo el parámetro cleansession en false, el valor de `clientIdentifier` que se utilice en cada sesión deberá ser el mismo. El valor predeterminado de clientid es *nombre_host.NombreConexión*, que continúa siendo el mismo.

Instalación, verificación, configuración y control del daemon de WebSphere MQ Telemetry daemon para dispositivos

La instalación, la verificación, la configuración y el control se basan en archivos.

Instale el daemon copiando el kit de desarrollo de software en el dispositivo donde se va a ejecutar el daemon.

Por ejemplo, ejecute el programa de utilidad de cliente MQTT y conéctese al daemon de WebSphere MQ Telemetry para dispositivos como el intermediario de publicación/suscripción; consulte Utilizar el daemon de WebSphere MQ Telemetry para dispositivos como el intermediario de publicación/suscripción.

Configure el daemon mediante la creación de un archivo de configuración; consulte Archivo de configuración del daemon de WebSphere MQ Telemetry para dispositivos.

Controle un daemon que esté en ejecución mediante la creación de mandatos en el archivo `amqtdd.upd`. Cada 5 segundos, el daemon lee el archivo, ejecuta los mandatos, y suprime el archivo; consulte Archivo de mandatos del daemon de WebSphere MQ Telemetry para dispositivos.

Puertos de escucha del daemon de WebSphere MQ Telemetry para dispositivos

Conecte los clientes MQTT V3 al daemon de WebSphere MQ Telemetry para dispositivos utilizando los puertos de escucha. Puede calificar un puerto de escucha con un punto de montaje y un número máximo de conexiones.

Un puerto de escucha debe corresponder al número de puerto especificado en el método `connect(serverURI)` del cliente MQTT, de un cliente que se conecte a este puerto. Toma, como valor predeterminado 1883, tanto en el cliente como en el daemon.

Puede cambiar el puerto predeterminado para el daemon estableciendo la definición global `port` del archivo de configuración del daemon. Puede establecer puertos específicos añadiendo una definición `listener` al archivo de configuración del daemon.

Para cada puerto de escucha, que no sea el puerto predeterminado, puede especificar un punto de montaje para aislar los clientes. Los clientes conectados a un puerto que tengan un punto de montaje están aislados de otros clientes; consulte “Puntos de montaje del daemon de WebSphere MQ Telemetry para dispositivos” en la página 161.

Puede limitar el número de clientes que pueden conectarse a cualquier puerto. Establezca la definición global `max_connections` para limitar las conexiones al puerto predeterminado, o calificar cada puerto de escucha con `max_connections`.

Ejemplo

Un ejemplo de un archivo de configuración que cambia el puerto predeterminado 1883 por 1880, y limita las conexiones al puerto 1880 a 10000. Las conexiones al puerto 1884 están limitadas a 1000

conexiones. Los clientes conectados al puerto 1884 están aislados de los clientes conectados a otros puertos.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Puntos de montaje del daemon de WebSphere MQ Telemetry para dispositivos

Puede asociar un punto de montaje con un puerto de escucha que los clientes MQTT utilizan para conectarse a un daemon de WebSphere MQ Telemetry para dispositivos. Un punto de montaje aísla las publicaciones y las suscripciones que los clientes MQTT intercambian utilizando un puerto de escucha de los clientes MQTT que estén conectados a un puerto de escucha distinto.

Los clientes conectados a un puerto de escucha que tengan un punto de montaje nunca puede intercambiar temas directamente con los clientes que estén conectados a cualquier otro puerto de escucha. Los clientes que estén conectados a un puerto de escucha que no tengan un punto de montaje pueden publicar o suscribirse a temas de cualquier cliente. Los clientes no son conscientes de si están conectados a través de un punto de montaje o no; no tiene ninguna diferencia respecto a las series de tema que crean los clientes.

Un punto de montaje es una serie de texto que se prefija en la serie de tema de las publicaciones y las suscripciones. Actúa como prefijo de todas las series de tema que han creado los clientes conectados al puerto de escucha que tengan un punto de montaje. La serie de texto se elimina de todas las series de tema enviadas a los clientes que estén conectados al puerto de escucha.

Si un puerto de escucha no tiene ningún punto de montaje, no se modifican las series de tema de las publicaciones y suscripciones creadas y recibidas por los clientes conectados al puerto.

Cree series de punto de montaje con un carácter / final. De este modo, el punto de montaje es el tema padre del árbol de tema del punto de montaje.

Ejemplo

Un archivo de configuración contiene los puertos de escucha siguientes:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Un cliente, conectado al puerto 1883, crea una suscripción a `MyTopic`. El daemon registra la suscripción como `1883/MyTopic`. Otro cliente conectado al puerto 1883 publica un mensaje sobre el tema, `MyTopic`. El daemon cambia la serie de tema a `1883/MyTopic` y busca suscripciones coincidentes. El suscriptor en el puerto 1883 recibe la publicación con la serie de tema original `MyTopic`. El daemon ha eliminado el prefijo del punto de montaje de la serie de tema.

Otro cliente, conectado al puerto 1884, también publica sobre el tema `MyTopic`. Esta vez el daemon registra el tema como `1884/MyTopic`. El suscriptor en el puerto 1883 no recibe la publicación, debido a que el punto de montaje diferente da como resultado una suscripción que tiene una serie de tema diferente.

Un cliente, conectado al puerto 1885, publica sobre el tema, `1883/MyTopic`. El daemon no cambia la serie de tema. El suscriptor en el puerto 1883 recibe la publicación en `MyTopic`.

Calidad de servicio del daemon de WebSphere MQ Telemetry para dispositivos, suscripciones duraderas y publicaciones retenidas

Los valores de la calidad del servicio se aplican sólo a un daemon que esté en ejecución. Si un daemon se detiene, ya sea de una forma controlada, o debido a una anomalía, se pierde el estado de los mensajes en curso. No se puede garantizar la entrega de un mensaje al menos una vez, o como máximo una vez, si se detiene el daemon. El daemon de WebSphere MQ Telemetry para dispositivos da soporte a la persistencia limitada. Establezca el parámetro de configuración **retained_persistence** para guardar las publicaciones retenidas y las suscripciones cuando se concluye el daemon.

A diferencia de WebSphere MQ, el daemon de WebSphere MQ Telemetry para dispositivos no registra por diario los datos persistentes. El estado de la sesión, el estado de los mensajes y las publicaciones retenidas no se guardan de forma transaccional. De forma predeterminada, el daemon descarta todos los datos cuando se detiene. Puede establecer una opción para aplicar un punto de comprobación a las suscripciones y a las publicaciones retenidas periódicamente. El estado de los mensajes se pierde siempre cuando se detiene el daemon. Se pierden todas las publicaciones que no se hayan retenido.

Establezca la opción de configuración del daemon `Retained_persistence` en `true`, para guardar periódicamente en un archivo las publicaciones retenidas. Cuando se reinicia el daemon, se vuelve a crear una instancia de las publicaciones retenidas que se guardaron automáticamente la última vez. De forma predeterminada, cuando el daemon se reinicia no se vuelven a crear las instancias de los mensajes retenidos que hayan creado los clientes.

Establezca la opción de configuración del daemon `Retained_persistence` en `true`, para guardar periódicamente en un archivo las subscripciones creadas en una sesión persistente. Si se establece `Retained_persistence` en `true`, las suscripciones que los clientes creen en una sesión en la que se haya establecido `CleanSession` en `false`, se restaurará una "sesión persistente". El daemon restaura las suscripciones cuando se reinicia, y se empiezan a recibir las publicaciones. El cliente recibe las publicaciones cuando se reinicia y se ha establecido `CleanSession` en `false`. De forma predeterminada, el estado de la sesión del cliente no se guarda cuando un daemon se detiene y, por tanto, no se restauran las suscripciones, aunque el cliente establezca `CleanSession` en `false`.

`Retained_persistence` es un mecanismo de guardado automático. Es posible que no guarde las publicaciones retenidas o las suscripciones más recientes. Puede cambiar la frecuencia con que se guardan las publicaciones retenidas y las suscripciones. Establezca el intervalo entre operaciones de guardado, o el número de cambios entre operaciones de guardado, utilizando las opciones de configuración `autosave_on_changes` y `autosave_interval`.

Configuración de ejemplo para establecer la persistencia

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
# Autosave every minute
port 1882
persistence_location /tmp/
retained_persistence true
autosave_on_changes false
autosave_interval 60
```

Seguridad del daemon de WebSphere MQ Telemetry para dispositivos

El daemon de WebSphere MQ Telemetry para dispositivos puede autenticar clientes que se conecten al mismo, utilizar credenciales para conectarse a otros intermediarios, y controlar los accesos a los temas. La seguridad del daemon es limitada por el hecho de que se crea utilizando el cliente C de WebSphere MQ Telemetry, que no proporciona soporte SSL. Por consiguiente, puede que las conexiones a y desde el daemon no estén cifradas, y no se puedan autenticar utilizando los certificados.

De forma predeterminada, no hay activado ningún tipo de seguridad.

Autenticación de clientes

Los clientes MQTT pueden establecer un nombre de usuario y una contraseña utilizando los métodos `MqttConnectOptions.setUserName` y `MqttConnectOptions.setPassword`.

Autentique a un cliente que se conecta al daemon comprobando el nombre de usuario y la contraseña que un cliente proporciona respecto a las entradas del archivo de contraseñas. Para habilitar la autenticación, cree un archivo de contraseñas y establezca el parámetro `password_file` en el archivo de configuración del daemon; consulte password_file.

Establezca el parámetro `allow_anonymous` en el archivo de configuración del daemon para permitir a los clientes que se conectan sin nombre de usuario ni contraseña se conecten a un daemon que esté comprobando la autenticación; consulte allow_anonymous. Si un cliente proporciona un nombre de usuario o una contraseña, se comprueban siempre respecto al archivo de contraseñas, si se ha establecido el parámetro `password_file`.

Establezca el parámetro `clientid_prefixes` en el archivo de configuración del daemon para limitar las conexiones a clientes específicos. Los clientes deben tener `clientIdentifiers` que empiecen con uno de los prefijos indicados n el parámetro `clientid_prefixes`; consulte clientid_prefixes.

Seguridad de conexión de puente

Cada daemon de WebSphere MQ Telemetry para la conexión de puentes de dispositivos es un cliente MQTT V3. Puede establecer el nombre de usuario y la contraseña para cada conexión de puente como un parámetro de conexión de puente en el archivo de configuración del daemon; consulte username y password. Un puente puede, por tanto, autenticarse a sí mismo ante un intermediario.

Control de accesos de temas

Si los clientes se autentican, el daemon también puede proporcionar el control de acceso a temas para cada usuario. El daemon otorga el control de accesos en función de si existe coincidencia con el tema en el que un cliente publica o al que se suscribe con una serie de tema en el archivo de control de acceso; consulte acl_file.

La lista de control de accesos tiene dos partes. La primera parte controla los accesos de todos los clientes, incluidos los clientes anónimos. La segunda parte tiene una sección para cualquier usuario en el archivo de contraseñas. Lista el control de accesos específico para cada usuario.

Ejemplo

En el siguiente ejemplo se muestran los parámetros de seguridad.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

Figura 42. Archivo de configuración del daemon

```
Fred:Fredpassword
Barney:Barneypassword
```

Figura 43. Archivo de contraseñas, passwords.txt

```
topic home/public/#
topic read meters/#
user Fred
topic write meters/fred
topic home/fred/#
user Barney
topic write meters/barney
topic home/barney/#
```

Figura 44. Archivo de control de accesos, acl.txt

Resolución de problemas en los clientes MQTT

Utilice las tareas de resolución de problemas como ayuda para resolver un problema al ejecutar clientes MQTT.

Tareas relacionadas

"Rastreo y depuración del cliente Java MQTT (Paho)" en la página 173
El registrador predeterminado utiliza el recurso de registro Java estándar que se conoce como `java.util.logging` (JSR47). Puede configurarlo utilizando un archivo de configuración o bien mediante programación.

"Rastreo del cliente MQTT JavaScript" en la página 175
Puede utilizar el cliente JavaScript para recopilar el rastreo alterando la aplicación web cliente para llamar a métodos en el objeto de cliente conectado.

"Rastreo del servicio de telemetría (MQXR)" en la página 168
Siga estas instrucciones para iniciar un rastreo de la telemetría de servicio, establecer los parámetros que controlan el rastreo y buscar la salida de rastreo.

"Rastreo del cliente Java MQTT v3" en la página 170
Siga estas instrucciones para crear un rastreo de cliente Java MQTT y controlar su salida.

"Rastreo del cliente MQTT para C" en la página 171
Establezca la variable de entorno `MQTT_C_CLIENT_TRACE` para rastrear una aplicación C de cliente MQTT

"Resolución del problema: el cliente MQTT no se conecta" en la página 182
Resuelva el problema de un programa cliente MQTT que no se puede conectar al servicio de telemetría (MQXR).

"Resolución del problema: Se ha abandonado la conexión del cliente MQTT" en la página 184
Determine la causa por la que un cliente emite excepciones inesperadas `ConnectionLost` después de establecer conexión y ejecutarse correctamente durante un periodo tiempo.

"Resolución del problema: mensajes perdidos en una aplicación MQTT" en la página 185
Resuelva el problema de la pérdida de un mensaje. ¿Es el mensaje no persistente, se ha enviado a un lugar equivocado o no se ha enviado nunca? Un programa cliente codificado incorrectamente puede perder mensajes.

"Resolución del problema: el servicio de telemetría (MQXR) no se inicia" en la página 187
Corregir el problema por el que el servicio de telemetría (MQXR) no se inicia. compruebe la instalación de WebSphere MQ Telemetry y compruebe que no faltan archivos ni se han movido ni tienen los permisos equivocados. Compruebe las vías de acceso que utiliza el servicio de telemetría (MQXR) y localice los programas del servicio de telemetría (MQXR).

"Resolución del problema: El servicio de telemetría no ha llamado al módulo de inicio de sesión JAAS" en la página 188
Averigüe si el servicio de telemetría (MQXR) no está llamando al módulo de inicio de sesión JAAS y configure JAAS para corregir el problema.

"Resolución del problema: iniciar o ejecutar el daemon" en la página 191
Consulte el registro de consola del daemon de IBM WebSphere MQ Telemetry para dispositivos o bien utilice la tabla de síntomas de este tema para resolver los problemas relacionados con el daemon.

“Resolución del problema: los clientes MQTT no se conectan al daemon” en la página 192
Los clientes no se conectan al daemon o el daemon no se conecta a los demás daemons o a un canal de telemetría de WebSphere MQ.

Referencia relacionada

“Ubicación de los registros de telemetría, los registros de errores y los archivos de configuración” en la página 165
Busque los registros, los registros de errores y los archivos de configuración que utiliza IBM WebSphere MQ Telemetry.

“Códigos de razón del cliente MQTT v3 de Java” en la página 167
Busque las causas de los códigos de razón en una excepción de cliente Java MQTT v3 o throwable.

“Requisitos del sistema para utilizar las suites de cifrado de clientes SHA-2 con clientes MQTT” en la página 176
Para Java 6 desde IBM, SR13 en adelante, puede utilizar las suites de cifrado SHA-2 para proteger los canales de MQTT y las apps de cliente. Sin embargo, las suites de cifrado SHA-2 no están habilitadas de forma predeterminada hasta Java 7 a partir de IBM, SR4 , por lo que en versiones anteriores debe especificar la suite necesaria. Si va a ejecutar un cliente MQTT con su propio JRE, deberá garantizar que es compatible con suites de cifrado SHA-2. Para que las aplicaciones cliente utilicen suites de cifrado SHA-2, el cliente también debe establecer el contexto SSL en un valor que soporte Transport Layer Security (TLS) versión 1.2.

“Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL” en la página 177
Las funciones de los diferentes navegadores varían según las distintas plataformas. Conocer las diferencias resulta útil para configurar las aplicaciones, autoridades de certificados (CA) y certificados de clientes para conectar utilizando Cliente MQTT de mensajería para JavaScript con SSL y WebSockets.

Ubicación de los registros de telemetría, los registros de errores y los archivos de configuración

Busque los registros, los registros de errores y los archivos de configuración que utiliza IBM WebSphere MQ Telemetry.

Nota: Los ejemplos se han codificado para Windows. Cambie la sintaxis para ejecutar los ejemplos en Linux

Registros del extremo servidor

El asistente de instalación de IBM WebSphere MQ MQ Telemetry escribe mensajes en su registro de instalación:

```
WMQ program directory\mqxr
```

El servicio de telemetría (MQXR) graba mensajes en el registro de errores del gestor de colas de WebSphere MQ y los archivos FDC en el directorio de errores de IBM WebSphere MQ:

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

También graba un registro para el servicio de telemetría (MQXR). En el registro se muestra las propiedades con las que se ha iniciado el servicio y los errores que ha encontrado al actuar como proxy para un cliente MQTT Por ejemplo, al eliminar una suscripción que el cliente no creó. La vía de acceso del registro es:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

La configuración de ejemplo de IBM WebSphere MQ Telemetry que ha creado IBM WebSphere MQ Explorer inicia el servicio de telemetría utilizando el mandato **runMQXRService**. **runMQXRService** está en *WMQ Telemetry install directory*\bin. Graba datos en:

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Modifique **runMQXRService** para que muestre las vías de acceso configuradas para el servicio de telemetría (MQXR) o para que cree un echo de la inicialización antes de iniciar el servicio de telemetría (MQXR).

Archivos de configuración del extremo servidor

Canales de telemetría y servicio de telemetría (MQXR)

Restricción: El formato, la ubicación, el contenido y la interpretación del archivo de configuración del canal de telemetría pueden cambiar en futuros releases. Debe utilizar IBM WebSphere MQ Explorer para configurar los canales de telemetría.

IBM WebSphere MQ Explorer guarda las configuraciones de telemetría en el archivo `mqxr_win.properties` en Windowsy el archivo `mqxr_unix.properties` en Linux. Los archivos de propiedades se guardan en el directorio de configuración de telemetría:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Figura 45. Directorio de configuración del servicio de telemetría en Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figura 46. Directorio de configuración de telemetría en Linux

JVM

Establezca las propiedades Java que se pasan como argumentos al servicio de telemetría (MQXR) en el archivo, `java.properties`. Las propiedades establecidas en el archivo se pasan directamente a la JVM que ejecuta el servicio de telemetría (MQXR). Se pasan como propiedades JVM adicionales en la línea de mandatos Java. Las propiedades establecidas en la línea de mandatos tienen prioridad sobre las propiedades que se añadan a la línea de mandatos desde el archivo `java.properties`.

Busque el archivo `java.properties` en la misma carpeta que las configuraciones de telemetría, consulte Figura 45 en la página 166 y Figura 46 en la página 166.

Modifique `java.properties` especificando cada propiedad como una línea separada. Dé el formato adecuado a cada propiedad exactamente tal como lo haría para pasar la propiedad a la JVM como un argumento; por ejemplo:

```
-Xmx1024m
-Xms1024m
```

JAAS

El archivo de configuración JAAS se describe en Configuración JAAS del canal de telemetría, que incluye el archivo de configuración JAAS de ejemplo, JAAS.config, que se proporciona con IBM WebSphere MQ Telemetry.

Si configura JAAS, lo más probable es que escriba una clase para autenticar usuarios que sustituya a los procedimientos de autenticación JAAS estándar.

Para incluir la clase `Login` en la vía de acceso de clases utilizada por la vía de acceso de clases del servicio de telemetría (MQXR), proporcione un archivo de configuración WebSphere MQ `service.env` .

Establezca la vía de acceso de clases de `LoginModule` de JAAS en `service.env`. No puede utilizar la variable `%classpath%` en `service.env`. La vía de acceso de clases establecida en `service.env` se añade a la vía de acceso de clases ya establecida en la definición del servicio de telemetría (MQXR).

Muestre las vías de acceso de clases que está utilizando el servicio de telemetría (MQXR) añadiendo `echo set classpath` a `runMQXRService.bat`. La salida se envía a `mqxr.stdout`.

La ubicación predeterminada del archivo `service.env` es:

```
WMQ data directory\service.env
```

Altere temporalmente estos valores con un archivo `service.env` para cada gestor de colas en:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

La Figura 47 en la página 167 muestra un archivo `service.env` de ejemplo para utilizar el `LoginModule.class`de ejemplo.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Nota: `service.env` no debe contener ninguna variable. Sustituya el valor real de *`WMQ Install Directory`*.

Figura 47. `service.env` de ejemplo para Windows

Rastreo

Un ingeniero del servicio técnico de IBM podría pedirle que configure el rastreo; consulte "Rastreo del servicio de telemetría (MQXR)" en la página 168. Los parámetros para configurar el rastreo se almacenan en dos archivos:

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Archivos de registro del lado del cliente

La clase de persistencia de archivo predeterminada en el cliente Java SE MQTT proporcionado con IBM WebSphere MQ Telemetry crea una carpeta con el nombre: *clientIdentifier*`-tcp`*hostNamepuerto* o *clientIdentifier*`-ssl`*hostNamepuerto* en el directorio de trabajo del cliente. El nombre de carpeta le indica el `NombreHost` y el `puerto` utilizado en el intento de conexión.. La carpeta contiene mensajes que la clase de persistencia ha almacenado. Los mensajes se suprimen cuando se han entregado correctamente.

La carpeta se suprime cuando finaliza un cliente con una sesión limpia.

Si se activa el rastreo de cliente, el registro sin formato se almacena, de forma predeterminada, en el directorio de trabajo del cliente. El archivo de rastreo se llama `mqtt-`*`n`*`.trc`.

Archivos de configuración del lado del cliente

Establezca las propiedades de rastreo y SSL para el cliente Java MQTT utilizando archivos de propiedades Java, o establezca las propiedades mediante programación. Pase las propiedades al cliente Java MQTT utilizando el conmutador `-D` de JVM: por ejemplo,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Consulte "Rastreo del cliente Java MQTT v3" en la página 170.Para obtener enlaces a la documentación de la API de cliente para las bibliotecas de cliente MQTT, consulte Referencia de programación de cliente MQTT.

Códigos de razón del cliente MQTT v3 de Java

Busque las causas de los códigos de razón en una excepción de cliente Java MQTT v3 o throwable.

Tabla 5. Códigos de razón del cliente MQTT v3 de Java		
Código de razón	**Valor**	**Motivo**
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	El cliente ya está conectado.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	El cliente ya se ha desconectado.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Se arroja cuando se ha intentado llamar a `MqttClient.disconnect` desde dentro de un método `MqttCallback`.
REASON_CODE_CLIENT_DISCONNECTING	32102	El cliente se encuentra desconectado y no puede aceptar ningún trabajo nuevo.
REASON_CODE_CLIENT_EXCEPTION	0	El cliente ha encontrado una excepción.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	El cliente no está conectado al servidor.
REASON_CODE_CLIENT_TIMEOUT	32000	El cliente ha excedido el tiempo de espera de una respuesta del servidor.
REASON_CODE_FAILED_AUTHENTICATION	4	Ha fallado la autenticación con el servidor por un nombre de usuario o contraseñas erróneos.
REASON_CODE_INVALID_CLIENT_ID	2	El servidor ha rechazado el ID de cliente proporcionado.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	La versión del protocolo solicitada no está admitida en el servidor.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Error interno debido a que no hay disponible ningún ID de mensaje nuevo.
REASON_CODE_NOT_AUTHORIZED	5	No está autorizado a realizar la operación solicitada.
REASON_CODE_SERVER_CONNECT_ERROR	32103	No se ha podido conectar con el servidor.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	EL URI del servidor y la SocketFactory proporcionada no coinciden.
REASON_CODE_SSL_CONFIG_ERROR	32106	Error de configuración de SSL.
REASON_CODE_UNEXPECTED_ERROR	6	Se ha producido un error inesperado.

Rastreo del servicio de telemetría (MQXR)

Siga estas instrucciones para iniciar un rastreo de la telemetría de servicio, establecer los parámetros que controlan el rastreo y buscar la salida de rastreo.

Antes de empezar

El rastreo es una función de soporte técnico. Siga estas instrucciones si un técnico de mantenimiento de IBM le solicita que rastree su servicio de telemetría(MQXR). La documentación del producto no documenta el formato del archivo de rastreo ni indica cómo utilizarlo para depurar un cliente.

Acerca de esta tarea

Puede utilizar los mandatos IBM WebSphere MQ **strmqtrc** y **endmqtrc** para iniciar y detener el rastreo de IBM WebSphere MQ . El mandato **strmqtrc** captura el rastreo para el servicio de telemetría (MQXR). Cuando se utiliza **strmqtrc**, pueden transcurrir unos dos segundos antes de que se inicie el rastreo del servicio de telemetría. Para obtener más información sobre el servicio de rastreo de IBM WebSphere MQ, consulte Utilización del recurso de rastreo. De forma alternativa, puede rastrear el servicio de telemetría (MQXR) mediante el procedimiento siguiente:

Procedimiento

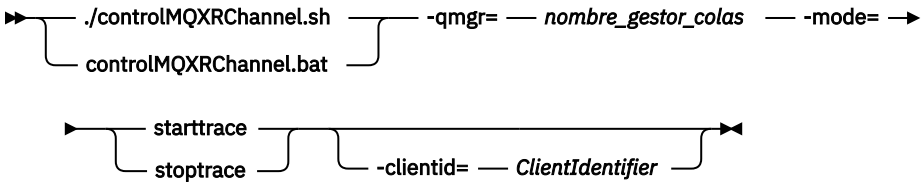
1. Establezca las opciones de rastreo para controlar el nivel de detalle y el tamaño del rastreo. Las opciones se aplican a un rastreo que se haya iniciado con los mandatos **strmqtrc** o **controlMQXRChannel**.

   Establezca las opciones de rastreo en los archivos siguientes:

   ```
   mqxrtrace.properties
   trace.config
   ```

   Los archivos se encuentran en el directorio:

   - En Windows, *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr.
   - En Linux, var/mqm/qmgrs/ *qMgrName*/mqxr.
2. Abra una ventana de mandato en el directorio siguiente:
   - En sistemas Windows , *WebSphere MQ installation directory*\mqxr\bin .
   - En sistemas Linux /opt/mqm/mqxr/bin.
3. Ejecute el mandato siguiente para iniciar un rastreo de SYSTEM.MQXR.SERVICE:



Parámetros obligatorios

qmgr=*qmgrName*
Establezca *nombre_gestor_colas* en el nombre del gestor de colas.

mode=starttrace| stoptrace
Establecer starttrace para que empiece el rastreo o en stoptrace para que finalice el rastreo

Parámetros opcionales

clientid=*ClientIdentifier*
Establezca *identificador_cliente* en el valor ClientIdentifier de un cliente. clientid filtra el rastreo a un único cliente. Ejecute el mandato de rastreo varias veces para rastrear varios clientes.

Por ejemplo:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

Resultados

Para ver la salida del rastreo, vaya al directorio siguiente:

- En Windows , *WebSphere MQ data directory*\trace.
- En los sistemas Linux, /var/mqm/trace.

Los archivos de rastreo se denominan mqxr_PPPPP.trc, donde PPPPP es el ID de proceso.

Referencia relacionada

strmqtrc

Rastreo del cliente Java MQTT v3

Siga estas instrucciones para crear un rastreo de cliente Java MQTT y controlar su salida.

Antes de empezar

Este tema sólo es aplicable a IBM WebSphere MQ versión 7.5.0.0. Para obtener información sobre cómo describir el cliente Java para versiones posteriores, consulte “Rastreo y depuración del cliente Java MQTT (Paho)” en la página 173.

El rastreo es una función de soporte técnico. Siga estas instrucciones si un ingeniero del servicio técnico de IBM le pide que rastree su cliente Java MQTT. La documentación del producto no documenta el formato del archivo de rastreo ni indica cómo utilizarlo para depurar un cliente.

El rastreo sólo funciona para el cliente Java de WebSphere MQ Telemetry.

Acerca de esta tarea

Nota: Los ejemplos están codificados para Windows. Cambie la sintaxis para ejecutar los ejemplos en Linux[2].

Procedimiento

1. Cree un archivo de propiedades Java que contenga la configuración de rastreo.

   En el archivo de propiedades, especifique las propiedades opcionales siguientes. Si se especifica la clave de propiedad más de una vez, la última vez que aparece es la que establece la propiedad.

   a) com.ibm.micro.client.mqttv3.trace.outputName

   El directorio en el que se graba el archivo de rastreo. Toma como valor predeterminado el directorio de trabajo del cliente. El archivo de rastreo se denomina mqtt-*n*.trc.

   ```
   java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
   ```

   b) com.ibm.micro.client.mqttv3.trace.count

   El número de archivos de rastreo que se graban. De forma predeterminada, se graba uno sin limitación de tamaño.

   ```
   java com.ibm.micro.client.mqttv3.trace.count=5
   ```

   c) com.ibm.micro.client.mqttv3.trace.limit

   El tamaño máximo del archivo que se graba; su valor predeterminado es 500000. El límite sólo se aplica si se solicita más de un archivo de rastreo.

   ```
   java com.ibm.micro.client.mqttv3.trace.limit=100000
   ```

   d) com.ibm.micro.client.mqttv3.trace.client.*clientIdentifier*.status

[2] Java utiliza el delimitador de vía de acceso correcto. Puede codificar el delimitador en un archivo de propiedades como '/' o '\\'; '\' es el carácter de escape

Activar o desactivar el rastreo por cliente. Si *clientIdentifier*=*, el rastreo está activado o desactivado para todos los clientes. De forma predeterminada, el rastreo está desactivado para todos los clientes.

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. Pase el archivo de propiedades de rastreo a la JVM utilizando una propiedad del sistema.

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. Ejecute el cliente.
4. Convierta el archivo de rastreo de codificación binaria a texto o `.html`. Utilice el siguiente mandato:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

donde los argumentos son:

-?
Muestra la ayuda

-i traceFile
Obligatorio. Pasa el archivo de entrada (por ejemplo, `mqtt-0.trc`).

-o outputFile
Obligatorio. Define el archivo de salida (por ejemplo, `mqtt-0.trc.html` o `mqtt-0.trc.txt`).

-h
Salida como HTML. La extensión de los archivos de salida debe ser `.html`. Si no se especifica, la salida es texto sin formato.

-d time
Sangra una línea con * si la diferencia de tiempo en milisegundos es mayor o igual que (> =) hora. No aplicable para salida HTML.

El ejemplo siguiente genera la salida del archivo de rastreo en formato HTML

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

El segundo ejemplo genera la salida del archivo de rastreo como texto sin formato, con las indicaciones de fecha y hora consecutivas que tengan milisegundos con una diferencia de 50 o superior sangradas con un asterisco (*).

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

El último ejemplo genera la salida del archivo de rastreo como texto sin formato:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Rastreo del cliente MQTT para C

Establezca la variable de entorno `MQTT_C_CLIENT_TRACE` para rastrear una aplicación C de cliente MQTT

Antes de empezar

El rastreo de cliente MQTT para C está disponible para las bibliotecas de cliente MQTT para C precompiladas en Windows y en Linux, y para las bibliotecas de iOS que cree usted.

Acerca de esta tarea

Establezca la variable de entorno `MQTT_C_CLIENT_TRACE` en la vía de acceso deseada, donde se guardará la salida del rastreo. La salida del rastreo se guarda en ese archivo.

Procedimiento

Establezca `MQTT_C_CLIENT_TRACE=mqttcclient.log` antes de ejecutar la aplicación C de cliente MQTT.

a) Por ejemplo, modifique el script de ejemplo de "Iniciación al cliente MQTT para C" en la página 26:

```
@echo off
setlocal
set MQTT_C_CLIENT_TRACE=mqttcclient.log
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

b) Ejecute el script desde el directorio `%sdkroot%/sdk/client/c/samples`.

Resultados

Los archivos de salida de rastreo comienzan con las siguientes líneas:

```
=========================================================
                    Trace Output
=========================================================
19700101 000000.000 (8084)  (1)> Socket_outInitialize:113
19700101 000000.000 (8084)   (2)> SocketBuffer_initialize:81
19700101 000000.000 (8084)   (2)< SocketBuffer_initialize:85
19700101 000000.000 (8084)  (1)< Socket_outInitialize:129
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> Thread_create_sem:189
19700101 000000.000 (8084)  (1)< Thread_create_sem:222 (0)
19700101 000000.000 (8084)  (1)> MQTTPersistence_create:43
19700101 000000.000 (8084)  (1)< MQTTPersistence_create:89 (0)
19700101 000000.000 (8084)  (1)> MQTTPersistence_initialize:104
19700101 000000.000 (8084)  (1)< MQTTPersistence_initialize:112 (0)
19700101 000000.000 (8084) (0)< MQTTClient_create:267 (0)
19700101 000000.000 (8084) (0)> MQTTClient_connect:701
19700101 000000.000 Connecting to serverURI localhost:1883
20130201 125912.234 (8084)  (1)> MQTTProtocol_connect:93
20130201 125912.234 (8084)   (2)> MQTTProtocol_addressPort:43
20130201 125912.234 (8084)   (2)< MQTTProtocol_addressPort:68
20130201 125912.234 (8084)   (2)> Socket_new:594
20130201 125912.234 New socket 1860 for localhost, port 1883
```

Tareas relacionadas

"Iniciación al cliente MQTT para C" en la página 26

Puede empezar a trabajar con el ejemplo de cliente MQTT seguro para C en cualquier plataforma en la que pueda compilar el código C fuente. Verifique que puede ejecutar el cliente MQTT de ejemplo para C con IBM MessageSight o IBM WebSphere MQ como servidor de MQTT.

Rastreo y depuración del cliente Java MQTT (Paho)

El registrador predeterminado utiliza el recurso de registro Java estándar que se conoce como `java.util.logging` (JSR47). Puede configurarlo utilizando un archivo de configuración o bien mediante programación.

Acerca de esta tarea

Nota: El cliente Paho Java sólo es aplicable a las versiones de IBM WebSphere MQ versiones 7.5.0.1 y posteriores. Para obtener información sobre el rastreo del cliente Java en IBM WebSphere MQ versión 7.5.0.0, consulte “Rastreo del cliente Java MQTT v3” en la página 170.

Nota: El rastreo es una función de soporte técnico. Siga estas instrucciones si un ingeniero de servicio de IBM le solicita que rastree el cliente Java MQTT . La documentación del producto no documenta el formato del archivo de rastreo ni indica cómo utilizarlo para depurar un cliente. El rastreo sólo funciona para el cliente Java de IBM WebSphere MQ Telemetry.

El método más sencillo para utilizar un archivo de configuración es especificar su nombre en la propiedad `java.util.logging.config.file`.

Se proporciona un archivo de propiedades de trabajo `jsr47min.properties` en el paquete `org.eclipse.paho.client.mqttv3.logging`.

El recurso de registro JSR47 se puede utilizar de distintas maneras:

- Para recopilar mensajes en un conjunto seleccionado de paquetes.
- Para recopilar mensajes desde un nivel de registro y por debajo de él.
- Para elegir varios destinos para los mensajes de registro
- Proporcionando un registro incorporado que graba un archivo y controla el tamaño y el número de archivos utilizados
- Proporcionando un registrador incorporado que se graba en la memoria y habilita los mensajes en memoria que se van a generar en base a un desencadenante
- Si la aplicación que utiliza la biblioteca de cliente MQTT también se instrumenta utilizando JSR47, los mensajes de la aplicación y la biblioteca de cliente se entremezclan

Se proporciona una clase de programa de utilidad para ayudar a recopilar información de depuración. Esta clase incluye los mensajes de registro y rastreo que se han descrito anteriormente, pero que pueden recopilar información como, por ejemplo, las propiedades del sistema Java y el valor de las variables del cliente Paho.

Se proporciona el recurso de depuración en la clase pública `Debug`, que forma parte del paquete `org.eclipse.paho.client.mqttv3.util`. Se puede obtener una instancia de depuración utilizando el método getDebug() en los objetos de cliente MQTT síncronos y asíncronos.

Por ejemplo:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

El método dumpClientDebug() vuelca la cantidad máxima de información de depuración. Debe habilitar el recurso de registro para capturar la información de depuración completa, que se graba en él. Para capturar la información de depuración completa, invoque un método de depuración cuando sepa que se va a producir el problema; por ejemplo, después de que se produzca una excepción determinada.

Procedimiento

1. Cree un archivo de configuración o bien utilice el archivo jsr47min.properties suministrado.

Si utiliza el archivo de propiedades suministrado, compruebe que el desencadenante de envío esté establecido en el nivel de error correcto. De forma predeterminada, está establecido en un error de nivel Grave, pero puede que sea necesario grabar de forma continua el rastreo en el archivo en lugar de conservarlo en la memoria hasta que se produzca un error. Para ello cambie lo siguiente:

```
java.util.logging.MemoryHandler.push=SEVERE
```

a

```
java.util.logging.MemoryHandler.push=ALL
```

2. Pase el archivo de configuración de rastreo a la JVM utilizando una propiedad del sistema.

   Si utiliza el archivo jsr4min.properties, ejecute lo siguiente:

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Ejecute el cliente.

Resultados

Cuando se produce una excepción o un problema, la clase Paho Debug graba el rastreo en memoria en el destino del archivo configurado.

El rastreo no se graba automáticamente en el archivo tal como se ha generado; esto solamente se produce cuando se pulsa el desencadenante de envío o bien cuando la clase de depuración hace que se grabe el rastreo. En este último caso, es posible que sea preciso realizar cambios en el código de aplicación.

Cada línea se graba en el manejador de archivos a medida que se va creando. Puede controlar el formato en el que se graban los mensajes configurando un FileHandler. Se proporciona un manejador de archivos personalizado con Paho que graba más que SimpleHandler y menos que el XMLHandler suministrado con JRE. Los registros de rastreo que utilizan el formateador de registro Paho tienen el formato siguiente:

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Ejemplo

Se facilita un archivo de trabajo `jsr47min.properties`. Este archivo contiene una configuración recomendada para recopilar rastreo que ayuda a resolver problemas que están relacionados con el cliente MQTT de Paho. Configura el rastreo para que se recopile de forma continua en la memoria con un impacto mínimo sobre el rendimiento. Cuando se produce el desencadenante de envío o se crea una solicitud específica para enviarla, el rastreo en memoria se envía al manejador de destino configurado. El desencadenante de envío predeterminado es un mensaje de nivel Grave, que es una conexión interrumpida. De forma predeterminada, el rastreo que se recopila en memoria se graba en el archivo especificado en este punto. De forma predeterminada, este archivo es el archivo `java.util.logging.FileHandler` estándar. Puede utilizar la clase Paho Debug para enviar el rastreo de memoria al destino.

Los detalles completos de JSR47 los puede encontrar en el Javadoc para el paquete java.util.logging.

```
# Loggers
# ----------------------------------------
# A memory handler is attached to the Paho packages
# and the level specified to collect all trace related
# to Paho packages.  This will override any root/global
# level handlers if set.
org.eclipse.paho.client.mqttv3.handlers=java.util.logging.MemoryHandler
org.eclipse.paho.client.mqttv3.level=ALL
# It is possible to set more granular trace on a per class basis e.g.
#org.eclipse.paho.client.mqttv3.internal.ClientComms.level=ALL

# Handlers
# ----------------------------------------
# Note: the target handler that is associated with the Memory Handler is not a root handler
# and hence not returned when getting the handlers from root. It appears accessing
```

```
# target handler programmatically is not possible as target is a private variable in
# class MemoryHandler
java.util.logging.MemoryHandler.level=FINEST
java.util.logging.MemoryHandler.size=10000
java.util.logging.MemoryHandler.push=SEVERE
java.util.logging.MemoryHandler.target=java.util.logging.FileHandler
#java.util.logging.MemoryHandler.target=java.util.logging.ConsoleHandler

# --- FileHandler ---
# Override of global logging level
java.util.logging.FileHandler.level=ALL

# Naming style for the output file:
# (The output file is placed in the directory
# defined by the "user.home" System property.)
# See java.util.logging for more options
java.util.logging.FileHandler.pattern=%h/paho%u.log

# Limiting size of output file in bytes:
java.util.logging.FileHandler.limit=200000

# Number of output files to cycle through, by appending an
# integer to the base file name:
java.util.logging.FileHandler.count=3

# Style of output (Simple or XML):
java.util.logging.FileHandler.formatter=org.eclipse.paho.client.mqttv3.logging.SimpleLogFormatte
r
```

Qué hacer a continuación

Para poder recopilar el rastreo mediante programación, se proporciona una clase de programa de utilidad para ayudar a recopilar información de depuración. Esta clase incluye los mensajes de registro y rastreo que se han descrito anteriormente, pero que pueden recopilar información como, por ejemplo, las propiedades del sistema Java y el valor de las variables del cliente Paho.

Se proporciona el recurso de depuración en la clase pública `Debug`, que forma parte del paquete `org.eclipse.paho.client.mqttv3.util`. Se puede obtener una instancia de depuración utilizando el método getDebug() en los objetos de cliente MQTT síncronos y asíncronos.

Por ejemplo:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

El método dumpClientDebug() vuelca la cantidad máxima de información de depuración. Debe habilitar el recurso de registro para capturar la información de depuración completa, que se graba en él. Para capturar la información de depuración completa, invoque un método de depuración cuando sepa que se va a producir el problema; por ejemplo, después de que se produzca una excepción determinada.

Rastreo del cliente MQTT JavaScript

Puede utilizar el cliente JavaScript para recopilar el rastreo alterando la aplicación web cliente para llamar a métodos en el objeto de cliente conectado.

Acerca de esta tarea

Para recopilar el rastreo, puede utilizar los métodos siguientes:

- `client.startTrace()` inicia el rastreo del cliente.
- `client.stopTrace()` detiene el rastreo del cliente.
- `client.getTraceLog()` devuelve el almacenamiento intermedio de rastreo actual.

Puede generar la salida del almacenamiento intermedio de rastreo para enviarlo a IBM Software Support. Hay diferentes formas de hacerlo. En el ejemplo se muestra el inicio del rastreo, luego la salida que se envía a la consola y a una dirección de correo específica y finalmente el rastreo que se detiene.

```
client = new Messaging.Client(location.hostname, Number(location.port), "clientId");
// Start the client tracing, the trace records capture the method calls and network
//flows from now on.
client.startTrace();

client.onConnectionLost = onConnectionLost;
client.connect({onSuccess:onConnect});

function onConnect() {
  console.log("onConnect, will now disconnect then email Trace");
  client.disconnect();
};
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0)
    console.log("ConnectionLost:"+responseObject.errorMessage);
    console.log(client.getTraceLog());
    window.location="mailto:helpdesk@"+location.hostname+
        "?Subject=Web%20Messaging%20Utility%20Trace&body="+
        client.getTraceLog().join("%0A");
    client.stopTrace();
 };
```

Salida de ejempl:

```
Client.startTrace, "2013-10-03T10:58:10.531Z", "0.0.0.0",
Client.connect, {"keepAliveInterval":60,"cleanSession":true},, false,
Client._socket_send, {"type":1,"keepAliveInterval":60,"cleanSession":true,
    "clientId":"clientId"},
Client._on_socket_message, {},
Client._on_socket_message, {"type":2,"topicNameCompressionResponse":0,"returnCode":0},
Client.disconnect,Client._socket_send, {"type":14},
Client.getTraceLog, "2013-10-03T10:58:10.548Z",
Client.getTraceLog in flight messages,
```

V 7.5.0.2 Requisitos del sistema para utilizar las suites de cifrado de clientes SHA-2 con clientes MQTT

Para Java 6 desde IBM, SR13 en adelante, puede utilizar las suites de cifrado SHA-2 para proteger los canales de MQTT y las apps de cliente. Sin embargo, las suites de cifrado SHA-2 no están habilitadas de forma predeterminada hasta Java 7 a partir de IBM, SR4 , por lo que en versiones anteriores debe especificar la suite necesaria. Si va a ejecutar un cliente MQTT con su propio JRE, deberá garantizar que es compatible con suites de cifrado SHA-2. Para que las aplicaciones cliente utilicen suites de cifrado SHA-2, el cliente también debe establecer el contexto SSL en un valor que soporte Transport Layer Security (TLS) versión 1.2.

Para Java 7 desde IBM, SR4 en adelante, las suites de cifrado SHA-2 están habilitadas de forma predeterminada. Para Java 6 desde IBM, SR13 y releases de servicio posteriores, si define un canal MQTT sin especificar una suite de cifrado, el canal no aceptará conexiones de un cliente utilizando una suite de cifrado SHA-2 . Para utilizar suites de cifrado SHA-2, debe especificar la suite deseada en la definición de canal. Esto hace que el servidor MQTT habilite la suite antes de realizar las conexiones. También significa que sólo las aplicaciones cliente que utilizan la suite especificada se pueden conectar a este canal.

Hay una limitación similar para Cliente MQTT para Java. Si el código de cliente se ejecuta en un Java 1.6 JRE de IBM, las suites de cifrado SHA-2 necesarias deben estar habilitadas explícitamente. Para utilizar estas suites, el cliente debe definir también el contexto SSL a un valor que admita la versión 1.2 del protocolo TLS (Transport Layer Security). Por ejemplo:

```
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
java.util.Properties sslClientProps = new java.util.Properties();
sslClientProps.setProperty("com.ibm.ssl.keyStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.keyStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.trustStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.trustStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
sslClientProps.setProperty("com.ibm.ssl.enabledCipherSuites",
```

```
"SSL_RSA_WITH_AES_256_CBC_SHA256" );
mqttConnectOptions.setSSLProperties(sslClientProps);
```

Como en junio de 2013, Internet Explorer 10 es el único navegador que funciona con Cliente MQTT de mensajería para JavaScript y también da soporte al protocolo TLS 1.2 , por lo que es el único navegador que puede utilizar si desea realizar conexiones SHA-2 con el cliente JavaScript .

Para obtener una lista de las suites de cifrado actualmente soportadas, consulte los enlaces relacionados.

Conceptos relacionados

"Configuración del cliente MQTT para la autenticación de cliente mediante SSL" en la página 107
Para autenticar el cliente MQTT mediante SSL, el cliente se conecta a un canal de telemetría utilizando SSL. Debe especificar un puerto TCP que corresponda a un canal de telemetría que se haya configurado para autenticar clientes SSL.

"Configuración del cliente de MQTT para la autenticación de canal mediante SSL" en la página 109
Para autenticar el canal de telemetría mediante SSL, el cliente debe conectarse al canal de telemetría utilizando SSL. Debe especificar un puerto que corresponda a un canal de telemetría que esté configurado para SSL. La configuración debe incluir un almacén de claves protegido con frase de contraseña que contenga el certificado digital firmado en privado del servidor.

V 7.5.0.1 Limitaciones de compatibilidad de navegador con las aplicaciones web de mensajerías con SSL

Las funciones de los diferentes navegadores varían según las distintas plataformas. Conocer las diferencias resulta útil para configurar las aplicaciones, autoridades de certificados (CA) y certificados de clientes para conectar utilizando Cliente MQTT de mensajería para JavaScript con SSL y WebSockets.

La mensajería en móviles que utilizan JavaScript con SSL es muy reciente, por lo que no sorprende que diferentes combinaciones de navegador y plataforma hayan implementado las funciones de diferentes formas y con diferentes extensiones. La tabla siguiente ofrece una visión general de cómo funciona actualmente con cada combinación de navegador (Firefox, Chrome, Internet Explorer y Safari) y plataforma (Windows, Linux, Mac, iOS y Android).

Tabla 6. Compatibilidad de SSL por plataforma y navegador. Para cada combinación de plataforma y navegador, la tabla especifica si son compatibles las conexiones anónimas y no anónimas de SSL y la extensión con la que trabaja el navegador con todas las autoridades certificadas (CA) y certificados de clientes.

Navegador	**Compatibilidad de SSL (S/N)**	**SSL funciona con cualquier CA (S/N)**	**Más información**
Firefox para escritorio	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Añadir el CA y el certificado de cliente al navegador. Firefox utiliza su propio almacén de certificados. Para importar un certificado de CA, pulse **Herramientas** > **Opciones** > **Avanzado** > **Cifrado** > **Ver Certificados** > **Autorizaciones** > **Importar** Para importar un certificado de cliente, pulse **Herramientas** > **Opciones** > **Avanzado** > **Cifrado** > **Ver Certificados** > **Sus Certificados** > **Importar** Para habilitar una conexión segura, especifique `https://` en el URL. Firefox le da la opción de seleccionar un certificado automáticamente o de preguntarle cada vez. Firefox también le da la opción de utilizar SSL 3.0 o TLS 1.0; compruebe que están seleccionados los dos.

Tabla 6. Compatibilidad de SSL por plataforma y navegador. Para cada combinación de plataforma y navegador, la tabla especifica si son compatibles las conexiones anónimas y no anónimas de SSL y la extensión con la que trabaja el navegador con todas las autoridades certificadas (CA) y certificados de clientes. *(continuación)*

Navegador	**Compatibilidad de SSL (S/N)**	**SSL funciona con cualquier CA (S/N)**	**Más información**
Chrome para escritorio.	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Utilice el navegador para añadir el CA y el certificado de cliente al almacén de certificados del sistema operativo, el cual se comparte con otros software. Para importar un certificado de CA, pulse **Valores** > **Mostrar valores avanzados** > **Gestionar certificados** > **Entidades de certificación raíz de confianza** > **Importar** Para importar un certificado de cliente, pulse **Valores** > **Mostrar valores avanzados** > **Gestionar certificados** > **Personal** > **Importar** Para habilitar una conexión segura, especifique `https://` en el URL. Chrome le pide que indique distintas opciones; seleccione la que corresponda en función de si va a configurar una conexión anónima o no anónima.

Tabla 6. Compatibilidad de SSL por plataforma y navegador. Para cada combinación de plataforma y navegador, la tabla especifica si son compatibles las conexiones anónimas y no anónimas de SSL y la extensión con la que trabaja el navegador con todas las autoridades certificadas (CA) y certificados de clientes. *(continuación)*

Navegador	**Compatibilidad de SSL (S/N)**	**SSL funciona con cualquier CA (S/N)**	**Más información**
Internet Explorer.	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Cuando realice una conexión SSL no anónima, se le pedirá que elija el certificado de cliente que corresponda. Internet Explorer utiliza el almacén de certificados de Windows, el cual se comparte con otro software. Para importar un certificado de CA, pulse **Herramientas** > **Opciones de Internet** > **Contenido** > **Certificados** > **Entidades de certificación raíz de confianza** > **Importar** Para importar un certificado de cliente, pulse **Herramientas** > **Opciones de Internet** > **Contenido** > **Certificados** > **Personal** > **Importar**
Safari para escritorio.	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Utilice el navegador para añadir el CA y el certificado de cliente al almacén de certificados del sistema operativo, el cual se comparte con otros software.

Tabla 6. Compatibilidad de SSL por plataforma y navegador. Para cada combinación de plataforma y navegador, la tabla especifica si son compatibles las conexiones anónimas y no anónimas de SSL y la extensión con la que trabaja el navegador con todas las autoridades certificadas (CA) y certificados de clientes. *(continuación)*

Navegador	**Compatibilidad de SSL (S/N)**	**SSL funciona con cualquier CA (S/N)**	**Más información**
Firefox en Android	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - No	No anónima: Los certificados de cliente no funcionan, porque no cumplen el requisito para añadir el CA a la lista en Firefox. Para importar un certificado de cliente, pulse **Valores** > **Seguridad** > **Almacenamiento de credenciales**. Si el certificado está firmado por un CA de confianza en la lista, puede realizar una conexión segura.
Chrome en Android	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - No	No anónima: Los certificados de cliente no funcionan, porque no cumplen el requisito para añadir el CA a la lista en Chrome. **Nota:** Plan de Google para admitir esto en la versión 27 de Chrome. Esto ha sido un error abierto desde la versión 18. Para importar un certificado de cliente, pulse **Valores** > **Seguridad** > **Almacenamiento de credenciales**. Si el certificado está firmado por un CA de confianza en la lista, puede realizar una conexión segura.

Tabla 6. Compatibilidad de SSL por plataforma y navegador. Para cada combinación de plataforma y navegador, la tabla especifica si son compatibles las conexiones anónimas y no anónimas de SSL y la extensión con la que trabaja el navegador con todas las autoridades certificadas (CA) y certificados de clientes. *(continuación)*

Navegador	Compatibilidad de SSL (S/N)	SSL funciona con cualquier CA (S/N)	Más información
Safari en iOS	Conexión SSL anónima - Sí Conexión SSL no anónima - Sí	Conexión SSL anónima - Sí Conexión SSL no anónima - No	No anónima: El dispositivo no confía en el certificado de cliente, incluso cuando el certificado de CA está instalado a la vez. Safari utiliza el almacén de certificado del dispositivo. Para importar en esta tienda, pulse **Valores** > **General** > **Perfil**y sirva el certificado de CA o de cliente desde una página web, o envíelo por correo electrónico.
Chrome en iOS	Conexión SSL anónima - Sí Conexión SSL no anónima - No	Conexión SSL anónima - No Conexión SSL no anónima - No	Acceso anónimo: Sólo las aplicaciones de Apple pueden acceder al almacén raíz del sistema de iOS. Por lo que Chrome debe utilizar su propia lista de CA, a la cual no se puede añadir. No anónima: Los certificados de cliente no funcionan, porque no cumplen el requisito para añadir el CA a la lista.

Tareas relacionadas

"Conexión de Cliente MQTT de mensajería para JavaScript sobre SSL y WebSockets" en la página 79
Conexión de la aplicación web de forma segura a IBM WebSphere MQ utilizando las páginas HTML de ejemplo de Cliente MQTT de mensajería para JavaScript con SSL y el WebSocket protocol.

Información relacionada

Mozilla: (SSL) ¿Utiliza Firefox el almacén de CA de Android CA o el suyo propio?

Chromium: Problema 134418 - Implementar compatibilidad de certificado de cliente

No se puede abrir el sitio https con certificado que no sea de confianza en ie10

Resolución del problema: el cliente MQTT no se conecta

Resuelva el problema de un programa cliente MQTT que no se puede conectar al servicio de telemetría (MQXR).

Antes de empezar

¿Está el problema en el servidor, en el cliente o en la conexión? ¿Tiene escrito su propio cliente de manejo de protocolo MQTT v3 o una aplicación de cliente MQTT utilizando los clientes MQTT de WebSphere C o Java?

Ejecute la aplicación de verificación que se proporciona con WebSphere MQ Telemetry en el servidor y compruebe que el canal de telemetría y el servicio de telemetría (MQXR) se están ejecutando correctamente. A continuación, transfiera la aplicación de verificación al cliente y ejecútela ahí.

Acerca de esta tarea

Existen una serie de razones por las que no se conecta un cliente MQTT, o el usuario puede que haya deducido que no se ha conectado, al servicio de telemetría.

Procedimiento

1. Tenga en cuenta las conclusiones que se pueden obtener a partir del código de razón que el servicio de telemetría (MQXR) ha devuelto a `MqttClient.Connect`. ¿De qué tipo de error de conexión se trata?

Opción	Descripción
REASON_CODE_INVALID_PROTOCOL_VERSION	Asegúrese de que la dirección de socket corresponde a un canal de telemetría y no ha utilizado la misma dirección para otro intermediario.
REASON_CODE_INVALID_CLIENT_ID	Compruebe que el identificador de cliente no supera los 23 bytes y que contiene sólo caracteres incluidos en: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	Compruebe que el identificador de cliente no es el mismo que el nombre del gestor de colas.
REASON_CODE_SERVER_CONNECT_ERROR	Compruebe que el servicio de telemetría (MQXR) y el gestor de colas se están ejecutado normalmente. Utilice **netstat** para comprobar que la dirección de socket no se ha asignado a otra aplicación.

   Si ha escrito una biblioteca de clientes MQTT en vez de utilizar una de las que proporciona IBM WebSphere MQ Telemetry, mire el código de retorno `CONNACK`.

   A partir de estos tres errores, puede deducir que el cliente se ha conectado al servicio de telemetría (MQXR), pero el servicio ha encontrado un error.

2. Tenga en cuenta las conclusiones que se pueden obtener a partir de los códigos de razón que el cliente genera cuando el servicio de telemetría (MQXR) no responde:

Opción	Descripción
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	Busque un archivo FDC en el servidor; consulte "Registros del extremo servidor" en la página 165. Cuando el servicio de telemetría (MQXR) detecta que el cliente ha excedido el tiempo de espera, el servicio escribe un archivo de captura de datos en primer error (FDC). Se escribe un archivo FDC cada vez que la conexión se interrumpe de forma inesperada.

   Puede que el servicio de telemetría (MQXR) no haya respondido al cliente y se haya sobrepasado el tiempo de espera del cliente. El cliente Java de WebSphere MQ Telemetry sólo se cuelga si la

aplicación ha establecido un tiempo de espera indefinido. El cliente emite una de estas excepciones si se excede el tiempo de espera definido para `MqttClient.Connect` y se devuelve un error de conexión no diagnosticado.

A menos que el usuario encuentre un archivo FDC que se correlacione con el error de conexión, el usuario no puede deducir que el cliente ha intentado conectar con el servidor:

a) Confirme que el cliente ha enviado una solicitud de conexión.

Compruebe la solicitud TCPIP con una herramienta como **tcpmon**, disponible en https://java.net/projects/tcpmon

b) ¿Coincide la dirección de socket remoto que utiliza el cliente con la definida para el canal de telemetría?

La clase de persistencia de archivo predeterminada en el cliente Java SE MQTT proporcionado con IBM WebSphere MQ Telemetry crea una carpeta con el nombre: *clientIdentifier*-`tcp`*hostNamepuerto* o *clientIdentifier*-`ssl`*hostNamepuerto* en el directorio de trabajo del cliente. El nombre de carpeta le indica el `NombreHost` y el `puerto` utilizado en el intento de conexión.; consulte “Archivos de registro del lado del cliente” en la página 167.

c) ¿Puede ejecutar ping para la dirección del servidor remoto?

d) ¿Muestra la ejecución de **netstat** en el servidor que el canal de telemetría se está ejecutando en el puerto al que se conecta el cliente?

3. Compruebe si el servicio de telemetría (MQXR) ha encontrado un problema en la solicitud del cliente.

El servicio de telemetría (MQXR) graba los errores que detecta en `mqxr.log`, y el gestor de colas graba los errores en `AMQERR01.LOG`; consulte

4. Intente localizar el problema ejecutando otro cliente.

- Ejecute la aplicación de muestra MQTT utilizando el canal de telemetría.
- Ejecute el cliente de la GUI **wmqttSample** para verificar la conexión. Obtenga **wmqttSample** descargando SupportPac IA92.

Nota: Las versiones anteriores de IA92 no incluyen la biblioteca de cliente Java MQTT v3 .

Ejecute los programas de muestra en la plataforma del servidor para eliminar las dudas que haya sobre la conexión de red y, a continuación, ejecútelos en la plataforma de cliente.

5. Otras asuntos que deben comprobarse:

a) ¿Están intentando conectarse decenas de miles de clientes MQTT a la vez?

Los canales de telemetría tienen una cola para poner en almacenamiento intermedio las conexiones entrantes acumuladas. Se procesan más de 10.000 conexiones por segundo. El tamaño del almacenamiento intermedio de conexiones pendientes se puede configurar utilizando el asistente de canal de telemetría en IBM WebSphere MQ Explorer. El tamaño predeterminado es 4096. Compruebe que el volumen de conexiones pendientes no se haya establecido en un valor bajo.

b) ¿Están todavía en ejecución el servicio de telemetría (MQXR) y el gestor de colas?

c) ¿Se ha conectado el cliente a un gestor de colas de alta disponibilidad que ha cambiado su dirección TCPIP?

d) ¿Existe algún cortafuegos que está filtrando de forma selectiva los paquetes de datos que salen o se devuelven?

Resolución del problema: Se ha abandonado la conexión del cliente MQTT

Determine la causa por la que un cliente emite excepciones inesperadas `ConnectionLost` después de establecer conexión y ejecutarse correctamente durante un periodo tiempo.

Antes de empezar

El cliente MQTT se ha conectado correctamente. El cliente puede estar activo durante periodo largo de tiempo. Los clientes se están iniciando dejando un intervalo corto entre ellos, es posible que el tiempo entre una conexión satisfactoria y la conexión que se interrumpe sea corto.

No es difícil distinguir entre conexión interrumpida y una conexión que se ha establecido correctamente y luego se ha interrumpido. Una conexión descartada se define mediante el cliente MQTT que llama al método `MqttCallback.ConnectionLost` . El método sólo se invoca después de que se haya establecido la conexión correctamente. El síntoma es diferente a la emisión de una excepción por `MqttClient.Connect` después de recibir un acuse de recibo negativo o exceder el tiempo de espera.

Si la aplicación de cliente MQTT no está utilizando las bibliotecas de cliente MQTT proporcionadas por IBM WebSphere MQ, el síntoma depende del cliente. En el protocolo MQTT v3, el síntoma es la falta de respuesta a tiempo ante una solicitud al servidor o el fallo de la conexión TCP/IP.

Acerca de esta tarea

El cliente MQTT llama a `MqttCallback.ConnectionLost` con una excepción Throwable en respuesta a cualquier problema en el lado del servidor que pueda encontrarse después de recibir un acuse de recibo positivo de la conexión. Cuando un cliente MQTT vuelve de `MqttTopic.publish` y `MqttClient.subscribe`, la solicitud se transfiere a una hebra de cliente MQTT responsable de enviar y recibir mensajes. Los errores del extremo servidor se notifican de forma asíncrona pasando una excepción Throwable al método `ConnectionLost`.

El servicio de telemetría (MQXR) siempre graba un archivo de captura de datos en primer error si se pierde la conexión.

Procedimiento

1. ¿Se ha iniciado algún otro cliente que utilizara el mismo `ClientIdentifier`?

   Si se ha iniciado un segundo cliente o se ha reiniciado el mismo cliente utilizando el mismo `ClientIdentifier`, la primera conexión con el primer cliente se elimina.
2. ¿Ha accedido el cliente a algún tema para el que no tiene autorización de publicación o suscripción?

   Cualquier acción que el servicio de telemetría emprende en nombre de un cliente y que devuelve un resultado `MQCC_FAIL` hace que el servicio elimine la conexión de cliente.

   El cliente no recibe el código de razón.

   - Busque mensajes en los archivos de registro `mqxr.log` y `AMQERR01.LOG` referentes al gestor de colas al que está conectado el cliente; consulte “Registros del extremo servidor” en la página 165.
3. ¿Se ha interrumpido la conexión TCP/IP?

   Es posible que un cortafuegos tenga un parámetro de tiempo de espera bajo para marcar como inactiva una conexión TCPIP e interrumpa la conexión.

   - Acorte el tiempo inactivo de conexión TCPIP mediante `MqttConnectOptions.setKeepAliveInterval`.

Resolución del problema: mensajes perdidos en una aplicación MQTT

Resuelva el problema de la pérdida de un mensaje. ¿Es el mensaje no persistente, se ha enviado a un lugar equivocado o no se ha enviado nunca? Un programa cliente codificado incorrectamente puede perder mensajes.

Antes de empezar

¿Está seguro de que el mensaje que envió se ha perdido? ¿Piensa que un mensaje se ha perdido porque no se ha recibido? Si el mensaje es una publicación, ¿qué mensaje se pierde: el mensaje que envía

el publicador o el que envía el suscriptor? ¿O se ha perdido la suscripción y el intermediario no está enviando publicaciones al suscriptor para esa suscripción ?

Si la solución afecta a la publicación/suscripción distribuida, la utilización de clústeres o a jerarquías de publicación/suscripción, existen muchos errores de configuración que pueden provocar la pérdida de un mensaje.

Si envía un mensaje con una calidad de servicio de "Al menos una vez" o "Como máximo una vez", es probable que el mensaje que cree que se ha perdido no se haya entregado tal y como esperaba. Es poco probable que el mensaje se haya suprimido del sistema por equivocación. Es posible que el sistema no haya creado la publicación o suscripción esperada.

El paso más importante que debe emprender en la determinación de problemas para la pérdida de mensajes es confirmar que el mensaje se ha perdido. Recree el escenario y la pérdida de más mensajes. Utilice la calidad de servicio "Al menos una vez" o "Como máximo una vez" para eliminar todos los casos en los que el sistema descarta mensajes.

Acerca de esta tarea

En el diagnóstico de la pérdida de mensajes hay cuatro puntos a tener en cuenta.

1. Mensajes "transmitir y olvidar" que funcionan como se ha previsto. El sistema descarta en ocasiones los mensajes "transmitir y olvidar".
2. Configuración: establecer la publicación/suscripción con las autorizaciones correctas en un entorno distribuido no es algo sencillo.
3. Errores de programación de clientes: la responsabilidad de la entrega de mensajes no es únicamente del código escrito por IBM.
4. Si ya ha agotado todas las posibilidades, puede tomar la decisión de involucrar al servicio de IBM.

Procedimiento

1. Si el mensaje perdido tenía la calidad de servicio de "transmitir y olvidar", defínala en "Al menos una vez" o "Como máximo una vez". Intente perder el mensaje de nuevo.
   - IBM WebSphere MQ descarta los mensajes con calidad de servicio "transmitir y olvidar" bajo una serie de circunstancias:
     - Se ha perdido la comunicación y el canal se ha detenido.
     - El gestor de colas se ha cerrado.
     - Número excesivo de mensajes.
   - La entrega de mensajes "transmitir y olvidar" depende de la fiabilidad de TCP/IP. TCP/IP continúa enviando paquetes de datos hasta recibir acuse de recibo de la entrega. Si se interrumpe la sesión TCP/IP, se pierden los mensajes con calidad de servicio "transmitir y olvidar". La interrupción de la sesión puede ser debida al cierre del cliente o el servidor, un problema de comunicaciones o un cortafuegos que ha desconectado la sesión.
2. Compruebe que el cliente está reiniciando la sesión anterior para enviar los mensajes no entregados con una calidad de servicio de "Al menos una vez" o "Como máximo una vez" de nuevo.
   a) Si la aplicación cliente utiliza el cliente MQTT de Java SE, compruebe que establece `MqttClient.CleanSession` en `false`
   b) Si utiliza diferentes bibliotecas de cliente, compruebe que la sesión se está reiniciando correctamente.
3. Compruebe que la aplicación cliente está reiniciando la misma sesión y no inicia una sesión diferente por error.

   Para volver a iniciar la misma sesión, `cleanSession = falsey Mqttclient.clientIdentifier` y `MqttClient.serverURI` deben ser los mismos que en la sesión anterior.
4. Si una sesión se cierra prematuramente, compruebe que el mensaje está disponible en el almacén de persistencia en el cliente para que pueda enviarse de nuevo.

a) Si la aplicación cliente está utilizando el cliente MQTT de Java SE, compruebe que el mensaje se está guardando en la carpeta de persistencia; consulte “Archivos de registro del lado del cliente” en la página 167

b) Si utiliza diferentes bibliotecas de clientes o ha implementado su propio mecanismo, compruebe que funciona correctamente.

5. Compruebe que nadie ha suprimido el mensaje antes de que se haya entregado.

   Los mensajes no entregados que esperan ser entregados a los clientes MQTT se almacenan en `SYSTEM.MQTT.TRANSMIT.QUEUE`. Los mensajes en espera de entrega al servidor de telemetría se almacenan mediante el mecanismo de persistencia del cliente; consulte Persistencia de mensajes en clientes MQTT.

6. Compruebe que el cliente tiene una suscripción para la publicación que espera recibir.

   Listar suscripciones utilizando WebSphere MQ Explorer, o utilizando **runmqsc** o mandatos PCF. A todas las suscripciones de cliente MQTT se les asigna un nombre. Se les asigna un nombre con el formato: *ClientIdentifier*:*Topic name*

7. Compruebe que el publicador tiene autorización para publicar y el suscriptor para suscribirse al tema de publicación.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   En un sistema de publicación/suscripción con clústeres, el suscriptor debe tener autorización para el tema en el gestor de colas al que se conecta el suscriptor. No es necesario que el suscriptor tenga autorización para suscribirse al tema en el gestor de colas en el que se ha publicado la publicación. Los canales entre los gestores de cola deben tener la autorización correcta para pasar la suscripción de proxy y reenviar la publicación.

   Cree la misma suscripción y publique en ella utilizando IBM WebSphere MQ Explorer. Mediante el programa de utilidad cliente, haga una simulación de publicación y suscripción por el cliente de aplicación. Inicie el programa de utilidad de IBM WebSphere MQ Explorer y cambie su ID de usuario para coincidir con el que ha adoptado la aplicación cliente.

8. Compruebe que el suscriptor tiene permiso para publicar en `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

9. Compruebe que la aplicación punto a punto IBM WebSphere MQ tenga autorización para colocar su mensaje en el `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

   Consulte "Envío de un mensaje a un cliente directamente" en Configurar colas distribuidas para enviar mensajes a clientes MQTT.

Resolución del problema: el servicio de telemetría (MQXR) no se inicia

Corregir el problema por el que el servicio de telemetría (MQXR) no se inicia. compruebe la instalación de WebSphere MQ Telemetry y compruebe que no faltan archivos ni se han movido ni tienen los permisos equivocados. Compruebe las vías de acceso que utiliza el servicio de telemetría (MQXR) y localice los programas del servicio de telemetría (MQXR).

Antes de empezar

La característica WebSphere MQ Telemetry está instalada. IBM WebSphere MQ Explorer tiene una carpeta de telemetría en **IBM WebSphere MQ > Gestores de colas > *qMgrNombre* > Telemetría**. Si no existe la carpeta, la instalación ha fallado.

Es necesario haber creado el servicio de telemetría (MQXR) para que se pueda iniciar. Si no se ha creado el servicio de telemetría (MQXR), ejecute el mandato **Definir configuración de ejemplo ...** asistente en la carpeta `Telemetry` .

Si el servicio de telemetría (MQXR) ya se ha iniciado anteriormente, se habrán creado otras carpetas **Channels** y **Channel Status** bajo la carpeta `Telemetry`. El servicio de telemetría, `SYSTEM.MQXR.SERVICE`, está en la carpeta **Services**. Está visible si el botón de opción del explorador está pulsado para mostrando Objetos del Sistema.

Pulse con el botón derecho del ratón en `SYSTEM.MQXR.SERVICE` para iniciar y detener el servicio, mostrar su estado y visualizar si el ID de usuario tiene autorización para iniciar el servicio.

Acerca de esta tarea

El servicio de telemetría (MQXR) `SYSTEM.MQXR.SERVICE` no se inicia. Este error de inicio se puede manifestar de dos formas diferentes:

1. El mandato de inicio falla de forma inmediata.
2. El mandato de inicio se ejecuta satisfactoriamente e inmediatamente después el servicio se detiene.

Procedimiento

1. Inicie el servicio.

   Resultado
   El servicio se detiene inmediatamente. Aparece una ventana con un mensaje de error, por ejemplo:

   ```
   WebSphere MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Razón

   Faltan archivos de la instalación o se han definido incorrectamente los permisos para archivos instalados.
   Se instala la función de IBM WebSphere MQ Telemetry sólo en uno de dos gestores de colas de alta disponibilidad. Si la instancia del gestor de colas conmuta a un sistema de reserva, intenta iniciar `SYSTEM.MQXR.SERVICE`. El mandato para iniciar el servicio falla porque el servicio de telemetría (MQXR) no está instalado en el sistema de reserva.

   Investigación
   Examine los registros de errores; consulte “Registros del extremo servidor” en la página 165.

   Acciones
   Instale, o desinstale y vuelva a instalar, la característica WebSphere MQ Telemetry.
2. Inicie el servicio, espere 30 segundos, actualice el explorador y compruebe el estado del servicio.

   Resultado
   El servicio se inicia y luego se detiene.

   Razón
   `SYSTEM.MQXR.SERVICE` ha iniciado el mandato **`runMQXRService`**, pero el mandato ha fallado.

   Investigación

   Examine los registros de errores; consulte “Registros del extremo servidor” en la página 165.
   Vea si el problema se produce únicamente con el canal de ejemplo definido.
   Realice una copia de seguridad y borre el contenido del directorio *`WMQ data directory`*`\Qmgrs\`*`qMgrName`*`\mqxr\` . Ejecute el asistente de configuración de ejemplo e intente iniciar el servicio.

   Acciones
   Busque problemas relacionados con permisos y vías de acceso.

Resolución del problema: El servicio de telemetría no ha llamado al módulo de inicio de sesión JAAS

Averigüe si el servicio de telemetría (MQXR) no está llamando al módulo de inicio de sesión JAAS y configure JAAS para corregir el problema.

Antes de empezar

Ha modificado *WMQ installation directory*\mqxr\samples\LoginModule.java para crear su propia clase de autenticación *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. Como alternativa, ha escrito sus propias clases de autenticación JAAS y las ha colocado en un directorio de su elección. Después de algunas pruebas iniciales con el servicio de telemetría (MQXR), sospecha que la clase de autenticación no es invocada por el servicio de telemetría (MQXR).

Nota: Impida que sus clases de autenticación puedan ser sobrescritas por el mantenimiento aplicado a WebSphere MQ. Utilice su propia vía de acceso para las clases de autenticación en lugar de una vía de acceso dentro del árbol de directorios de WebSphere MQ.

Acerca de esta tarea

La tarea utiliza un caso de ejemplo para ilustrar cómo resolver el problema. En el caso de ejemplo, un paquete denominado `security.jaas` contiene una clase de autenticación JAAS denominada `JAASLogin.class`. La clase se almacena en la vía de acceso `C:\WMQTelemetryApps\security\jaas`. Consulte Configuración JAAS de canales de telemetría para obtener ayuda para configurar JAAS para IBM WebSphere MQ Telemetry. El ejemplo "Configuración JAAS de ejemplo" en la página 189 muestra una configuración de ejemplo.

Procedimiento

1. Busque en `mqxr.log` una excepción generada por `javax.security.auth.login.LoginException`.

   Consulte "Registros del extremo servidor" en la página 165 para conocer la vía de acceso de `mqxr.log` y Figura 54 en la página 191 para ver un ejemplo de la excepción que aparece en el archivo de registro.
2. Corrija la configuración JAAS comparándola con la del ejemplo mostrado en "Configuración JAAS de ejemplo" en la página 189.
3. Sustituya la clase de inicio de sesión de acuerdo con el ejemplo `JAASLoginModule`, tras refactorizarla en su paquete de autenticación y desplegarla utilizando la misma vía de acceso. Cambie el valor de `loggedIn` entre `true` y `false`.

   Si el problema desaparece cuando `loggedIn` es `true`, y aparece cuando `loggedIn` es `false`, el problema se encuentra en la clase de inicio de sesión.
4. Compruebe si el problema tiene que ver con la autorización en vez de con la autenticación.
   a) Cambie la definición del canal de telemetría para que realice la comprobación de autorización mediante un ID de usuario fijo. Seleccione un ID de usuario que sea miembro del grupo `mqm`.
   b) Vuelva a ejecutar la aplicación cliente.

      Si desaparece el problema, la solución reside en el ID de usuario que se pasa para la autorización. ¿Cuál es el nombre de usuario que se pasa? Envíe el nombre a un archivo desde su módulo de inicio de sesión. Compruebe los permisos de acceso del nombre de usuario utilizando IBM WebSphere MQ Explorer, o **dspmqauth**.

Configuración JAAS de ejemplo

Utilice el asistente **Nuevo canal de telemetría** en WebSphere MQ Explorer para configurar un canal de telemetría. El cliente se conecta en el puerto 1884 y se conecta al canal de telemetría `JAASMCAUser`. En Figura 48 en la página 190 se muestra un ejemplo del archivo de propiedades de telemetría que crea e l asistente de telemetría. No edite este archivo directamente. El canal realiza la autenticación utilizando JAAS mediante la configuración denominada `JAASConfig`. Una vez que el cliente ha sido autenticado, utiliza el ID de usuario `Admin` para autorizar su acceso a objetos IBM WebSphere MQ.

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Figura 48. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties

El archivo de configuración JAAS tiene una stanza denominada `JAASConfig` que nombra la clase Java `security.jaas.JAASLogin`, que JAAS debe utilizar para autenticar clientes.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Figura 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Cuando se inicia `SYSTEM.MQTT.SERVICE`, agrega la vía de acceso de Figura 50 en la página 190 a su vía de acceso de clase.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figura 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

En Figura 51 en la página 190 se muestra la vía de acceso adicional de Figura 50 en la página 190 añadida a la vía de acceso de clases que se ha configurado para el servicio de telemetría (MQXR).

```
CLASSPATH=;C:\IBM\MQ\Program\mqxr\bin\\..\lib\MQXRListener.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\WMQCommonServices.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\objectManager.utils.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\com.ibm.micro.xr.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jmqi.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mqjms.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jar;
C:\WMQTelemtryApps;
```

Figura 51. Salida de la variable CLASSPATH de runMQXRService.bat

La salida de Figura 52 en la página 190 muestra que el servicio de telemetría (MQXR) se ha iniciado con la definición de canal que se muestra en Figura 48 en la página 190.

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCAUser value
com.ibm.mq.MQXR.Port=1884;
com.ibm.mq.MQXR.JAASConfig=JAASConfig;
com.ibm.mq.MQXR.UserName=Admin;
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Figura 52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log

Cuando la aplicación cliente se conecta al canal JAAS, si `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` no coincide con el nombre de una stanza de JAAS del archivo `jaas.config`, la conexión falla y el cliente emite una excepción con el código de retorno `0`; consulte Figura 53 en la página 191. Se ha emitido la segunda excepción, `Client is not connected (32104)`, porque el cliente ha intentado desconectarse cuando no estaba conectado.

```
C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
Userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
        at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
        at java.io.DataInputStream.readByte(DataInputStream.java:269)
        at
com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.ja
va:56)
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:90)
        ... 1 more
Client is not connected (32104)
        at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:117)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
        at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
        at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)
```

Figura 53. Excepción arrojada al conectar con com.ibm.mq.id.PubAsyncRestartable

`mqxr.log` contiene el otro resultado mostrado en Figura 53 en la página 191.

JAAS detecta el error y emite `javax.security.auth.login.LoginException` con el motivo No `LoginModules configured for JAAS`. Podría ser provocado, como en Figura 54 en la página 191, por un nombre de configuración incorrecto. También puede deberse a otros problemas que haya encontrado el servicio JAAS al cargar la configuración JAAS.

Si JAAS no notifica ninguna excepción, JASS ha cargado correctamente la clase `security.jaas.JAASLogin` designada en la stanza `JAASConfig`.

```
21/05/2010 12:06:12 [ServerWorker0] com.ibm.mq.MQXRService.MQTTCommunications
AMQXR2050E: Unable to load JAAS config: JAASWrongConfig.
The following exception occurred javax.security.auth.login.LoginException:
No LoginModules configured for JAAS
```

Figura 54. mqxr.log: error al cargar la configuración JAAS

Resolución del problema: iniciar o ejecutar el daemon

Consulte el registro de consola del daemon de IBM WebSphere MQ Telemetry para dispositivos o bien utilice la tabla de síntomas de este tema para resolver los problemas relacionados con el daemon.

Procedimiento

1. Compruebe el registro de la consola.

   Si el daemon se está ejecutando en primer plano, los mensajes de la consola se escriben en la ventana del terminal. Si el daemon se ha iniciado en segundo plano, la consola es el lugar donde se ha redirigido `stdout`.
2. Reinicie el daemon.

   Los cambios en el archivo de configuración no se aplican hasta que no se haya reiniciado el daemon.
3. Consulte Tabla 7 en la página 192:

Tabla 7. Tabla de síntomas

Problema	Solución sugerida

Cuando se inicia el daemon en Windows aparece el siguiente mensaje:
`El sistema no puede ejecutar el programa especificado`
o
`La aplicación no se ha podido iniciar porque su configuración en paralelo es incorrecta.`
	Instale Microsoft Visual C++ 2008 Redistributable Package.

Dos o más daemons o servidores capacitados con MQTT están interconectados mediante uno o más puentes, y el procesador muestra una carga excesiva.	Puede que haya un bucle de mensajes, con uno o varios mensajes pasándose repetidamente de un servidor a otro. Examine los parámetros de tema en el archivo de configuración. Cuando sea posible utilice temas más específicos. Los caracteres de comodines generales en ambas direcciones son la causa más común de bucles de conexión.
El puente no puede conectarse a un servidor capacitado con MQTT remoto al que pueden conectarse otros clientes MQTT.	Puede que el servidor remoto sea incompatible con los intentos para determinar si el servidor remoto también es un daemon de WebSphere MQ Telemetry para dispositivos. Intente definir **`try_private`** en `off` para inhabilitar el proceso especial de eliminar los bucles de mensajes.
Aparece este mensaje cuando se configura un puente: `Aviso: la conexión no ha sido el primer paquete en el socket 1888, se ha obtenido CONNACK.`	Probablemente ha configurado un puente de forma que devuelve en bucle al daemon local. La función de bucle de retorno no está soportada.

Resolución del problema: los clientes MQTT no se conectan al daemon

Los clientes no se conectan al daemon o el daemon no se conecta a los demás daemons o a un canal de telemetría de WebSphere MQ.

Acerca de esta tarea

Rastree cada uno de los paquetes MQTT enviados y recibidos por el daemon.

Procedimiento

Establezca el parámetro **`trace_output`** en `protocol` en el archivo de configuración de daemon o envíe un mandato al daemon utilizando el archivo `amqtdd.upd`.

Consulte Transferir mensajes entre el daemon de IBM WebSphere MQ Telemetry para dispositivos y IBM WebSphere MQ, para obtener un ejemplo de cómo utilizar el archivo `amqtdd.upd`.

Cuando se utiliza el valor protocol, el daemon imprime un mensaje en la consola donde describe cada uno de los paquetes de MQTT que envía y recibe.

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los Estados Unidos.

Es posible que IBM no ofrezca los productos, servicios o las características que se tratan en este documento en otros países. Consulte al representante local de IBM para obtener información sobre los productos y servicios disponibles actualmente en su zona. Las referencias a programas, productos o servicios de IBM no pretenden indicar ni implicar que sólo puedan utilizarse los productos, programas o servicios de IBM. En su lugar podrá utilizarse cualquier producto, programa o servicio equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio no IBM.

IBM puede tener patentes o solicitudes de patentes pendientes que cubran el tema principal descrito en este documento. El suministro de este documento no le otorga ninguna licencia sobre estas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director
of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Para consultas sobre licencias relacionadas con información de doble byte (DBCS), póngase en contacto con el Departamento de propiedad intelectual de IBM de su país o envíe las consultas por escrito a:

Licencias de Propiedad Intelectual
Ley de Propiedad intelectual y legal
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokio 103-8510, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones contradigan la legislación vigente: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN NINGÚN TIPO DE GARANTÍA, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUYENDO, PERO SIN LIMITARSE A, LAS GARANTÍAS IMPLÍCITAS DE NO INCUMPLIMIENTO, COMERCIABILIDAD O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas legislaciones no contemplan la exclusión de garantías, ni implícitas ni explícitas, en determinadas transacciones, por lo que puede haber usuarios a los que no les afecte dicha norma.

Esta información puede contener imprecisiones técnicas o errores tipográficos. La información aquí contenida está sometida a cambios periódicos; tales cambios se irán incorporando en nuevas ediciones de la publicación. IBM puede efectuar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento y sin previo aviso.

Cualquier referencia en esta información a sitios web que no son de IBM se realiza por razones prácticas y de ninguna manera sirve como un respaldo de dichos sitios web. Los materiales de dichos sitios web no forman parte de este producto de IBM y la utilización de los mismos será por cuenta y riesgo del usuario.

IBM puede utilizar o distribuir cualquier información que el usuario le proporcione del modo que considere apropiado sin incurrir por ello en ninguna obligación con respeto al usuario.

Los titulares de licencias de este programa que deseen información del mismo con el fin de permitir: (i) el intercambio de información entre los programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N

Rochester, MN 55901
U.S.A.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluyendo, en algunos casos, el pago de una cantidad.

El programa bajo licencia que se describe en esta información y todo el material bajo licencia disponible para el mismo lo proporciona IBM bajo los términos del Acuerdo de cliente de IBM, el Acuerdo de licencia de programas internacional de IBM o cualquier acuerdo equivalente entre las partes.

Los datos de rendimiento incluidos en este documento se han obtenido en un entorno controlado. Por consiguiente, los resultados obtenidos en otros entornos operativos pueden variar de manera significativa. Es posible que algunas mediciones se hayan realizado en sistemas en nivel de desarrollo y no existe ninguna garantía de que estas mediciones serán las mismas en sistemas disponibles generalmente. Además, algunas mediciones pueden haberse estimado por extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a su entorno específico.

La información relativa a productos que no son de IBM se obtuvo de los proveedores de esos productos, sus anuncios publicados u otras fuentes de disponibilidad pública. IBM no ha comprobado estos productos y no puede confirmar la precisión de su rendimiento, compatibilidad o alguna reclamación relacionada con productos que no sean de IBM. Las preguntas relacionadas con las posibilidades de los productos que no sean de IBM deben dirigirse a los proveedores de dichos productos.

Todas las declaraciones relacionadas con una futura intención o tendencia de IBM están sujetas a cambios o se pueden retirar sin previo aviso y sólo representan metas y objetivos.

Este documento contiene ejemplos de datos e informes que se utilizan diariamente en la actividad de la empresa. Para ilustrar los ejemplos de la forma más completa posible, éstos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con los nombres y direcciones utilizados por una empresa real es puramente casual.

LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente que ilustran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar ninguna cuota a IBM para fines de desarrollo, uso, marketing o distribución de programas de aplicación que se ajusten a la interfaz de programación de aplicaciones para la plataforma operativa para la que se han escrito los programas de ejemplo. Los ejemplos no se han probado minuciosamente bajo todas las condiciones. IBM, por tanto, no puede garantizar la fiabilidad, servicio o funciones de estos programas.

Puede que si visualiza esta información en copia software, las fotografías e ilustraciones a color no aparezcan.

Información acerca de las interfaces de programación

La información de interfaz de programación, si se proporciona, está pensada para ayudarle a crear software de aplicación para su uso con este programa.

Este manual contiene información sobre las interfaces de programación previstas que permiten al cliente escribir programas para obtener los servicios de IBM WebSphere MQ.

Sin embargo, esta información puede contener también información de diagnóstico, modificación y ajustes. La información de diagnóstico, modificación y ajustes se proporciona para ayudarle a depurar el software de aplicación.

Importante: No utilice esta información de diagnóstico, modificación y ajuste como interfaz de programación porque está sujeta a cambios.

Marcas registradas

IBM, el logotipo de IBM , ibm.com, son marcas registradas de IBM Corporation, registradas en muchas jurisdicciones de todo el mundo. Hay disponible una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas.

Microsoft y Windows son marcas registradas de Microsoft Corporation en EE.UU. y/o en otros países.

UNIX es una marca registrada de Open Group en Estados Unidos y en otros países.

Linux es una marca registrada de Linus Torvalds en Estados Unidos y en otros países.

Este producto incluye software desarrollado por Eclipse Project (http://www.eclipse.org/).

Java y todas las marcas registradas y logotipos son marcas registradas de Oracle o sus afiliados.

Número Pieza:

(1P) P/N: