7.5

Messaggistica mobile e M2M

Nota

Prima di utilizzare queste informazioni e il prodotto che supportano, leggere le informazioni in “Informazioni particolari” a pagina 189.

Questa edizione si applica alla versione 7 release 5 di IBM® WebSphere MQ e a tutte le release e modifiche successive, se non diversamente indicato nelle nuove edizioni.

Quando si inviano informazioni a IBM, si concede a IBM un diritto non esclusivo di utilizzare o distribuire le informazioni in qualsiasi modo ritenga appropriato senza incorrere in alcun obbligo verso l'utente.

Indice

Introduzione a MQTT

Ulteriori informazioni sull'invio di messaggi tra applicazioni mobili utilizzando MQ telemetry transportMQTT). Il protocollo è destinato all'uso su reti wireless e a bassa larghezza di banda. Un'applicazione mobile che utilizza MQTT invia e riceve messaggi richiamando una libreria MQTT . I messaggi vengono scambiati tramite un server di messaggistica MQTT . Il client e il server MQTT gestiscono le complessità di consegna dei messaggi in modo affidabile per l'applicazione mobile e mantengono basso il costo della gestione della rete.

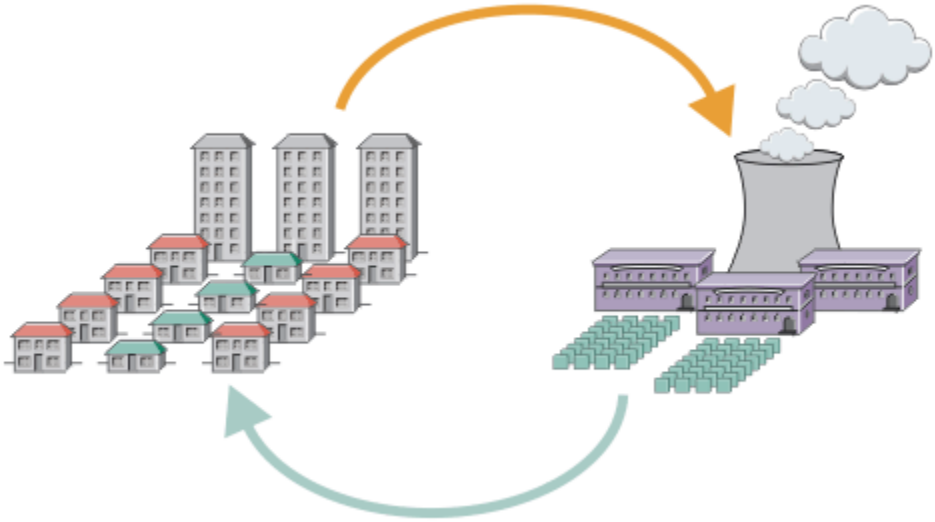
Le applicazioni MQTT vengono eseguite su dispositivi mobili, come smartphone e tablet. MQTT viene utilizzato anche per la telemetria per ricevere dati dai sensori e per controllarli in remoto. Per dispositivi mobili e sensori, MQTT offre un protocollo di pubblicazione / sottoscrizione altamente scalabile con consegna garantita. Per inviare e ricevere messaggi MQTT , aggiungere una libreria client MQTT alla propria applicazione.

La libreria del client MQTT è piccola. La libreria funziona come una casella di posta, inviando e ricevendo messaggi con altre applicazioni MQTT connesse a un server MQTT . Inviando messaggi, anziché rimanere connessi a un server in attesa di una risposta, le applicazioni MQTT conservano la durata della batteria. La libreria invia messaggi ad altre periferiche tramite un server MQTT che esegue il protocollo MQTT version 3.1 . È possibile inviare messaggi a un client specifico oppure utilizzare la messaggistica di pubblicazione / sottoscrizione per collegare più dispositivi.

Le librerie client MQTT collegano le applicazioni per periferiche mobili e sensori a un server MQTT utilizzando il protocollo MQTT .

IBM MessageSight e IBM WebSphere MQ sono server MQTT . Possono collegare grandi volumi di applicazioni client di MQTT e possono collegare insieme le reti MQTT e IBM WebSphere MQ . Consultare “Introduzione ai server MQTT” a pagina 136. IBM WebSphere MQ e IBM MessageSight possono entrambi formare un ponte tra applicazioni Web esterne in esecuzione su sensori e dispositivi mobili e altri tipi di applicazioni di pubblicazione / sottoscrizione e messaggistica in esecuzione all'interno dell'azienda. Il bridge semplifica la creazione di "soluzioni intelligenti" che integrano sensori e dispositivi mobili.

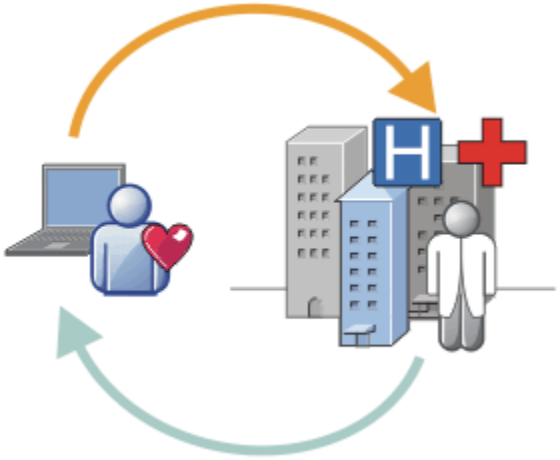
Le soluzioni intelligenti sbloccano la ricchezza di informazioni disponibili su Internet per le applicazioni in esecuzione su dispositivi mobili e sensori. Due esempi di applicazioni intelligenti basate sulla telemetria sono l'elettricità intelligente e i servizi sanitari intelligenti.



- Un messaggio MQTT che contiene i dati di utilizzo dell'energia inviati al service provider.
- Un'applicazione di telemetria invia comandi di controllo basati sull'analisi dei dati di utilizzo dell'energia.
- Per ulteriori informazioni, consultare Scenario di telemetria: Monitoraggio e controllo dell'energia domestica.

Figura 1. Misurazione intelligente dell'elettricità

Figura 2. Monitoraggio sanitario intelligente



- Un'applicazione di telemetria invia dati sanitari all'ospedale e al medico.
- Gli avvisi o i feedback dei messaggi MQTT possono essere inviati in base all'analisi dei dati di integrità.
- Per ulteriori informazioni, consultare Scenario di telemetria: monitoraggio del paziente a domicilio.

Puoi creare MQTT in piccoli dispositivi scrivendo la tua applicazione per MQTT protocol. Per aiutarti a farlo, IBM fornisce librerie client che supportano le applicazioni in esecuzione su MQTT. Consultare “Introduzione ai client MQTT” a pagina 11. IBM fornisce le librerie client per le applicazioni iOS e per le applicazioni Android V 7.5.0.1 e un JavaScript client browser per applicazioni web indipendenti dalla piattaforma. V 7.5.0.1 Le pagine del JavaScript client si collegano a IBM MessageSight e IBM WebSphere MQ con il protocollo MQTT su WebSockets. IBM fornisce anche MQTT applicazioni di esempio per C e Java su Linux® e Windows.

Le librerie C e Java vengono eseguite su iOS, Android, Windowse su diverse piattaforme UNIX and Linux . È possibile portare il codice di origine C per la libreria client MQTT su altre piattaforme. Le librerie client

MQTT per C e Java sono disponibili con una licenza open source dal progetto Eclipse Paho . Consultare Eclipse Paho. La specifica MQTT protocol è aperta e disponibile da MQTT.org.

MQTT protocol

MQTT protocol è leggero nel senso che i client sono piccoli e utilizza la larghezza di banda della rete in modo efficace. Il protocollo MQTT supporta la consegna garantita e i trasferimenti fire - and - forget. Nel protocollo, la consegna del messaggio viene disaccoppiata dall'applicazione. L'estensione del disaccoppiamento in un'applicazione dipende dal modo in cui vengono scritti un client MQTT e un server MQTT . La distribuzione disaccoppiata libera un'applicazione da qualsiasi connessione server e dall'attesa di messaggi. Il modello di interazione è simile all'email, ma ottimizzata per la programmazione delle applicazioni.

Il protocollo MQTT V3.1 viene pubblicato; consultare MQTT V3.1 Protocol Specification. La specifica identifica una serie di caratteristiche distintive del protocollo:

- È un protocollo di pubblicazione / sottoscrizione.

  Oltre a fornire la distribuzione dei messaggi uno - a - molti, le applicazioni di pubblicazione / sottoscrizione vengono disaccodati. Entrambe le funzioni sono utili nelle applicazioni che hanno molti client.
- Non dipende in alcun modo dal contenuto del messaggio.
- Viene eseguito su TCP/IP, che fornisce la connettività di rete di base.
- Ha tre qualità di servizio per la consegna dei messaggi:

  "Al massimo una volta"
  I messaggi vengono consegnati in base ai migliori sforzi della rete Internet Protocol sottostante. Potrebbe verificarsi una perdita di messaggi.

  Utilizzare questa QoS (quality of service) con i dati dei sensori dell'ambiente di comunicazione, ad esempio. Non importa se una singola lettura viene persa, se la successiva viene pubblicata subito dopo.

  "Almeno una volta"
  L'arrivo dei messaggi è sicuro, ma potrebbero verificarsi dei duplicati.

  "Esattamente una volta"
  I messaggi sono sicuri di arrivare esattamente una volta.

  Utilizzare questa qualità di servizio con i sistemi di fatturazione, ad esempio. Messaggi duplicati o persi potrebbero causare disagi o imporre addebiti errati.
- È economico nel modo in cui gestisce il flusso di messaggi sulla rete. Ad esempio, l'intestazione a lunghezza fissa è lunga solo 2 byte e gli scambi di protocollo sono ridotti al minimo per ridurre il traffico di rete.
- Dispone di una funzione "Ultimo testamento" che notifica ai sottoscrittori la disconnessione anomala di un client dal server MQTT . Consultare “Ultimo testamento e pubblicazione testamentaria” a pagina 129.

MQTT version 3.1 è supportato da IBM IBM WebSphere MQ e IBM MessageSight. MQTT è implementato su TCP/IP. Un'altra versione del protocollo, MQTT-S, è disponibile per reti non TCP/IP. Consultare MQTT-S version 1.2 - specifica.

Comunità MQTT

IBM sta eseguendo IBM Developer Messaggistica comunità per sviluppatori MQTT che stanno scrivendo applicazioni per IBM MessageSight e IBM WebSphere MQ.

MQTT.org è un buon posto per conoscere e discutere le implementazioni e le estensioni del protocollo MQTT .

MQTT è un progetto Eclipse open source, in Eclipse Technology Project. La community Paho sta sviluppando client e server open source. Consultare Eclipse Paho.

Introduzione a MQTT

Ulteriori informazioni sull'invio di messaggi tra applicazioni mobili utilizzando MQ telemetry transportMQTT). Il protocollo è destinato all'uso su reti wireless e a bassa larghezza di banda. Un'applicazione mobile che utilizza MQTT invia e riceve messaggi richiamando una libreria MQTT . I messaggi vengono scambiati tramite un server di messaggistica MQTT . Il client e il server MQTT gestiscono le complessità di consegna dei messaggi in modo affidabile per l'applicazione mobile e mantengono basso il costo della gestione della rete.

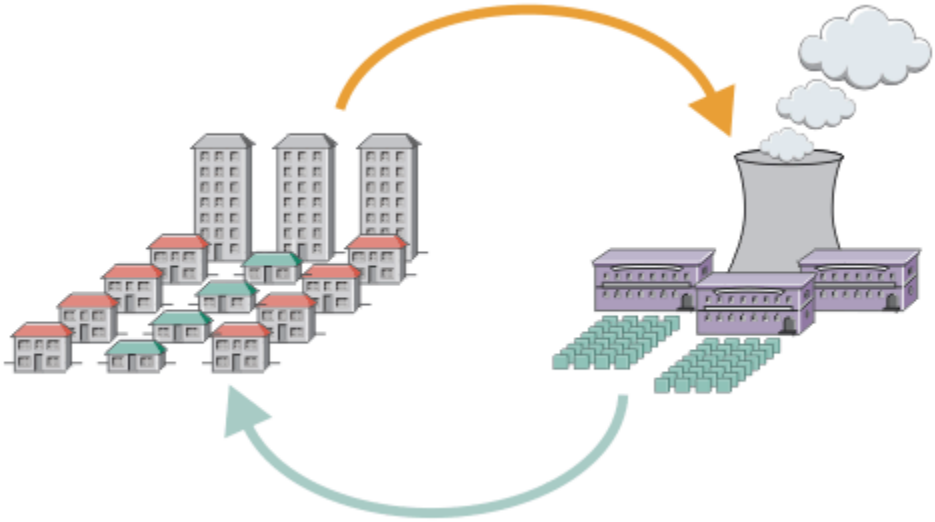
Le applicazioni MQTT vengono eseguite su dispositivi mobili, come smartphone e tablet. MQTT viene utilizzato anche per la telemetria per ricevere dati dai sensori e per controllarli in remoto. Per dispositivi mobili e sensori, MQTT offre un protocollo di pubblicazione / sottoscrizione altamente scalabile con consegna garantita. Per inviare e ricevere messaggi MQTT , aggiungere una libreria client MQTT alla propria applicazione.

La libreria del client MQTT è piccola. La libreria funziona come una casella di posta, inviando e ricevendo messaggi con altre applicazioni MQTT connesse a un server MQTT . Inviando messaggi, anziché rimanere connessi a un server in attesa di una risposta, le applicazioni MQTT conservano la durata della batteria. La libreria invia messaggi ad altre periferiche tramite un server MQTT che esegue il protocollo MQTT version 3.1 . È possibile inviare messaggi a un client specifico oppure utilizzare la messaggistica di pubblicazione / sottoscrizione per collegare più dispositivi.

Le librerie client MQTT collegano le applicazioni per periferiche mobili e sensori a un server MQTT utilizzando il protocollo MQTT .

IBM MessageSight e IBM WebSphere MQ sono server MQTT . Possono collegare grandi volumi di applicazioni client di MQTT e possono collegare insieme le reti MQTT e IBM WebSphere MQ . Consultare “Introduzione ai server MQTT” a pagina 136. IBM WebSphere MQ e IBM MessageSight possono entrambi formare un ponte tra applicazioni Web esterne in esecuzione su sensori e dispositivi mobili e altri tipi di applicazioni di pubblicazione / sottoscrizione e messaggistica in esecuzione all'interno dell'azienda. Il bridge semplifica la creazione di "soluzioni intelligenti" che integrano sensori e dispositivi mobili.

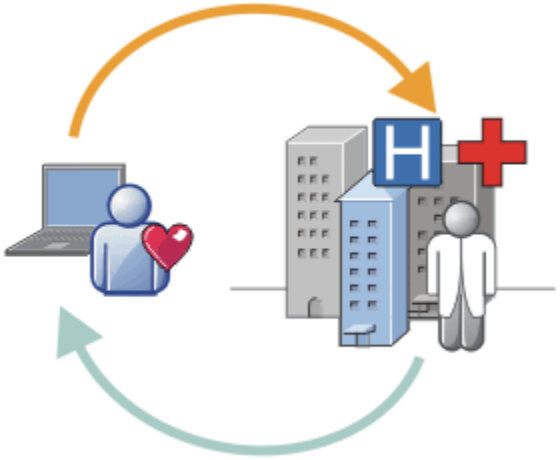
Le soluzioni intelligenti sbloccano la ricchezza di informazioni disponibili su Internet per le applicazioni in esecuzione su dispositivi mobili e sensori. Due esempi di applicazioni intelligenti basate sulla telemetria sono l'elettricità intelligente e i servizi sanitari intelligenti.



- Un messaggio MQTT che contiene i dati di utilizzo dell'energia inviati al service provider.
- Un'applicazione di telemetria invia comandi di controllo basati sull'analisi dei dati di utilizzo dell'energia.
- Per ulteriori informazioni, consultare Scenario di telemetria: Monitoraggio e controllo dell'energia domestica.

Figura 3. Misurazione intelligente dell'elettricità

Figura 4. Monitoraggio sanitario intelligente



- Un'applicazione di telemetria invia dati sanitari all'ospedale e al medico.
- Gli avvisi o i feedback dei messaggi MQTT possono essere inviati in base all'analisi dei dati di integrità.
- Per ulteriori informazioni, consultare Scenario di telemetria: monitoraggio del paziente a domicilio.

Puoi creare MQTT in piccoli dispositivi scrivendo la tua applicazione per MQTT protocol. Per aiutarti a farlo, IBM fornisce librerie client che supportano le applicazioni in esecuzione su MQTT. Consultare “Introduzione ai client MQTT” a pagina 11. IBM fornisce le librerie client per le applicazioni iOS e per le applicazioni Android V 7.5.0.1 e un JavaScript client browser per applicazioni web indipendenti dalla piattaforma. V 7.5.0.1 Le pagine del JavaScript client si collegano a IBM MessageSight e IBM WebSphere MQ con il protocollo MQTT su WebSockets. IBM fornisce anche MQTT applicazioni di esempio per C e Java su Linux e Windows.

Le librerie C e Java vengono eseguite su iOS, Android, Windowse su diverse piattaforme UNIX and Linux . È possibile portare il codice di origine C per la libreria client MQTT su altre piattaforme. Le librerie client

MQTT per C e Java sono disponibili con una licenza open source dal progetto Eclipse Paho . Consultare Eclipse Paho. La specifica MQTT protocol è aperta e disponibile da MQTT.org.

MQTT protocol

MQTT protocol è leggero nel senso che i client sono piccoli e utilizza la larghezza di banda della rete in modo efficace. Il protocollo MQTT supporta la consegna garantita e i trasferimenti fire - and - forget. Nel protocollo, la consegna del messaggio viene disaccoppiata dall'applicazione. L'estensione del disaccoppiamento in un'applicazione dipende dal modo in cui vengono scritti un client MQTT e un server MQTT . La distribuzione disaccoppiata libera un'applicazione da qualsiasi connessione server e dall'attesa di messaggi. Il modello di interazione è simile all'email, ma ottimizzata per la programmazione delle applicazioni.

Il protocollo MQTT V3.1 viene pubblicato; consultare MQTT V3.1 Protocol Specification. La specifica identifica una serie di caratteristiche distintive del protocollo:

- È un protocollo di pubblicazione / sottoscrizione.

  Oltre a fornire la distribuzione dei messaggi uno - a - molti, le applicazioni di pubblicazione / sottoscrizione vengono disaccodati. Entrambe le funzioni sono utili nelle applicazioni che hanno molti client.
- Non dipende in alcun modo dal contenuto del messaggio.
- Viene eseguito su TCP/IP, che fornisce la connettività di rete di base.
- Ha tre qualità di servizio per la consegna dei messaggi:

  "Al massimo una volta"
  I messaggi vengono consegnati in base ai migliori sforzi della rete Internet Protocol sottostante. Potrebbe verificarsi una perdita di messaggi.

  Utilizzare questa QoS (quality of service) con i dati dei sensori dell'ambiente di comunicazione, ad esempio. Non importa se una singola lettura viene persa, se la successiva viene pubblicata subito dopo.

  "Almeno una volta"
  L'arrivo dei messaggi è sicuro, ma potrebbero verificarsi dei duplicati.

  "Esattamente una volta"
  I messaggi sono sicuri di arrivare esattamente una volta.

  Utilizzare questa qualità di servizio con i sistemi di fatturazione, ad esempio. Messaggi duplicati o persi potrebbero causare disagi o imporre addebiti errati.
- È economico nel modo in cui gestisce il flusso di messaggi sulla rete. Ad esempio, l'intestazione a lunghezza fissa è lunga solo 2 byte e gli scambi di protocollo sono ridotti al minimo per ridurre il traffico di rete.
- Dispone di una funzione "Ultimo testamento" che notifica ai sottoscrittori la disconnessione anomala di un client dal server MQTT . Consultare “Ultimo testamento e pubblicazione testamentaria” a pagina 129.

MQTT version 3.1 è supportato da IBM IBM WebSphere MQ e IBM MessageSight. MQTT è implementato su TCP/IP. Un'altra versione del protocollo, MQTT-S, è disponibile per reti non TCP/IP. Consultare MQTT-S version 1.2 - specifica.

Comunità MQTT

IBM sta eseguendo IBM Developer Messaggistica comunità per sviluppatori MQTT che stanno scrivendo applicazioni per IBM MessageSight e IBM WebSphere MQ.

MQTT.org è un buon posto per conoscere e discutere le implementazioni e le estensioni del protocollo MQTT .

MQTT è un progetto Eclipse open source, in Eclipse Technology Project. La community Paho sta sviluppando client e server open source. Consultare Eclipse Paho.

Introduzione ai client MQTT

Puoi iniziare a sviluppare un'applicazione mobile o M2M (machine-to-machine) creando ed eseguendo un'applicazione client MQTT di esempio che utilizza una libreria client MQTT . Le applicazioni di esempio e librerie client associate sono disponibili in Mobile Messaging and M2M Pacchetto client da IBM. Ci sono versioni delle app e delle librerie client scritte in Java, in JavaScripte in C. Puoi eseguire queste applicazioni sulla maggior parte delle piattaforme e dei dispositivi, inclusi i dispositivi e i prodotti Android da Apple.

Prima di iniziare

Per creare ed eseguire la tua app, hai bisogno di una certa esperienza di creazione di app per il dispositivo o la piattaforma di destinazione e il linguaggio di programmazione utilizzato. Una piccola esperienza è di solito sufficiente per ottenere un'applicazione di esempio attiva e in esecuzione sul dispositivo o sulla piattaforma scelti.

Se utilizzi un server MQTT di livello aziendale come IBM WebSphere MQ o IBM MessageSight, puoi scambiare le informazioni dalla tua applicazione di esempio con le tue applicazioni aziendali esistenti.

Informazioni su questa attività

Gli obiettivi sono i seguenti:

1. Scegliere un server MQTT a cui è possibile connettere l'app client.
2. Scarica Mobile Messaging and M2M Pacchetto client.
3. Creare, per il dispositivo o la piattaforma di destinazione, le applicazioni di esempio dal pacchetto del client.
4. Verificare che gli esempi si comportino come previsto collegandoli al server MQTT .

Come risultato della creazione e del test delle applicazioni di esempio per il dispositivo o la piattaforma, puoi creare un ambiente di sviluppo funzionante che puoi quindi utilizzare per creare le tue proprie applicazioni client.

Mobile Messaging and M2M Pacchetto client contiene MQTT SDK. Questo SDK ti fornisce le seguenti risorse:

- Applicazioni client MQTT di esempio scritte in Java, in JavaScripte in C.
- Le librerie client MQTT che supportano queste applicazioni client e ne consentono l'esecuzione sulla maggior parte delle piattaforme e dei dispositivi.

L'SDK include anche il codice origine per Client MQTT per C. È possibile adattare questo codice sorgente per creare librerie client MQTT per C per altre piattaforme. Per ulteriori informazioni, consultare “Creazione del client MQTT per le librerie C” a pagina 31. Il codice sorgente per Client MQTT per C è disponibile anche con una licenza open source da Eclipse Paho.

Procedura

I seguenti articoli ti guidano attraverso la procedura specifica della piattaforma per la creazione e l'esecuzione di un'app MQTT di esempio su un computer desktop o su un dispositivo mobile per Android o da Apple:

- “Introduzione al client MQTT per Java” a pagina 12
- “Introduzione a Client MQTT per Java su Android” a pagina 18
- V 7.5.0.1 “Introduzione a MQTT messaging client per JavaScript” a pagina 24
- “Introduzione al client MQTT per C” a pagina 26
- “Introduzione al client MQTT per C su iOS” a pagina 47

Operazioni successive

Per sviluppare una nuova applicazione MQTT , devi avere o acquisire le seguenti competenze:

- Programmazione nel linguaggio richiesto per il dispositivo o la piattaforma.
- Programmazione per la periferica o piattaforma di destinazione.
- Progettazione di applicazioni di pubblicazione / sottoscrizione.
- Progettazione di programmi per il modello di programmazione MQTT .
- Progettazione di programmi da eseguire sul dispositivo mobile scelto.
- Utilizzo di SSL e JAAS per proteggere i programmi.

Non hai bisogno di alcuna capacità di programmazione di rete per collegare un client MQTT con un altro dispositivo o applicazione, perché MQTT è un sistema di messaggistica e accodamento. Le librerie client MQTT gestiscono le connessioni di rete per la tua applicazione.

Per integrare il tuo client MQTT con le applicazioni enterprise esistenti, hai due scelte. È possibile condividere gli argomenti di pubblicazione / sottoscrizione di MQTT con (ad esempio) un'applicazione IBM WebSphere MQ o JMS oppure è possibile scrivere il proprio adattatore di integrazione come un altro client MQTT .

Le fonti di informazione da consultare oggi sono:

- Sviluppo di applicazioni per WebSphere MQ Telemetry
- MQTT.org
- Eclipse Paho

Concetti correlati

“Introduzione ai server MQTT” a pagina 136

Introduzione al client MQTT per Java

Attivazione ed esecuzione con il client MQTT per applicazioni di esempio Java , utilizzando IBM MessageSight o IBM WebSphere MQ come server MQTT. Le applicazioni di esempio utilizzano una libreria client da SDK (software development toolkit) MQTT da IBM. L'applicazione di esempio `SampleAsyncCallBack` è un modello per la scrittura di applicazioni MQTT per Android e altri sistemi operativi basati su eventi.

Prima di iniziare

- Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile". Consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.
- Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .

Informazioni su questa attività

Lo scopo dell'attività è verificare che sia possibile creare ed eseguire un client MQTT per l'applicazione di esempio Java , connetterlo a IBM WebSphere MQ o IBM MessageSight come server MQTT version 3 e scambiare messaggi.

Seguire questa attività per eseguire l'applicazione di esempio dal workbench Eclipse o da una riga comandi. I passaggi nell'esempio sono per Windows. Con piccole modifiche, è possibile eseguire l'applicazione di esempio su qualsiasi piattaforma che supporta JSE 1.5 o superiore.

È possibile eseguire le applicazioni sullo stesso server di IBM WebSphere MQ, in cui è configurato l'ambiente per l'esecuzione delle applicazioni che si connettono a IBM WebSphere MQ . Seguire l'attività “Configurazione del servizio MQTT dalla riga di comando” a pagina 140 per installare e configurare IBM WebSphere MQ con l'opzione IBM WebSphere MQ Telemetry su Windows o Linux. Quando l'ambiente è installato e configurato, eseguire l'applicazione di esempio, `MQTTV3Sample`, per verificare l'installazione.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 . Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.
2. Opzionale: Configurare il server MQTT .
   - Su IBM WebSphere MQ, è necessario completare una o l'altra delle seguenti attività per impostare un gestore code e configurare il servizio di telemetria (MQXR):
     - “Configurazione del servizio MQTT dalla riga di comando” a pagina 140
     - “Configurazione del servizio MQTT con IBM WebSphere MQ Explorer” a pagina 143
   - Su altri server, consultare la documentazione del server. Non è richiesta alcuna procedura di configurazione per Really Small Message Broker. Consultare Really Small Message Broker.
3. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.

   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.

   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*\SDK.
4. Installare un kit di sviluppo Java (JDK) Versione 6 o successiva.

   Poiché stai sviluppando un'applicazione Java for Android, il JDK deve provenire da Oracle. È possibile ottenere JDK da Download di Java SE
5. Compilare ed eseguire uno o più client MQTT per applicazioni di esempio Java :
   - “Compilare ed eseguire i programmi di esempio Paho dalla riga comandi” a pagina 14
   - “Compila ed esegui tutte le applicazioni MQTT di esempio client Java da Eclipse” a pagina 16
   - “Introduzione a Client MQTT per Java su Android” a pagina 18

   Le seguenti applicazioni MQTT di esempio client Java sono incluse nell'SDK:

   `MQTTV3Sample`
   L'esempio è incluso anche con IBM WebSphere MQ e link al package `com.ibm.micro.client.mqttv3.jar` .

   `Sample`
   `Sample` si trova nel pacchetto Paho e si collega al pacchetto `org.eclipse.paho.client.mqttv3` . È simile a `MQTTV3Sample`; attende il completamento di ogni azione MQTT .

   `SampleAsyncWait`
   `SampleAsyncWait` si trova nel pacchetto `org.eclipse.paho.client.mqttv3`. Utilizza la API MQTT asincrona; attende su un thread differente fino al completamento di un'azione. Il thread principale può svolgere altro lavoro finché non si sincronizza sul thread che sta attendendo il completamento dell'azione MQTT.

   `SampleAsyncCallBack`
   `SampleAsyncCallBack` si trova nel pacchetto `org.eclipse.paho.client.mqttv3`. Richiama l'API MQTT asincrona. L'API asincrona non attende MQTT per completare l'elaborazione di una chiamata; ritorna all'applicazione. L'applicazione continua con le altre attività e attende quindi che arrivi l'evento successivo di cui eseguire l'elaborazione. MQTT invia una notifica di evento nuovamente all'applicazione quando completa l'elaborazione. L'interfaccia MQTT guidata dagli eventi è adatta al modello di programmazione di servizi e attività di Android e altri sistemi operativi guidati dagli eventi.

Come esempio, vedere come l'esempio `mqttExerciser` integra MQTT in Android utilizzando il modello di programmazione di servizi e attività.

`mqttExerciser`
`mqttExerciser` è un programma di esempio per Android. Poiché è stato creato ed eseguito in modo diverso, viene descritto separatamente. Consultare “Introduzione a Client MQTT per Java su Android” a pagina 18.

Risultati

Sono state compilate ed eseguite le applicazioni di esempio di MQTT Java connesse a IBM WebSphere MQ o IBM MessageSight come server MQTT .

Operazioni successive

Esaminare le informazioni di riferimento Javadoc ; consultare il passo “3” a pagina 16 di “Compila ed esegui tutte le applicazioni MQTT di esempio client Java da Eclipse” a pagina 16. In alternativa, aprire i file html Javadoc presenti nella directory SDK`\clients\java\doc\javadoc` in Mobile Messaging and M2M Pacchetto client.

Compilare ed eseguire i programmi di esempio Paho dalla riga comandi

Compila ed esegui l' Paho applicazione di esempio `Sample.java` dalla riga di comandi. L'esempio è in MQTT SDK. L'esempio viene creato con le librerie del client MQTT Paho nel pacchetto `org.eclipse.paho.client.mqttv3` . Illustra un publisher e un sottoscrittore MQTT . Due altri esempi Paho nella stessa directory possono essere creati ed eseguiti nello stesso modo. Differiscono chiamando la libreria MQTT in modo asincrono.

Prima di iniziare

Eseguire i passaggi da “1” a pagina 13 a “4” a pagina 13 nell'attivit ... principale per configurare un server MQTT e scaricare Mobile Messaging and M2M Pacchetto client.

Informazioni su questa attività

Compila ed esegui `Sample.java` dalla sottodirectory degli esempi client SDK SDK`\clients\java\samples`. Il codice Java si trova nella directory SDK`\clients\java\samples\org\eclipse\paho\sample\mqttv3app`.

Procedura

1. Creare uno script nella directory degli esempi client per compilare ed eseguire `Sample` sulla piattaforma scelta.

   Il seguente script compila ed esegue l'esempio su 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. Compila ed esegui `Sample.java`
2. Esegui lo script.

Risultati:

```
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. MQTTV3Sample Abbonato

```
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. MQTTV3Sample Publisher

Se si lascia l'applicazione del sottoscrittore in esecuzione, non è possibile eseguire nuovamente lo stesso sottoscrittore. L'identificativo client del nuovo sottoscrittore è lo stesso del vecchio sottoscrittore. Non è possibile eseguire due client MQTT con lo stesso identificativo client contemporaneamente. È possibile impostare l'opzione `-i` per impostare l'identificativo client in modo da poter eseguire diversi sottoscrittori contemporaneamente.

Se si esegue nuovamente lo stesso client, è possibile utilizzare l'opzione `-c` per avviare e avviare il client con `cleansession` impostato su false. Con tale opzione è possibile esplorare il comportamento delle sessioni client interrotte.

3. Terminare il sottoscrittore premendo Invio o chiudendo la finestra.

Operazioni successive

Creare gli script per compilare ed eseguire gli altri esempi nella sottodirectory `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` . Copiare lo script in Figura 5 a pagina 14 e sostituire `Sample` con `SampleAsyncWait` o `SampleAsyncCallBack`. Gli altri esempi sono versioni asincrone del programma `Sample` sincrono.

`SampleAsyncWait`
`SampleAsyncWait` si trova nel pacchetto `org.eclipse.paho.client.mqttv3`. Utilizza la API MQTT asincrona; attende su un thread differente fino al completamento di un'azione. Il thread principale può svolgere altro lavoro finché non si sincronizza sul thread che sta attendendo il completamento dell'azione MQTT.

`SampleAsyncCallBack`
`SampleAsyncCallBack` si trova nel pacchetto `org.eclipse.paho.client.mqttv3`. Richiama l'API MQTT asincrona. L'API asincrona non attende MQTT per completare l'elaborazione di una chiamata; ritorna all'applicazione. L'applicazione continua con le altre attività e attende quindi che arrivi l'evento successivo di cui eseguire l'elaborazione. MQTT invia una notifica di evento nuovamente all'applicazione quando completa l'elaborazione. L'interfaccia MQTT guidata dagli eventi è adatta al modello di programmazione di servizi e attività di Android e altri sistemi operativi guidati dagli eventi.

Come esempio, vedere come l'esempio `mqttExerciser` integra MQTT in Android utilizzando il modello di programmazione di servizi e attività.

Gli esempi asincroni dimostrano come ridurre la quantità di tempo che un'applicazione MQTT blocca mentre è in attesa del client MQTT . È importante eliminare le chiamate di blocco nel thread principale per aumentare la reattività e la durata della batteria in ambiente mobile.

Gli esempi illustrano due pattern per la chiamata di interfacce asincrona nel client MQTT .

1. `SampleAsyncWait` non si blocca mentre MQTT è in attesa di interazioni di rete.
2. `SampleAsyncCallBack` non blocca l'attesa che il client MQTT completi le azioni. Quest' ultima è necessaria quando una pagina JavaScript richiama il client MQTT da un browser. Le pagine JavaScript non devono essere bloccate. Le risposte alle azioni devono essere inviate nuovamente al thread del browser principale, che richiama il gestore eventi MQTT scritto per elaborare la notifica.

Compila ed esegui tutte le applicazioni MQTT di esempio client Java da Eclipse

Compila ed esegui le applicazioni MQTT client di esempio Java che si trovano in Mobile Messaging and M2M Pacchetto client. Mostrano un publisher e un sottoscrittore MQTT .

Informazioni su questa attività

Compilare ed eseguire gli esempi MQTT Java , Samplee `MQTTV3Sample` in Eclipse. `Sample` si trova nella sottodirectory dei client *`sdkroot`*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` SDK e `MQTTV3Sample.java` si trova in *`sdkroot`*`\SDK\clients\java\samples`.

Procedura

1. Scaricare Eclipse IDE for Java Developers.
2. Crea un progetto Java denominato `MQTT Samples` in Eclipse.
   a) **File** > **Nuovo** > **progetto Java** e tipo `MQTT Samples`. Fare clic su **Avanti**.

      Verificare che il JRE sia alla versione corretta o successiva.JSE deve essere alla versione 1.5 o successiva.

   b) Nella finestra **Impostazioni Java** , fare clic su **Collega ulteriori cartelle di origine**.

   c) Individuare la directory in cui è stata installata la cartella MQTT Java SDK. Selezionare la cartella *`sdkroot`*`\SDK\clients\java\samples` e fare clic su **OK** > **Avanti** > **Fine**.

   d) Nella finestra **Impostazioni Java** , fare clic su **Librerie** > **Aggiungi jar esterni**

   e) Individuare la directory in cui è stata installata la cartella MQTT Java SDK. Individuare la cartella *`sdkroot`*`\SDK\clients\java` e selezionare i file `org.eclipse.paho.client.mqttv3.jar` e `com.ibm.micro.client.mqttv3.jar` ; fare clic su **Apri** > **Fine**.

      I link di esempio `MQTTV3Sample.java` a `com.ibm.micro.client.mqttv3.jare` gli esempi nella struttura di directory paho si collegano a `org.eclipse.paho.client.mqttv3.jar`. Il `com.ibm.micro.client.mqttv3.jar` viene conservato in modo che le applicazioni MQTT esistenti continuino a essere create ed eseguite senza modifiche.

   Il progetto `MQTT Samples` viene creato con alcune avvertenze, ma senza errori.

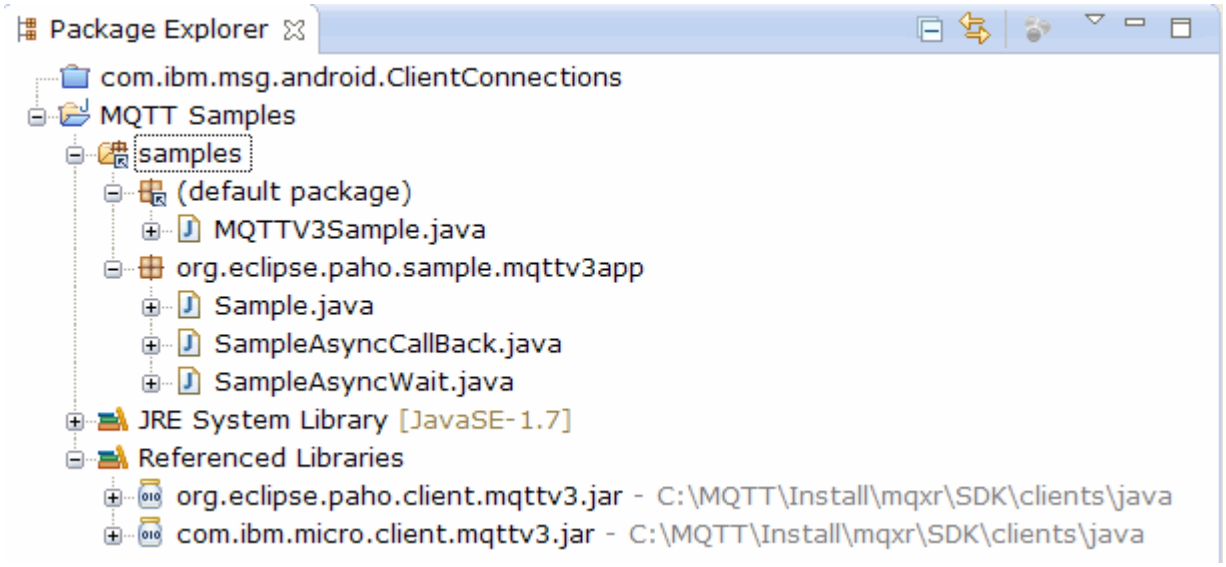


Figura 8. Client MQTT per progetto Java

3. Opzionale: Installare il client MQTT Javadoc.

   Con il Javadoc client MQTT installato, l'editor Java descrive le classi MQTT nella guida a comparsa.

a) Aprire **Esplora package** > **Librerie di riferimento** nel progetto Java. Fare clic con il pulsante destro del mouse `org.eclipse.paho.client.mqttv3.jar` **> Proprietà**.

b) Nel navigator Proprietà, fare clic su **Ubicazione Javadoc**.

c) Fare clic su **URL Javadoc** > **Sfoglia** nella pagina **Percorso Javadoc** e trovare la cartella `SDK\clients\java\doc\javadoc` **> OK**.

d) Fare clic su **Convalida** > **OK**

Viene richiesto di aprire un browser per visualizzare la documentazione.

e) Ripetere questa procedura per il file `com.ibm.micro.client.mqttv3.jar` .

4. Creare una configurazione di runtime del publisher e del sottoscrittore per eseguire l'applicazione `mqttv3app.Sample` .

a) Fare clic con il tasto destro del mouse sulla classe **Esempio** , fare clic su **Esegui come configurazione** > **Esegui**.

b) Fare clic con il pulsante destro del mouse su **Applicazione Java** > **Nuovo** e immettere il nome `SampleSubscriber`.

c) Fare clic sulla scheda degli argomenti e immettere gli argomenti del programma seguiti da **Applica**.

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

d) Ripetere l'ultimo passo per creare una configurazione `SamplePublisher` omettendo il parametro `-a subscribe` .

5. Eseguire il sottoscrittore `mqttv3app.Sample` seguito dal publisher.

a) Fare clic su **Esegui** > **Esegui configurazioni**

b) Fare clic su **SampleSubscriber** > **Esegui**.

Aprire la vista **Console** . Il sottoscrittore è in attesa di una pubblicazione.

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

c) Fare clic sul pulsante **SamplePublisher** > **Esegui**.

Aprire la vista **Console** . Viene visualizzata la pubblicazione creata dal publisher:

```
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) Passa le visualizzazioni della console alla console dell'utente.

L'icona delle console switch è  .

Il sottoscrittore ha ricevuto la pubblicazione:

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

6. Opzionale: Creare una configurazione di runtime del publisher e del sottoscrittore per `MQTTV3Sample`.

a) Fare clic con il pulsante destro del mouse sulla classe **MQTTV3Sample** , selezionare **Esegui come** > **Esegui configurazioni**.

b) Fare clic con il pulsante destro del mouse su **Applicazione Java** > **Nuovo** e immettere il nome `MQTTV3SampleSubscriber`.

c) Fare clic sulla scheda degli argomenti e immettere gli argomenti del programma seguiti da **Applica**.

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

d) Ripetere l'ultimo passo per creare una configurazione `MQTTV3SamplePublisher` omettendo il parametro `-a subscribe` .

7. Opzionale: Eseguire il sottoscrittore `MQTTV3Sample` seguito dal publisher.
   a) Fare clic su **Esegui** > **Esegui configurazioni**
   b) Fare clic su **MQTTV3SampleSubscriber** > **Esegui**.

      Aprire la vista **Console** . Il sottoscrittore è in attesa di una pubblicazione.

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

   c) Fare clic su **MQTTV3SamplePublisher** > **Esegui**.

      Aprire la vista **Console** . È possibile visualizzare la pubblicazione creata dal publisher.

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

   d) Passa le visualizzazioni della console alla console dell'utente.

      L'icona delle console switch è  .

      Il sottoscrittore ha ricevuto la pubblicazione:

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

Introduzione a Client MQTT per Java su Android

È possibile installare un Applicazione di esempio del client MQTT Java per Android che scambia messaggi con un server MQTT . L'applicazione utilizza una libreria client dall'SDK MQTT da IBM. Puoi creare tu stesso l'applicazione o scaricare un'applicazione di esempio precompilata.

Prima di iniziare

- Per le piattaforme client MQTT supportate e di riferimento, consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.
- Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
- L'applicazione di esempio del client MQTT funziona su Ice Cream Sandwich (Android 4.0) e versioni successive. Questa versione di Android offre anche una risoluzione di visualizzazione più nitida sui tablet.

Informazioni su questa attività

Applicazione di esempio del client MQTT Java per Android è denominato "`mqttExerciser`". Questa applicazione utilizza una libreria client dall'SDK MQTT e scambia i messaggi con un server MQTT .

Puoi creare tu stesso l'applicazione di esempio ed esportarla da Eclipse come `mqttExerciser.apk`oppure utilizzare l'applicazione di esempio precompilata disponibile come file `mqttExerciser.apk` nella cartella *`sdkroot`*`\SDK\clients\android\samples\apks` di Mobile Messaging and M2M Pacchetto client. Se scegli di creare tu stesso l'applicazione, l'ambiente di sviluppo che crei è adattato per includere la messaggistica mobile nelle applicazioni for Android. Questo ti dovrebbe aiutare quando inizi a includere la messaggistica mobile nelle tue proprie applicazioni.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

Il server deve supportare il protocollo MQTT version 3.1 . Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.

2. Ottieni gli strumenti giusti.

   Installare un kit di sviluppo Java (JDK) Versione 6 o successiva. Poiché stai sviluppando un'applicazione Java for Android, il JDK deve provenire da Oracle. È possibile ottenere JDK da Download di Java SE

   È necessario anche un ambiente di sviluppo Eclipse . Deve essere Eclipse 3.6.2 (Helios) o superiore. Eclipse deve avere un livello del compilatore Java di almeno 6, per corrispondere al JDK. Puoi ottenere tutto questo da Eclipse Foundation.

   Infine, hai bisogno di Android SDK. Puoi ottenerlo da Get the Android SDK.

3. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.
   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.
   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*`\SDK`.

4. Opzionale: Crea l'applicazione di esempio mqttExerciser for Android.

   Configura gli strumenti Eclipse , Android e importa e crea il progetto `mqttExerciser` dall'SDK MQTT .

   Nota: Se non vuoi farlo ora, puoi utilizzare l'applicazione di esempio preintegrata disponibile come file `mqttExerciser.apk` nella cartella *`sdkroot`*`\SDK\clients\android\samples\apks` dell'SDK MQTT .

   a) Avviare l'ambiente di sviluppo Eclipse con il JRE dal JDK.

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

   b) Selezionare e installare una serie di pacchetti e piattaforme dall'SDK Android .

      Consultare Aggiunta di piattaforme e package per un elenco di piattaforme e package consigliati da Google .

      Nota: La piattaforma SDK deve essere Android API livello 16 o successivo. Con i livelli API precedenti, il progetto non può essere compilato correttamente.

   c) Aggiungere il plug-in Android Development Tools (ADT) a Eclipse.

   d) Importa il progetto dell'applicazione `mqttExerciser` di esempio in Eclipsee correggi gli errori.

      i) Importa il progetto dell'applicazione di esempio dall'SDK MQTT , nel percorso *`sdkroot`*`\SDK\clients\android\samples\mqttExerciser`.

         La visualizzazione **Problemi** elenca molti errori di creazione. Gli errori di build vengono risolti nei passi successivi.

      ii) Copiare la libreria `org.eclipse.paho.client.mqttv3.jar` nella cartella **libs** nel progetto Android . 

Ad esempio, su Windows, si trova nella cartella *`sdkroot`*`\SDK\clients\java` . Viene visualizzata una finestra **Operazione file** . Accettare la selezione **Copia file** e fare clic su **OK**.

      iii) Fare clic con il tasto destro del mouse sulla cartella del progetto, `com.ibm.msg.android`; selezionare **Strumenti Android ...** > **Aggiungi libreria di supporto ...**. Leggere e accettare i termini della licenza, quindi fare clic su **Installa**.

      iv) Fare clic con il tasto destro del mouse sulla cartella del progetto, `com.ibm.msg.android`; selezionare **Strumenti Android ...** > **Proprietà progetto fix**.

v) Se lo spazio di lavoro contiene ancora circa 84 errori, che fanno riferimento alla sovrascrittura di un metodo superclasse, il livello di conformità del compilatore è probabilmente impostato su 1.5 o su un valore inferiore. Android SDK versione 16 prevede che il livello di conformità del compilatore non sia superiore a 1.5. Per correggere gli errori rimanenti, completare la seguente procedura:

a) Controlla e (se necessario) aggiorna il tuo SDK Android e i plugin Eclipse corrispondenti a Android SDK versione 17.

b) Fare clic con il pulsante destro del mouse sulla cartella di progetto **com.ibm.msg.android** , quindi selezionare **Proprietà** > **Compilatore Java**. Controllare il livello di conformità del compilatore, impostarlo su almeno 1.6, quindi ricreare lo spazio di lavoro.

Il progetto viene creato, con alcune avvertenze e senza errori.

5. Installare e avviare Applicazione di esempio client MQTT Java su una periferica Android .

   Vedi la pagina `developer.android.com` Esecuzione della tua applicazione.

   Se hai creato tu stesso l'applicazione come un progetto Eclipse , puoi avviare l'applicazione da Eclipse.

   Se si dispone del file APK (application package) `mqttExerciser.apk`, è possibile installarlo all'esterno di Eclipse utilizzando il comando di installazione ADB (Android Debug Bridge). Questo comando utilizza l'ubicazione del file APK come argomento. Se stai usando l'applicazione di esempio preintegrata, l'ubicazione è *`sdkroot`*`\SDK\clients\android\samples\apks\mqttExerciser.apk`.

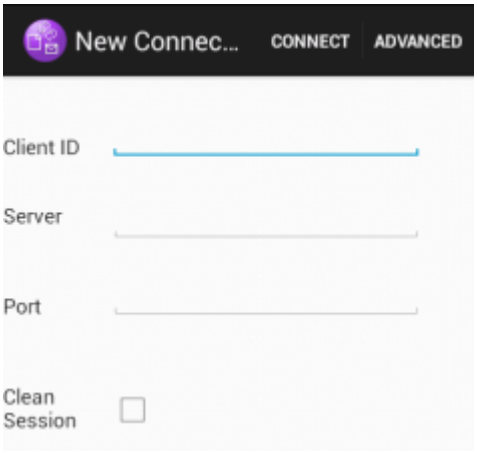
6. Utilizza l'applicazione di esempio mqttExerciser for Android per connettere, sottoscrivere e pubblicare un argomento.

   a) Aprire il Applicazione di esempio del client MQTT Java per Android.

   Questa finestra viene visualizzata nel dispositivo Android :

   b) Collegarsi a un MQTT server.

   i) Fare clic su **+** per aprire una nuova connessione MQTT .



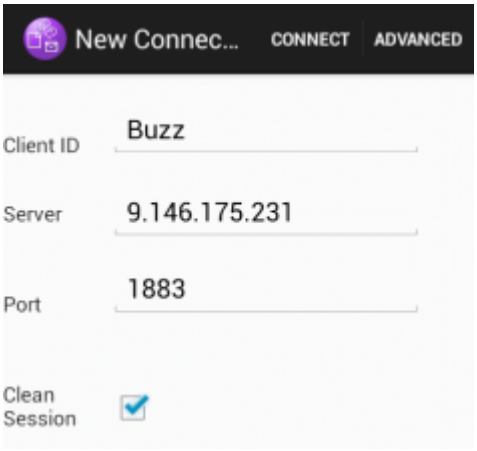
   ii) Immettere un qualsiasi identificativo univoco nel campo **ID client** . Siate pazienti, le sequenze di tasti possono essere lente.

   iii) Immettere nel campo **Server** l'indirizzo IP del server MQTT .

   Questo è il server scelto nel primo passo principale. L'indirizzo IP non deve essere `127.0.0.1`

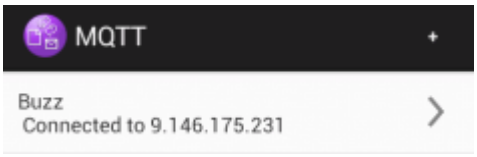
   iv) Immettere il numero di porta della connessione MQTT .

   Il numero di porta predefinito per una connessione MQTT normale è 1883.



v) Fare clic su **Connetti**.

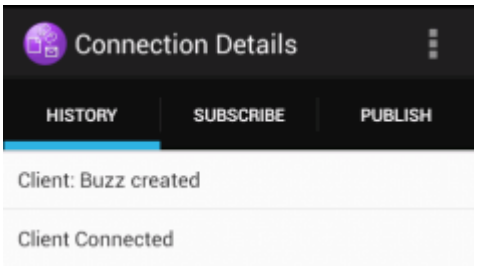
Se la connessione ha esito positivo, viene visualizzato un messaggio "Connessione in corso" seguito da questa finestra:



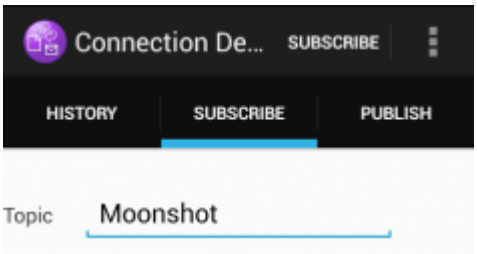
c) Sottoscrivere un argomento.

i) Fare clic su **Connesso** .

Viene visualizzata la finestra **Dettagli connessione** con la cronologia elencata:



ii) Selezionare la scheda **Sottoscrivi** e immettere una stringa di argomento.

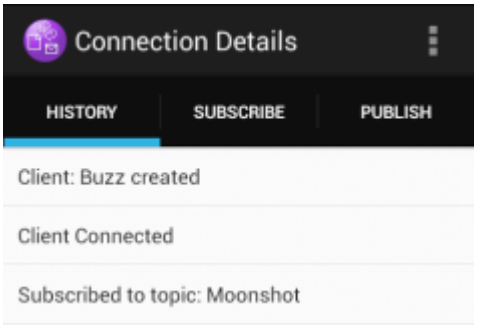


iii) Fare clic su **Sottoscrivi** .

Viene visualizzato un messaggio "Sottoscritto" per un breve periodo di tempo.

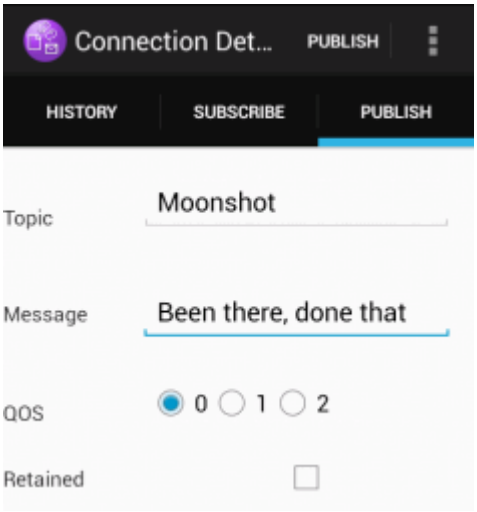
iv) Fare clic sulla scheda **Cronologia** .

La cronologia ora include la sottoscrizione:



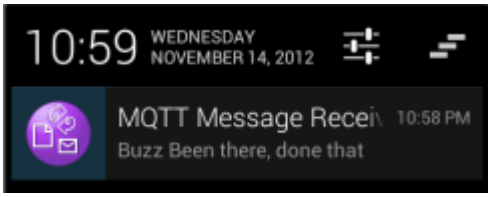
d) Ora pubblica sullo stesso argomento.

i) Fare clic sulla scheda **Pubblica** e immettere la stessa stringa di argomenti della sottoscrizione. Immettere un messaggio.

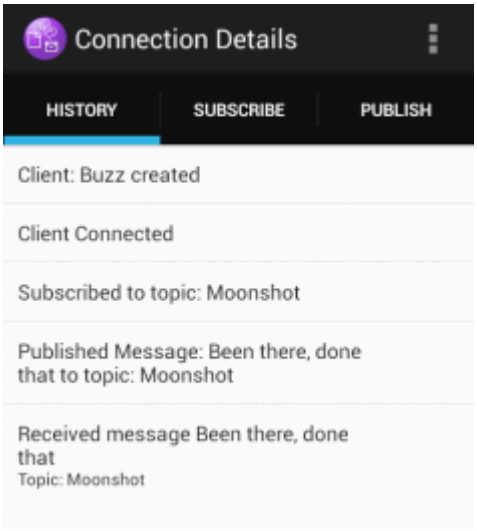


ii) Fare clic su **Pubblica** .

Vengono visualizzati due messaggi per un breve periodo di tempo, "Pubblicato" seguito da "Sottoscritto". La pubblicazione viene visualizzata nell'area di stato (tirare la barra di separazione verso il basso per aprire la finestra di stato).



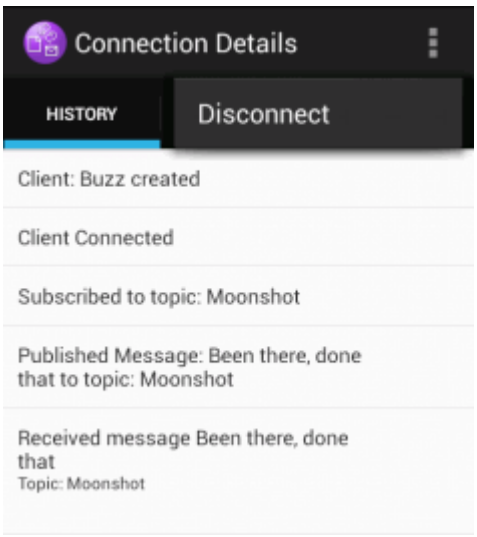
iii) Fare clic sulla scheda **Cronologia** per visualizzare la cronologia completa.



e) Disconnettere l'istanza client.

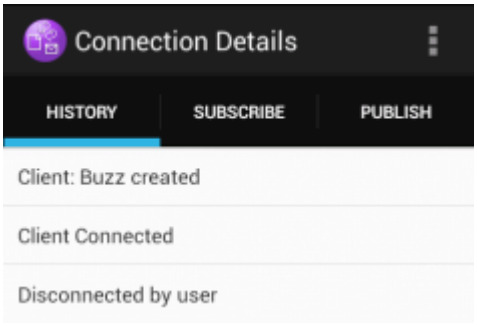
i) Fare clic sull'icona menu nella barra delle azioni. 

Applicazione di esempio del client MQTT Java per Android aggiunge un pulsante **Disconnetti** alla finestra MQTT **Dettagli connessione** .

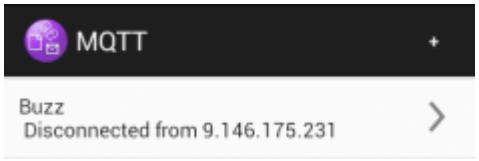


ii) Fare clic su **Disconnetti**.

Lo stato connesso cambia in disconnesso:



f) Fare clic su **Indietro** per tornare all'elenco di sessioni Applicazione di esempio client MQTT Java .



- Fare clic sul segno più per avviare una nuova sessione Applicazione di esempio client MQTT Java .
- Fare clic sul client disconnesso per riconnetterlo.
- Fare clic su **Indietro** per ritornare al launchpad.

g) Fare clic sul pulsante Attività per elencare le app in esecuzione. Individuare Applicazione di esempio client MQTT Java. Scorri l'icona fuori dallo schermo per chiuderla.

Operazioni successive

Se hai creato tu stesso l'applicazione di esempio, sei pronto a iniziare a sviluppare le tue Android applicazioni che richiamano le librerie MQTT per lo scambio di messaggi. Puoi modellare le tue applicazioni Android sulle classi in `mqttExerciser`. Per studiare l'esempio, generare Javadoc per le classi in `com.ibm.msg.android` e `com.ibm.msg.android.service` nel progetto `mqttExerciser` .

Informazioni correlate

Gestione di Progetti da Eclipse con ADT

Introduzione a MQTT messaging client per JavaScript

È possibile iniziare a utilizzare MQTT messaging client per JavaScript visualizzando la home page di esempio del client di messaggistica e sfogliando le risorse a cui si collega. Per visualizzare questa home page, configurare un server MQTT per accettare le connessioni da Esempio di client di messaggistica MQTT JavaScript pagine, quindi immettere l'URL configurato sul server in un browser Web. MQTT messaging client per JavaScript viene avviato automaticamente sul dispositivo e viene visualizzata la home page di esempio del client di messaggistica. Questa pagina contiene collegamenti ai programmi di utilità, alla documentazione dell'interfaccia di programmazione, a un'esercitazione e ad altre informazioni utili.

Prima di iniziare

Per un utilizzo avanzato o in produzione, si desidera rimodellare o rimuovere la home page di esempio del client di messaggistica. Si noti che le interfacce utente risultanti dal codice di esempio non sono garantite per essere conformi agli standard di accessibilità o ai requisiti di accessibilità.

È necessario un server MQTT per supportare MQTT messaging client per JavaScript. Questo server deve supportare il protocollo MQTT V3.1 su WebSockets. IBM MessageSight, e IBM WebSphere MQ Version 7.5.0, Fix Pack 1 e versioni successive, supportano MQTT protocol su WebSockets. Consultare “Introduzione ai server MQTT” a pagina 136. Per installare IBM WebSphere MQ per una valutazione gratuita di 90 giorni, consultare “InstallazioneIBM WebSphere MQ” a pagina 138.

Il WebSocket protocol è stato recentemente stabilito. Se c'è un firewall tra il client e il server, controllare che non blocchi il traffico WebSockets. Allo stesso modo, se il browser non supporta ancora WebSocket protocol[1]non sarà possibile utilizzare il programma di utilità del client o le esercitazioni disponibili dalla home page di esempio del client di messaggistica. La tabella Tabella 1 a pagina 25 elenca i browser le cui versioni più recenti sono state verificate e mostrate per funzionare con il client di messaggistica.

[1] In particolare, se non supporta lo standard RFC 6455 (WebSocket).

Tabella 1. Browser supportati per l'utilizzo con MQTT messaging client per JavaScript			
Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 e versioni successive Chrome for Android 25.0 e versioni successive	Safari 6.0 e versioni successive Chrome 14.0 e successive	Firefox 6.0 e versioni successive Chrome 14.0 e successive	Firefox 6.0 e versioni successive Chrome 14.0 e successive

Informazioni su questa attività

La maggior parte dei passi in questa attività consiste nella configurazione del server MQTT . Tutto ciò che è richiesto per accedere al client di messaggistica per JavaScript è l'esecuzione di un browser che supporta WebSocket protocol.

Su IBM WebSphere MQ, attenersi alla procedura per abilitare IBM WebSphere MQ Telemetry creando canali di esempio. Connettersi al canale MQTT WebSockets predefinito di esempio sulla porta 1883. L'URL della home page del client di messaggistica è `http://`*`hostname`*`:1883` su IBM WebSphere MQ.

Su IBM MessageSight, installare e impostare il dispositivo, configurare l'hub di messaggistica per accettare connessioni e creare un endpoint MQTT WebSockets .

Procedura

1. Scarica Mobile Messaging and M2M Pacchetto cliente scegli un server MQTT a cui puoi connettere l'app client.

   Consultare “Introduzione ai client MQTT” a pagina 11.
2. Configurare il proprio server MQTT per accettare le connessioni dalle pagine HTML di esempio MQTT messaging client per JavaScript .
   - Su IBM WebSphere MQ:
     - Se hai già un gestore code IBM WebSphere MQ configurato per MQTT, modifica il protocollo nella definizione del canale per supportare sia MQTT che HTTP. Vedere **ALTER CHANNEL**.
     - Per creare un gestore code IBM WebSphere MQ e configurare l'endpoint MQTT WebSockets di esempio, completare una delle seguenti attività:
       - “Configurazione del servizio MQTT dalla riga di comando” a pagina 140
       - “Configurazione del servizio MQTT con IBM WebSphere MQ Explorer” a pagina 143
3. Aprire un browser Web sul dispositivo.
4. Immettere l'URL della home page di esempio del client di messaggistica.
   - Su IBM WebSphere MQ, è `http://`*`hostname`*`:1883`
   - Su IBM MessageSight, è `http://`*`hostname`*`:`*`port`*

     dove *`hostname`* è il nome DNS o l'indirizzo IP del socket Ethernet configurato sul dispositivo IBM MessageSight come endpoint a cui deve connettersi il client e *`port`* è il numero di porta TCP/IP assegnato all'endpoint per il client.

   Viene visualizzata la home page dell'esempio client di messaggistica.

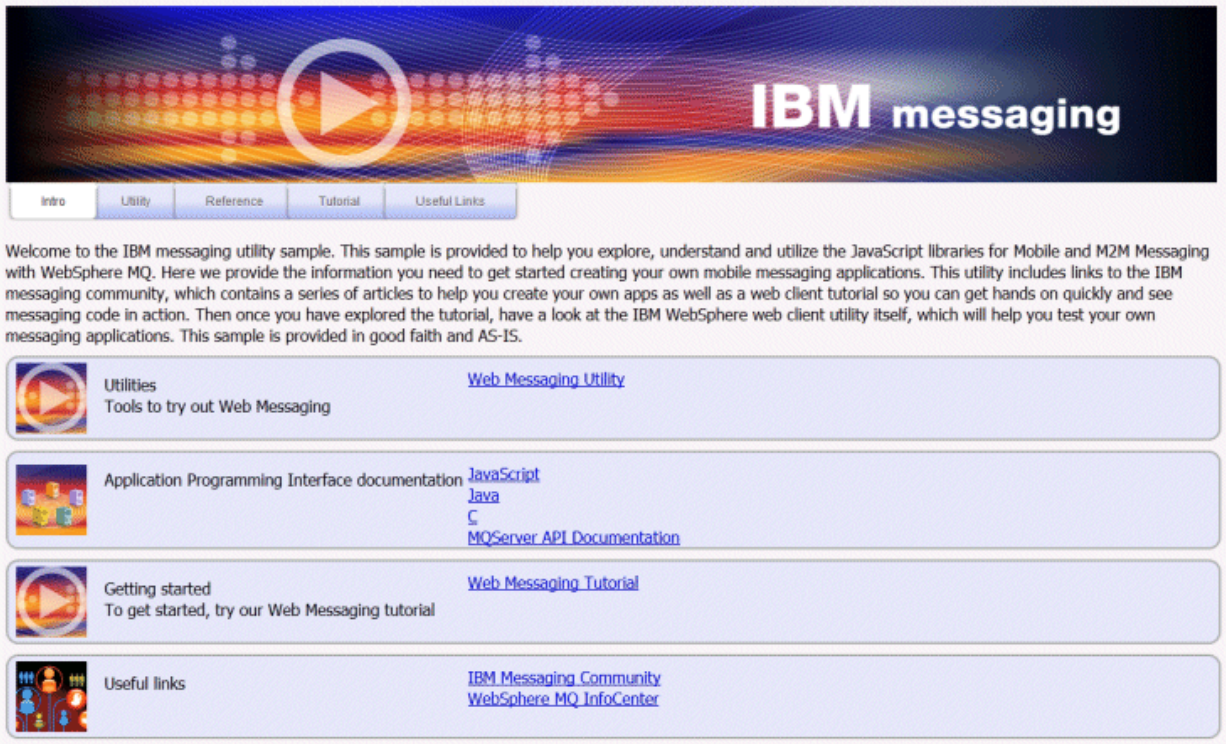


Figura 9. Home page di esempio MQTT messaging client per JavaScript

Risultati

È stata configurata un canale MQTT per WebSockets.

Nella home page di MQTT messaging client per JavaScript, fare clic su **Web Messaging Utility** per provare diverse funzioni nell'API del client di messaggistica. Ad esempio, è possibile connettersi al gestore code, sottoscrivere i messaggi e pubblicare alcuni messaggi. È anche possibile fare clic su **Esercitazione messaggistica web** per informazioni su come creare una pagina web che richiama il client di messaggistica MQTT per l'API JavaScript .

Concetti correlati

“MQTT messaging client per JavaScript e le applicazioni web” a pagina 116
“Come programmare le app di messaggistica in JavaScript” a pagina 120

Attività correlate

“Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77
Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

Introduzione al client MQTT per C

Introduzione e esecuzione con il client MQTT di esempio per C su qualsiasi piattaforma su cui è possibile compilare l'origine C. Verificare che sia possibile eseguire il client MQTT di esempio per C con IBM MessageSight o IBM WebSphere MQ come server MQTT.

Prima di iniziare

- Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
- Per il client MQTT supportato e di riferimento per le piattaforme C. Consultare: Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.

Informazioni su questa attività

Attenersi a questa attività per compilare ed eseguire il client MQTT di esempio per C su Windows dalla riga comandi o da Microsoft Visual Studio 2010. Microsoft Visual Studio 2010 viene utilizzato anche per compilare il client nell'esempio della riga comandi. Modificare gli script della linea di comando per compilare ed eseguire l'esempio su altre piattaforme.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 . Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.
2. Installare un ambiente di sviluppo C sulla piattaforma in cui si sta eseguendo la creazione.

   I makefile negli esempi in questo argomento sono destinati ai seguenti strumenti:

   - 

Per iOS, su Apple Mac con OS X 10.8.2 con gli strumenti di sviluppo iOS da Xcode.
   - 

Per Linux, gcc versione 4.4.6 da Red Hat® Enterprise Linux versione 6.2.

     Il livello minimo supportato della libreria C `glibc` è 2.12e del kernel Linux è 2.6.32.
   - 

Per Microsoft Windows, Visual Studio versione 10.0.
3. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.
   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.
   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*`\SDK`.
4. Opzionale: Seguire la procedura riportata in “Creazione del client MQTT per le librerie C” a pagina 31.

   Eseguire questa procedura solo se Mobile Messaging and M2M Pacchetto client non include la libreria client C per la piattaforma di destinazione.
5. Compila ed esegui l'applicazione C di esempio client MQTT , `MQTTV3Sample.c`.

   - Dalla riga di comando, segui la procedura in “Compila ed esegui l'applicazione C di esempio del client MQTT dalla riga di comando” a pagina 27.
   - Da un IDE, attieniti alla procedura in “Compilare ed eseguire l'applicazione C di esempio del client MQTT da Microsoft Visual Studio” a pagina 28.

Compila ed esegui l'applicazione C di esempio del client MQTT dalla riga di comando

Compila ed esegui l'applicazione C di esempio client MQTT dalla riga di comando. L'esempio è in MQTT SDK. Illustra un publisher e un sottoscrittore MQTT .

Prima di iniziare

Installare un ambiente di sviluppo C, ad esempio Microsoft Visual Studio 2010 come utilizzato nell'esempio.

Informazioni su questa attività

Compilare ed eseguire l'esempio C, `MQTTV3Sample`, nella sottodirectory dei client SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedura

Creare uno script nella directory degli esempi client per compilare ed eseguire `Sample` sulla piattaforma scelta.

Il seguente script compila ed esegue l'esempio su una piattaforma Windows a 32 bit, creata con Microsoft Visual Studio 2010. Eseguire lo script dalla sottodirectory *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
```

Risultati

Il publisher e il sottoscrittore scrivono l'output nelle relative finestre dei comandi:

```
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. Output del publisher

```
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. Output dal sottoscrittore

Compilare ed eseguire l'applicazione C di esempio del client MQTT da Microsoft Visual Studio

Compila ed esegui l'applicazione C di esempio client MQTT da Microsoft Visual Studio. L'esempio si trova in Mobile Messaging and M2M Pacchetto client. Illustra un publisher e un sottoscrittore MQTT .

Prima di iniziare

L'esempio utilizza Microsoft Visual Studio 2010. È possibile utilizzare altri ambienti di sviluppo C su Windows e altre piattaforme; ad esempio Eclipse IDE for C/C++ Developers.

Informazioni su questa attività

Compilare ed eseguire l'esempio C, `MQTTV3Sample` con Microsoft Visual Studio 2010. `MQTTV3Sample.c` si trova nella sottodirectory dei client SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedura

1. Avviare Microsoft Visual Studio.
2. Creare un nuovo progetto dal codice esistente.
   a) Fare clic su **File** > **Nuovo** > **Progetto da codice esistente**.
   b) Selezionare **Visual C++** come tipo di progetto da creare.
   c) Fare clic su **Avanti**.
3. Specificare i parametri nella finestra **Percorso del progetto e file di origine** .
   a) Fare clic su **Sfoglia** e individuare la directory *sdkroot*\SDK\clients\c\samples .
   b) Denominare il progetto MQTTV3Sample.
   c) Fare clic su **Avanti**.
4. Selezionare **Progetto applicazione console** nell'elenco **Tipo di progetto** . Fare clic su **Fine**
5. Configurare solo la configurazione di debug.

   Per impostazione predefinita, Microsoft Visual Studio crea una configurazione di release e di debug. Nell'esercitazione, si configura la configurazione di debug. Per eliminare gli errori di creazione, deselezionare l'opzione **Crea** per la configurazione della release.

   a) Fare clic su **Progetto** > **MQTTV3Sample Properties** > **Configuration Manager**. Selezionare **Release** come **Configurazione della soluzione attiva** e deselezionare **Build**.
   b) Selezionare **Debug** come **Configurazione della soluzione attiva** > **Chiudi**.

      Verificare di aver modificato la configurazione di debug in tutti i seguenti passi.
6. Modificare le impostazioni **C/C + +** in **Pagine delle proprietà MQTTV3Sample**.
   a) Nella finestra **MQTTV3Sample Property Pages** , aprire **Proprietà di configurazione** > **C/C++** > **Generale**.
   b) Nell'elenco delle proprietà generali, fare clic su **Directory di inclusione aggiuntive**, aggiungere il percorso di directory a *sdkroot*\SDK\clients\c\include e fare clic su **Applica**.
7. Modifica delle impostazioni **Linker**
   a) Aprire **Proprietà di configurazione** > **Linker** > **Generale**.
   b) Nell'elenco delle proprietà generali, fare clic su **Directory della libreria aggiuntive**e aggiungere il percorso della directory a *sdkroot*\SDK\clients\c\windows_ia32
   c) Nell'elenco delle proprietà di Linker, fare clic su **Riga comandi**. Immettere mqttv3c.lib nell'area di immissione dati **Opzioni aggiuntive** e fare clic su **Applica**.
8. Rimuovere il file di origine MQTTV3SSample.c dal progetto.
   a) Aprire la cartella **MQTTV3sample** > **File di origine** nella finestra **Esplora soluzioni** .
   b) Fare clic con il tasto destro del mouse su **MQTTV3SSample.c** > **Escludi dal progetto**
9. Creare il progetto MQTTV3Sample .
   a) Fare clic con il pulsante destro del mouse su **progettoMQTTV3sample** in Esplora soluzioni e fare clic su **Crea**.

      La build viene completata senza errori.
10. Aggiungere due nuovi progetti per eseguire **MQTTV3Sample** come istanza di runtime di Subscriber e Publisher.

    I progetti Publisher e Subscriber devono contenere i comandi per eseguire **MQTTV3Sample**. Non sono creati e non contengono codice.

    a) In **Esplora soluzioni**, fare clic con il pulsante destro del mouse su **Soluzione ` MQTTV3Sample `** > **Aggiungi** > **Nuovo progetto**.
    b) Immettere Subscriber nel campo **Nome** . Lasciare selezionato **Win32 Console Application** . Fare clic su **OK**.

       Viene avviata la **procedura guidata dell'applicazioneWin32** .

c) Nella **Procedura guidata dell'applicazioneWin32**, fare clic su **Avanti**. Selezionare **Progetto vuoto** > **Fine**

d) Ripetere questi passi per aggiungere un progetto Publisher.

e) Fare clic con il pulsante destro del mouse sul progetto Sottoscrittore e selezionare **Imposta come progetto StartUp**

La finestra **Esplora soluzioni** viene visualizzata in Figura 12 a pagina 30.

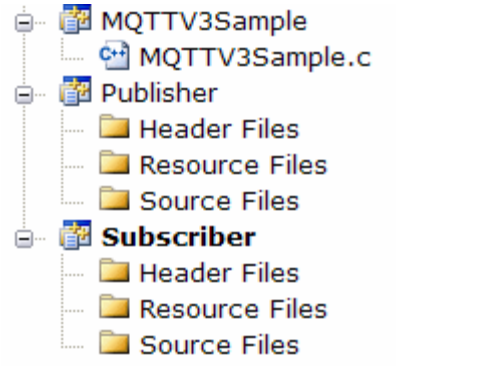


Figura 12. Soluzione MQTTV3Sample

11. Configurare le pagine delle proprietà del sottoscrittore.

a) Fare clic con il tasto destro del mouse su **Sottoscrittore** in Esplora soluzioni **Proprietà** > **Proprietà di configurazione** > **Proprietà** > **Debug**

Verificare che il titolo della finestra sia **Pagine proprietà sottoscrittore**.

b) Fare clic su **Ambiente**. Immettere `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` e fare clic su **Applica**.

Modifica *`sdkroot`* per adattarlo al tuo ambiente.

c) Fare clic su **Comando**e sostituire `$(TargetPath)` con il percorso del modulo `MQTTV3Sample`

Ad esempio, *`sdkroot`*`\SDK\clients\c\samples\debug\MQTTV3Sample`

Modifica *`sdkroot`* per adattarlo al tuo ambiente.

d) Fare clic su **Argomenti del comando** e immettere `-a subscribe -b localhost -p 1883` e fare clic su **Applica**.

12. Configurare le pagine delle proprietà Publisher.

Suggerimento: È possibile cambiare il progetto delle pagine delle proprietà facendo clic sui progetti nella finestra **Esplora soluzioni** .

a) Ripetere i passi del sottoscrittore per il publisher.

L'argomento del comando è `-b localhost -p 1883`

13. Arrestare il processo di build che crea i progetti Publisher e Subscriber.

a) Nelle pagine delle proprietà di uno qualsiasi dei progetti, fare clic su **Gestore configurazione** e deselezionare **Crea** per Publisher e Sottoscrittore in entrambe le configurazioni Release e Debug. Fai clic su **Chiudi**.

14. Eseguire l'esempio.

a) Fare clic su **F5** per avviare il sottoscrittore

b) Fare clic con il tasto destro del mouse su **Publisher** in Esplora soluzioni, **Debug** > **Avvio di una nuova istanza**

Risultati

L'output del publisher e del sottoscrittore nelle finestre dei comandi. Visual Studio chiude la finestra del publisher. Esaminare la finestra dell'utente, mostrata nella seguente figura, quindi chiudere la finestra dell'utente.

```
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. Output dal sottoscrittore

Operazioni successive

Creare ed eseguire il publisher e il sottoscrittore asincroni. Gli esempi sono `MQTTV3ASample.c` e `MQTTV3ASSample.c` in *sdkroot*`\SDK\clients\c\samples`.

Creazione del client MQTT per le librerie C

Attenersi alla seguente procedura per creare il client MQTT per le librerie C. L'argomento include gli switch di compilazione e di collegamento per un numero di piattaforme ed esempi di creazione delle librerie su iOS e Windows.

Prima di iniziare

1. Creare la libreria client C solo quando necessario. Collegare le librerie client precompilate nell'SDK (Software Development Kit) nella sottodirectory `SDK\clients\c` se una corrisponde alla piattaforma di destinazione.
2. Configurare un server MQTT per verificare la libreria creata con Applicazione C di esempio client MQTT. Consultare “Introduzione ai server MQTT” a pagina 136. Verifica la configurazione del server eseguendo una delle app di esempio del client MQTT .
3. Se si sta creando una versione sicura della libreria C, che supporta (SSL (Secure Sockets Layer)) SSL, è necessario creare anche una libreria OpenSSL . Vedere “Creazione del pacchetto di OpenSSL” a pagina 45.

   Importante: Il download e la ridistribuzione del package OpenSSL sono soggetti a rigide normative di importazione ed esportazione e a condizioni di licenza open source. Prestare particolare attenzione alle limitazioni e alle avvertenze prima di decidere se scaricare il pacchetto.

Informazioni su questa attività

Seguire le istruzioni in “Creazione del pacchetto di OpenSSL” a pagina 45 per scaricare e creare la libreria OpenSSL . È necessario creare OpenSSL per creare una versione sicura del client MQTT per la libreria C. Non è necessario OpenSSL per creare una versione non protetta della libreria MQTT . La procedura include esempi di creazione della libreria per iOS e Windows.

Creare la libreria MQTT client for C scaricando gli strumenti della libreria di sviluppo C e l'SDK (software development toolkit) MQTT sulla piattaforma di build. Scrivere un makefile per creare la libreria per la piattaforma di destinazione, incorporando le opzioni documentate in “MQTT opzioni di build per piattaforme differenti” a pagina 32. I passaggi specifici della piattaforma per creare ed eseguire un makefile sono riportati di seguito:

- iOS “Creazione di librerie client MQTT per C su un Apple Mac da utilizzare con periferiche iOS” a pagina 34
- Windows “Creazione delle librerie MQTT su Windows” a pagina 40

Procedura

1. Installare un ambiente di sviluppo C sulla piattaforma in cui si sta eseguendo la creazione.

   I makefile negli esempi in questo argomento sono destinati ai seguenti strumenti:

   - 

Per iOS, su Apple Mac con OS X 10.8.2 con gli strumenti di sviluppo iOS da Xcode.
   - 

Per Linux, gcc versione 4.4.6 da Red Hat Enterprise Linux versione 6.2.

     Il livello minimo supportato della libreria C `glibc` è 2.12e del kernel Linux è 2.6.32.
   - 

Per Microsoft Windows, Visual Studio versione 10.0.
2. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.
   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.
   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*`\SDK`.
3. Espandere il codice di origine per il client MQTT per le librerie C.

   Il file compresso del codice sorgente è *`sdkroot`*`\SDK\clients\c\source.zip`.
4. Opzionale: Creare OpenSSL.

   Vedere “Creazione del pacchetto di OpenSSL” a pagina 45.
5. Creare un client MQTT per le librerie C.

   I comandi e le opzioni per creare le librerie sono riportati in “MQTT opzioni di build per piattaforme differenti” a pagina 32.

   Seguire i passi nei seguenti esempi per scrivere un makefile per creare il client MQTT per le librerie C per la piattaforma di destinazione.

   - “Creazione di librerie client MQTT per C su un Apple Mac da utilizzare con periferiche iOS” a pagina 34
   - “Creazione delle librerie MQTT su Windows” a pagina 40

MQTT opzioni di build per piattaforme differenti

La seguente tabella elenca le opzioni di compilazione e creazione per creare il client MQTT per le librerie C su varie piattaforme.

Tabella 2. MQTT opzioni di build per piattaforme differenti

Piattaforma	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 a 32 bit	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			

Tabella 2. MQTT opzioni di build per piattaforme differenti (Continua)				
Piattaforma	**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 *Creazione di librerie client MQTT per C su un Apple Mac da utilizzare con periferiche iOS*

Attenersi alla seguente procedura per scrivere un makefile per creare le librerie client MQTT per C su Apple Mac, per un utilizzo successivo con periferiche iOS .

Prima di iniziare

1. Installare gli strumenti di build, sviluppare ed eseguire il makefile su Apple Mac con OS X 10.8.2o versioni successive.
2. Installare gli strumenti della riga comandi per Xcode, che includono il programma **make** . Scarica gli strumenti della riga di comando da Xcode.

Informazioni su questa attività

Creare un makefile che crei librerie client MQTT per C per iPhone o iPad che esegue un processore ARMv7 o ARMv7s e il simulatore iPhone che viene eseguito su un processore i386-64 bit. Vedi Elenco di dispositivi iOS.

Suggerimento: “Elenco makefile MQTTios.mak” a pagina 38 elenca il makefile completo.

1. Copiare e incollare l'elenco in un file.
2. Convertire il carattere iniziale di ogni riga che segue una destinazione in una scheda; consultare il passo “8” a pagina 36.
3. Eseguire il comando elencato nel passo “9” a pagina 38 della procedura.

Procedura

1. Scarica e installa gli strumenti di sviluppo iOS .
   a. Collegarsi con un ID utente con privilegi di gestione.
   b. Verificare che Apple Mac sia alla versione 10.8.2 o successiva.

c. Vai al sito web Xcode per scaricare Xcode dall'app store Mac .

d. Installare Xcode, l'ambiente della riga comandi e il simulatore.

Se l'app store Mac offre più versioni del simulatore, scegli la versione compatibile con il livello di iOS di destinazione per la tua applicazione.

2. Creazione del Makefile `MQTTios.mak`

Aggiungere un prologo:

```
# L'output di build viene prodotto nella directory corrente.
# MQTTCLIENT_DIR deve puntare alla directory di base contenente il codice sorgente client
MQTT.
# MQTTCLIENT_DIR predefinito è la directory corrente
# Il valore predefinito di TOOL_DIR è /Applications/Xcode.app/Contents/Developer/Platforms
# Il valore predefinito di OPENSSL_DIR è sdkroot/openssl, relativo a sdkroot/sdk/clients/c/
mqttv3c/src
# OPENSSL_DIR deve puntare alla directory di base contenente la creazione OpenSSL .
# Esempio: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. Impostare l'ubicazione del codice sorgente MQTT .

Eseguire il makefile nella stessa directory dei file di origine MQTT oppure impostare il parametro della riga comandi `MQTTCLIENT_DIR` :

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

Aggiungere le seguenti righe al makefile:

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

L'esempio imposta VPATH sulla directory in cui **make** ricerca i file di origine non esplicitamente identificati, ad esempio tutti i file di intestazione richiesti nella build.

4. Opzionale: Impostare l'ubicazione delle librerie OpenSSL

Questo passo è necessario per creare le versioni SSL del MQTT client per le librerie C.

Impostare il percorso predefinito per le librerie OpenSSL sulla stessa directory in cui è stato espanso l'SDK MQTT . Altrimenti, impostare `OPENSSL_DIR` come un parametro di riga comandi.

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

Suggerimento: *OpenSSL* è la directory OpenSSL che contiene tutte le sottodirectory OpenSSL . È possibile che sia necessario spostare la struttura ad albero di directory da dove ne è stata eseguita l'espansione perché contiene directory parent vuote non necessarie.

5. Impostare le directory degli strumenti di sviluppo.

Se Xcode è stato installato in un'ubicazione differente, impostare `TOOL_DIR` nella riga comandi.

```
ifndef DIR_PRECEDENTE
   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. Selezionare tutti i file di origine richiesti per creare ciascuna libreria MQTT . Inoltre, impostare il nome e l'ubicazione della libreria MQTT da compilare.

Aggiungere la seguente riga al makefile per elencare tutti i file di origine MQTT :

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

I file di origine dipendono dal fatto che si stia compilando una libreria sincrona o asincrona e dal fatto che la libreria includa SSL o meno.

Aggiungere una o più di queste righe, che dipendono dalle destinazioni da creare. Le librerie condivise vengono create nella directory `darwin_x86_64` .

- Sincrono, non protetto:

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

- Sincrona, protetta:

```
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
```

- Asincrono, non protetto:

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

- Asincrono protetto:

```
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. Definire il compilatore e le opzioni del compilatore.

   Consultare le opzioni per le diverse piattaforme riportate in Opzioni di creazione MQTT per le diverse piattaforme.

   a) Impostare il progetto Gnu C e C++ (**gcc**) come compilatore.

   Selezionare tre compilatori incrociati per creare la libreria per dispositivi differenti e il simulatore 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) Aggiungere le opzioni del compilatore.

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

   c) Aggiungere i percorsi di inclusione.

```
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. Definire le destinazioni di build.

   Suggerimento: Ogni riga successiva che definisce l'implementazione di una destinazione deve iniziare con un carattere di tabulazione.

   a) Definire la destinazione **all** .

   La destinazione "all" crea tutte le librerie.

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

   Elencandolo per primo, è la destinazione predefinita.

a) Creare la libreria sincrona non protetta, `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 $@
```

L'istruzione `rm *.o` elimina tutti i file oggetto creati per ciascuna libreria. `lipo` concatena tutte e tre le librerie in un file.

b) Creare la libreria asincrona non protetta, `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 $@
```

L'istruzione `rm *.o` elimina tutti i file oggetto creati per ciascuna libreria. `lipo` concatena tutte e tre le librerie in un file.

c) Creare la libreria protetta e sincrona, `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 $@
```

L'istruzione `rm *.o` elimina tutti i file oggetto creati per ciascuna libreria. `lipo` concatena tutte e tre le librerie in un file.

d) Creare la libreria protetta asincrona, `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 $@
```

L'istruzione `rm *.o` elimina tutti i file oggetto creati per ciascuna libreria. `lipo` concatena tutte e tre le librerie in un file.

e) Definire la destinazione **`clean`** .

La destinazione "clean" rimuove tutti i file e le directory generati dal makefile

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

9. Eseguire il makefile.

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

Risultati

I seguenti file vengono creati nella directory *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
```

Elenco makefile `MQTTios.mak`

```
# L'output di build viene prodotto nella directory corrente.
# MQTTCLIENT_DIR deve puntare alla directory di base contenente il codice sorgente client MQTT.
# MQTTCLIENT_DIR predefinito è la directory corrente
# Il valore predefinito di TOOL_DIR è /Applications/Xcode.app/Contents/Developer/Platforms
# Il valore predefinito di OPENSSL_DIR è sdkroot/openssl, relativo a sdkroot/sdk/clients/c/
mqttv3c/src
# OPENSSL_DIR deve puntare alla directory di base contenente la creazione OpenSSL .
# Esempio: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

MQTTCLIENT_DIR ifndef
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
OPENSSL_DIR ifndef
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/ ../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
ifndef DIR_PRECEDENTE
   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
FILE_DI_ORIGINE = ${filtro - 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 = ${filtro - 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: ripulitura
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *Creazione delle librerie MQTT su Windows*

Attenersi alla seguente procedura per scrivere un makefile per creare le librerie client MQTT per C su Windows.

Prima di iniziare

1. Se necessario, installare una versione di **Make** sulla workstation di build compatibile con i makefile scritti per Gnu make; altrimenti, scaricare Gnu make e crearla. Consultare Gnu Make. Il sito Web, Make for Windows, fornisce una versione installabile di **Make** per Windows.
2. È inoltre necessario utilizzare i comandi Linux per Windows per utilizzare la destinazione di ripulitura nell'esempio makefile. È possibile ottenere i comandi Linux per Windows da siti Web come Cygwin.

Informazioni su questa attività

Creare un makefile che crei librerie client MQTT per C per Windows a 32 bit.

Suggerimento: “Elenco makefile MQTTwin.mak” a pagina 44 elenca il makefile completo.

1. Copiare e incollare l'elenco in un file.
2. Convertire il carattere iniziale di ogni riga che segue una destinazione in una scheda; consultare il passo “7” a pagina 42.
3. Eseguire il comando elencato nel passo “9” a pagina 43 della procedura.

Procedura

1. Creazione del Makefile `MQTTwin.mak`

   Aggiungere un prologo:

   ```
   # L'output di build viene prodotto nella directory corrente.
   # MQTTCLIENT_DIR deve puntare alla directory di base contenente il codice sorgente client
   MQTT.
   # MQTTCLIENT_DIR predefinito è la directory corrente
   # Il valore predefinito di OPENSSL_DIR è sdkroot\openSSL, relativo a
   sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR deve puntare alla directory di base contenente la creazione OpenSSL .
   # Esempio: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # Impostare l'ambiente di build, ad esempio:
   # %comspec% /k "" C:\Programmi\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
   # set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
   ```

2. Impostare l'ubicazione del codice sorgente MQTT .

   Eseguire il makefile nella stessa directory dei file di origine MQTT oppure impostare il parametro della riga comandi `MQTTCLIENT_DIR` :

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

   Aggiungere le seguenti righe al makefile:

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

L'esempio imposta VPATH sulla directory in cui **make** ricerca i file di origine non esplicitamente identificati, ad esempio tutti i file di intestazione richiesti nella build.

3. Opzionale: Impostare l'ubicazione delle librerie OpenSSL

   Questo passo è necessario per creare le versioni SSL del MQTT client per le librerie C.

   Impostare il percorso predefinito per le librerie OpenSSL sulla stessa directory in cui è stato espanso l'SDK MQTT . Altrimenti, impostare OPENSSL_DIR come un parametro di riga comandi.

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

   Suggerimento: *OpenSSL* è la directory OpenSSL che contiene tutte le sottodirectory OpenSSL . È possibile che sia necessario spostare la struttura ad albero di directory da dove ne è stata eseguita l'espansione perché contiene directory parent vuote non necessarie.

4. Selezionare tutti i file di origine richiesti per creare ciascuna libreria MQTT . Inoltre, impostare il nome e l'ubicazione della libreria MQTT da compilare.

   Aggiungere la seguente riga al makefile per elencare tutti i file di origine MQTT :

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

   I file di origine dipendono dal fatto che si stia compilando una libreria sincrona o asincrona e dal fatto che la libreria includa SSL o meno.

   Aggiungere una o più di queste righe, che dipendono dalle destinazioni da creare. Le librerie condivise e i manifest vengono creati nella directory `windows_ia32` .

   - Sincrono, non protetto:

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

   - Sincrona, protetta:

     ```
     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
     ```

   - Asincrono, non protetto:

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

   - Asincrono protetto:

     ```
     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. Definire il compilatore e le opzioni del compilatore.

   Consultare le opzioni per le diverse piattaforme riportate in Opzioni di creazione MQTT per le diverse piattaforme.

   a) Impostare Microsoft Visual C++ come compilatore.

   ```
   CC = cl
   ```

   b) Aggiungere le opzioni del pre - processore.

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

c) Aggiungere le opzioni del compilatore.

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

d) Aggiungere i percorsi di inclusione.

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

e) Opzionale: Aggiungere un'opzione di pre - processore, se si sta creando una libreria protetta.

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

f) Opzionale: Aggiungere i file di intestazione OpenSSL , se si sta creando una libreria protetta.

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

Suggerimento: I file di intestazione sono in `${OPENSSL_DIR}/inc32/openssl`, ma il file `ssl.h` è incluso con "`openssl/ssl.h`".

6. Impostare le opzioni linker e linker.

a) Impostare Microsoft Visual C+ + come linker.

```
LD = link
```

b) Aggiungere le opzioni del linker.

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

c) Aggiungere i percorso della libreria.

```
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) Aggiungere i file di output intermedi.

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

e) Opzionale: Aggiungere le librerie OpenSSL , se si sta creando una libreria sicura.

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

f) Opzionale: Aggiungere il percorso della libreria OpenSSL , se si sta creando una libreria protetta.

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

7. Definire le quattro destinazioni di build.

a) Definire la destinazione **`all`** .

Suggerimento: Ogni riga successiva che definisce l'implementazione di una destinazione deve iniziare con un carattere di tabulazione.

La destinazione "all" crea tutte le librerie.

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

b) Creare la libreria sincrona non protetta, `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}
```

L'istruzione `-rm ${CURDIR}/MQTTAsync.obj` elimina qualsiasi `MQTTAsync.obj` creata per una destinazione precedente. `MQTTAsync.obj` e `MQTTClient.obj` si escludono a vicenda.

c) Creare la libreria asincrona non protetta, `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}
```

L'istruzione `-rm ${CURDIR}/MQTTClient.obj` elimina qualsiasi `MQTTClient.obj` creata per una destinazione precedente. `MQTTAsync.obj` e `MQTTClient.obj` si escludono a vicenda.

d) Creare la libreria protetta e sincrona, `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}
```

L'istruzione `-rm ${CURDIR}/MQTTAsync.obj` elimina qualsiasi `MQTTAsync.obj` creata per una destinazione precedente. `MQTTAsync.obj` e `MQTTClient.obj` si escludono a vicenda.

e) Creare la libreria protetta asincrona, `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}
```

L'istruzione `-rm $(CURDIR}/MQTTClient.obj` elimina qualsiasi `MQTTClient.obj` creata per una destinazione precedente. `MQTTAsync.obj` e `MQTTClient.obj` si escludono a vicenda.

f) Definire la destinazione **clean** .

La destinazione "clean" rimuove tutti i file e le directory generati dal makefile

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

8. Impostare il percorso Windows per eseguire il makefile.

Impostare le parti in corsivo in modo che corrispondano all'installazione.

a) Impostare l'ambiente Microsoft Visual Studio.

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

b) Impostare la variabile `Path` per includere il programma make e l'ambiente del comando Linux .

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

9. Eseguire il makefile.

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

Suggerimento: Il carattere separatore file deve essere una barra, non una barra rovesciata.

Risultati

I seguenti file vengono creati nella directory *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
```

Elenco makefile `MQTTwin.mak`

```
# L'output di build viene prodotto nella directory corrente.
# MQTTCLIENT_DIR deve puntare alla directory di base contenente il codice sorgente client MQTT.
# MQTTCLIENT_DIR predefinito è la directory corrente
# Il valore predefinito di OPENSSL_DIR è sdkroot\openSSL, relativo a
sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR deve puntare alla directory di base contenente la creazione OpenSSL .
# Esempio: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Impostare l'ambiente di build, ad esempio:
# %comspec% /k "" C:\Programmi\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
# set path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
MQTTCLIENT_DIR ifndef
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
OPENSSL_DIR ifndef
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/ ../../../../../openssl-1.0.1c
endif

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

MQTTLIB = mqttv3c
MQTTDLL = windows_ia32/${MQTTLIB} .dll
FILE_DI_ORIGINE = ${filtro - 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 = ${filtro - 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: ripulitura
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

Creazione del pacchetto di OpenSSL

Creare il pacchetto OpenSSL prima di creare le librerie client MQTT sicure per C, `mqttv3cs` e `mqttv3as`. La build crea le librerie richieste per creare una versione sicura del client MQTT per la libreria C e lo strumento di gestione dei certificati OpenSSL .

Prima di iniziare

1. iOS La personalizzazione del makefile iOS è per le periferiche di destinazione su cui è in esecuzione iOS6. La personalizzazione potrebbe essere diversa per le versioni precedenti o successive di iOS.
2. Windows La personalizzazione del makefile Windows è per finestre a 32 bit.

Informazioni su questa attività

Scaricare e installare il package OpenSSL e qualsiasi software prerequisito. Personalizzare i makefile OpenSSL e creare le librerie OpenSSL per la piattaforma di destinazione. Su Windows e Linux, crea anche lo strumento di gestione e creazione delle chiavi OpenSSL .

Procedura

1. Installare il package OpenSSL .
   a) Scaricare il pacchetto OpenSSL da OpenSSL

Importante: Il download e la ridistribuzione del package OpenSSL sono soggetti a rigide normative di importazione ed esportazione e a condizioni di licenza open source. Prestare particolare attenzione alle limitazioni e alle avvertenze prima di decidere se scaricare il pacchetto.

b) Espandere il contenuto del file compresso in *sdkroot*.

Cercare nella scheda **Notizie** sul sito OpenSSL per trovare il percorso di scaricamento dell'ultimo package. Il pacchetto viene compresso come un file tar con estensione `tar.gz`. Quando viene espanso, il package crea una cartella di livello superiore `openssl`*version*; ad esempio *openssl-1.0.1c*. Gli esempi si riferiscono al percorso della cartella come %openssl% su Windows e $openssl su iOS; ad esempio, su Windows, %openssl% è *sdkroot*\openssl-1.0.1c.

Suggerimento: Controllare il percorso di directory creato estraendo il pacchetto OpenSSL . Alcuni package hanno livelli duplicati della cartella `openssl`*version* .

2. Windows

Opzionale: Su Windows, scaricare e installare perl. Vedere perl.org.

Per l'esempio, perl è stato scaricato da ActivePerl - download.

3. iOS

Opzionale: Su iOS, creare altre tre directory.

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

Per iOS è necessario creare il pacchetto OpenSSL per tre diverse piattaforme hardware.

4. Generare il makefile OpenSSL per creare il package OpenSSL per l'hardware e il sistema operativo.
   a) Aprire una finestra comandi nella directory %openssl% o $openssl .
   b) Eseguire il comando **Configure** perl con i parametri appropriati.
   - 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

Su iOS, personalizzare il makefile OpenSSL generato per periferiche Apple diverse.

a) Crea tre copie del makefile generato, $openssl/Makefile

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

b) Modificare l'istruzione "CC=gcc" in ogni makefile.

L'istruzione CC=gcc si trova alla riga 62 o più. Modificarla con i comandi seguenti:

$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) Modificare l'istruzione "CFLAG=..." in ogni makefile.

L'istruzione è sulla riga 63 o seguenti (suddivisa in tre righe per la leggibilità):

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

L'ubicazione degli SDK mostrati dipende dalle tue scelte di installazione Xcode . La versione degli SDK dipende dal livello del sistema operativo per cui si sta creando il makefile.

Simulatore iPhone

Il makefile del simulatore iPhone è $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

I makefile iOS sono $openssl/Makefile_arm7 e $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. Eseguire il makefile generato.

- Windows

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

- iOS Su 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
```

Risultati

La build genera le librerie condivise, la libreria e i file di intestazione richiesti per creare versioni sicure della libreria del client MQTT per C.

Introduzione al client MQTT per C su iOS

Informazioni su come ottenere le applicazioni iOS per scambiare messaggi con un server MQTT . Per l'uso su periferiche iOS (ovvero, iPhone e iPad), è necessario creare la libreria client MQTT per C dal codice sorgente fornito come parte di MQTT Software Development Kit.

Prima di iniziare

1. Link a iOS Dev Center e informazioni su come sviluppare le applicazioni per iOS.
2. Ottieni un Apple Mac con OS X 10.8.2o successivo per eseguire l'IDE (integrated development environment) Xcode .
3. (Facoltativo) Configurare un ambiente di sviluppo C su Windows o Linux. È utile creare ed eseguire le applicazioni C di esempio del client MQTT su Windows o Linux prima di sviluppare un'applicazione MQTT iOS . In alternativa, studiare il codice sorgente di esempio senza creare gli esempi.
4. Per le piattaforme client MQTT di riferimento e supportate per C, consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.
5. Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .

Informazioni su questa attività

La procedura guida l'utente attraverso le seguenti operazioni:

1. Informazioni sulla programmazione per MQTT studiando, creando ed eseguendo le applicazioni di esempio del client MQTT e le librerie client MQTT per C.
2. Installare l'ambiente di sviluppo Xcode per iOS su Apple Mac.
3. Eseguire l'attività “Creazione del client MQTT per le librerie C” a pagina 31 per creare il client MQTT per le librerie C per le unità iOS .

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 . Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.
2. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.

   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.

   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*\SDK.
3. Opzionale: Familiarizza con l'API MQTT studiando Applicazione C di esempio client MQTT.

   a) Crea l' MQTT applicazione C di esempio client `MQTTV3sample.c` sincrona per Windows o Linux. Consultare “Introduzione al client MQTT per C” a pagina 26.

   b) Connettersi a un server MQTT version 3 e pubblicare e sottoscrivere gli argomenti sul server.

   c) Studia il codice di origine e la documentazione API MQTT . Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

      Informazioni su come creare e ripristinare i client MQTT e pubblicare e sottoscrivere gli argomenti MQTT studiando l'esempio sincrono. L'esempio sincrono è più semplice dell'esempio asincrono. Se MQTT non è stato programmato in precedenza, scrivere un programma MQTT sincrono per acquisire familiarità con il modello di programmazione MQTT e l'API.

   d) Creare la libreria client MQTT asincrona per C su Windows o Linux. Consultare “Creazione del client MQTT per le librerie C” a pagina 31.

   e) Creare ed eseguire l'applicazione C di pubblicazione e sottoscrizione dell'esempio client MQTT asincrono.

   f) Studia il codice sorgente dell'applicazione C asincrono di esempio del client MQTT e la documentazione di riferimento MQTT .

È necessario utilizzare l'interfaccia asincrono per scrivere un'applicazione MQTT per un dispositivo mobile. Le app ben scritte che chiamano l'interfaccia asincrona sono più reattive e allungano la durata della batteria rispetto alle app scritte sull'interfaccia sincrona.

L'interfaccia asincrona ha due gradi di asincrono:

i) Il primo grado consiste nello sbloccare l'applicazione mentre la libreria del client MQTT attende le pubblicazioni dal server.

ii) Il secondo grado è quello di sbloccare l'applicazione mentre la libreria client si connette al server, crea sottoscrizioni e pubblica pubblicazioni.

4. Scarica e installa gli strumenti di sviluppo iOS .
   a. Collegarsi con un ID utente con privilegi di gestione.
   b. Verificare che Apple Mac sia alla versione 10.8.2 o successiva.
   c. Vai al sito web Xcode per scaricare Xcode dall'app store Mac .
   d. Installare Xcode, l'ambiente della riga comandi e il simulatore.

   Se l'app store Mac offre più versioni del simulatore, scegli la versione compatibile con il livello di iOS di destinazione per la tua applicazione.
5. Creare le librerie del client MQTT per C su iOS. Consultare “Creazione del client MQTT per le librerie C” a pagina 31.

Operazioni successive

1. Verificare le librerie client MQTT per C create:
   a. Utilizzare l'ambiente di sviluppo Xcode per compilare l' Applicazione C di esempio client MQTTasincrono e collegarsi alla libreria client MQTT asincrona non protetta per C.
   b. Utilizza l'ambiente di sviluppo Xcode per eseguire l'app C di esempio del client MQTT asincrona su un dispositivo iOS . Collegare l'esempio al server MQTT version 3 configurato; consultare “Configurazione del servizio MQTT dalla riga di comando” a pagina 140.
2. Crea un'applicazione C client MQTT per iOS. Gli esempi di codifica per l'applicazione C asincrona potrebbero rivelarsi utili. Gli esempi sono `MQTTV3ASample.c` e `MQTTV3ASSample.c` in *`sdkroot`*`\SDK\clients\c\samples`. Come esercizio, iniziare implementando l'esempio di pubblicazione / sottoscrizione MQTT .

   Suggerimento: Per avere un'idea di cosa potrebbe essere e cosa fare l'applicazione, guarda le schermate di Applicazione C di esempio client MQTT. Consultare “Introduzione a Client MQTT per Java su Android” a pagina 18.

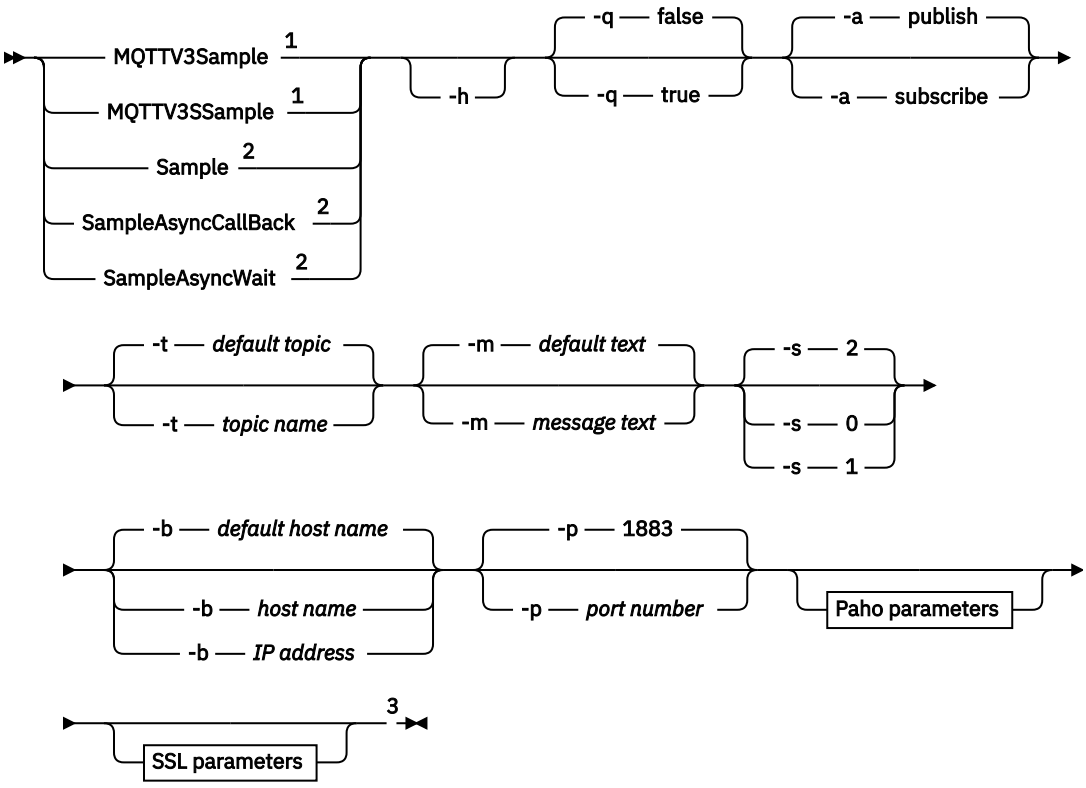
Programmi di esempio della riga comandi MQTT

La sintassi e i parametri dei programmi di esempio della riga comandi MQTT .

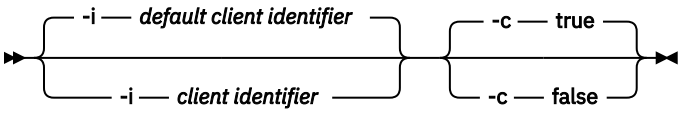
Finalità

Pubblicare e sottoscrivere un argomento.

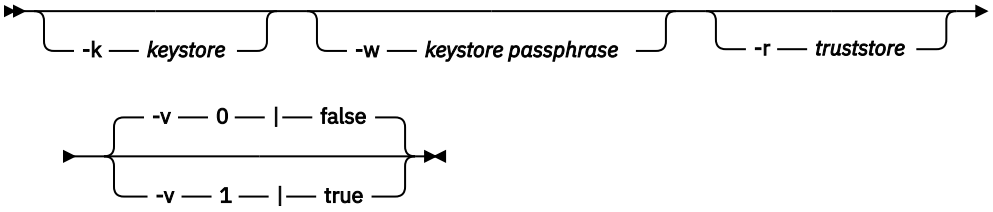
Syntax



Paho parameters



SSL parameters



Note:

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

Parametri

`-h`
Stampare questo testo di aiuto e uscire

`-q`
Impostare la modalità silenziosa, invece di utilizzare la modalità predefinita false.

`-a publish|subscribe`
Impostare l'azione su `publish` o `subscribe`, invece di assumere l'azione predefinita di pubblicazione.

-t ***nome argomento***
Pubblicare o sottoscrivere *topic name*, invece di pubblicare o sottoscrivere l'argomento predefinito. Gli argomenti predefiniti sono i seguenti:

Esempi Paho

Pubblica
`Sample/Java/v3`

Sottoscrivi
`Sample/#`

IBM WebSphere MQ esempi

Pubblica
`MQTTV3Sample/Java/v3` o `MQTTV3Sample/C/v3`

Sottoscrivi
`MQTTV3Sample/#`

-m ***testo messaggio***
Pubblicare *message text* invece di inviare il testo predefinito. Il testo predefinito è "`Message from MQTTv3 C client`"o "`Message from MQTTv3 Java client`"

-s 0|1|2
Impostare la qualità del servizio (QoS) invece di utilizzare il QoSpredefinito, 2.

-b ***nome host***
Connettersi a *host name* o all'indirizzo IP invece di connettersi al nome host predefinito. Il nome dell'host predefinito per gli esempi Paho è `m2m.eclipse.org`. Per gli esempi IBM WebSphere MQ è `localhost`.

-p ***numero porta***
Utilizzare la porta *port number* invece della porta predefinita, 1883.

Paho parametri

-i ***identificativo client***
Impostare l'identificativo client su *client identifier.* L'identificativo client predefinito è `SampleJavaV3_"+action`, dove `action` è `publish` o `subscribe`.

-c true|false
Impostare l'indicatore di sessione pulita. Il valore predefinito è `true`: le sottoscrizioni non sono durevoli.

Parametri SSL

-k ***keystore***
Impostare il percorso del keystore contenente la chiave privata che identifica il client su *keystore*. Per gli esempi C, l'archivio è un file PEM (Privacy-Enhanced Mail). Per gli esempi Java è un keystore Java (JKS).

-w ***passphrase keystore***
Impostare la passphrase per autorizzare il client ad accedere al keystore su *keystore passphrase*.

-r ***truststore***
Impostare il percorso del keystore contenente le chiavi pubbliche dei server MQTT che il client considera attendibili su *truststore*. Il keystore è un file PEM (Privacy - Enhanced Mail). Per gli esempi C, l'archivio è un file PEM (Privacy-Enhanced Mail). Per gli esempi Java è un keystore Java (JKS).

-v 0|false|1|true
Impostare l'opzione di verifica su `1|true` per richiedere un certificato server. Il valore predefinito Š `0|false`: il certificato del server non viene controllato. Il canale SSL è sempre codificato.

Impostare l'opzione su `0|1` per i programmi C e `true|false` per programmi Java .

Attività correlate

“Introduzione al client MQTT per Java” a pagina 12
Attivazione ed esecuzione con il client MQTT per applicazioni di esempio Java , utilizzando IBM MessageSight o IBM WebSphere MQ come server MQTT. Le applicazioni di esempio utilizzano una libreria client da SDK (software development toolkit) MQTT da IBM. L'applicazione di esempio `SampleAsyncCallBack` è un modello per la scrittura di applicazioni MQTT per Android e altri sistemi operativi basati su eventi.

“Introduzione al client MQTT per C” a pagina 26
Introduzione e esecuzione con il client MQTT di esempio per C su qualsiasi piattaforma su cui è possibile compilare l'origine C. Verificare che sia possibile eseguire il client MQTT di esempio per C con IBM MessageSight o IBM WebSphere MQ come server MQTT.

“Creazione del client MQTT per le librerie C” a pagina 31
Attenersi alla seguente procedura per creare il client MQTT per le librerie C. L'argomento include gli switch di compilazione e di collegamento per un numero di piattaforme ed esempi di creazione delle librerie su iOS e Windows.

Sicurezza MQTT

Tre concetti sono fondamentali per la sicurezza MQTT : identità, autenticazione e autorizzazione. L'identità consiste nel denominare il client che viene autorizzato e a cui viene fornita l'autorizzazione. L'autenticazione consiste nel dimostrare l'identit ... del client e l'autorizzazione riguarda la gestione dei diritti assegnati al client.

Prova gli esempi di protezione

- “Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro” a pagina 55
- “Connessione dell'applicazione Java di esempio del client MQTT su Android su SSL” a pagina 63
- “Autenticazione di un'applicazione Java client di MQTT con JAAS” a pagina 72
- V 7.5.0.1 “Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77
- “Creazione ed esecuzione di Applicazione C di esempio client MQTT sicuro” a pagina 85

Identità

Identificare un client MQTT in base all'identificativo client, all'ID utente o al certificato digitale pubblico. Uno o l'altro di questi attributi definisce l'identità del client. Un server MQTT autentica il certificato inviato dal client con il protocollo SSL o l'identità client con una password impostata dal client. Il server controlla le risorse a cui il client può accedere, in base all'identità del client.

Il server MQTT si identifica al client con il proprio indirizzo IP e il proprio certificato digitale. Il client MQTT usa il protocollo SSL per autenticare il certificato inviato dal server. In alcuni casi, utilizza il nome DNS del server per verificare che il server che ha inviato il certificato sia registrato come detentore del certificato.

Impostare l'identità del client in uno dei seguenti modi:

Identificativo client
La classe `MqttClient` (`MQTTClient_create` o `MQTTAsync_create` in C) imposta l'identificativo del client. Richiamare il costruttore della classe per impostare l'identificativo del client come parametro oppure restituire un identificativo del client generato in modo casuale. L'identificativo client deve essere univoco tra tutti i client che si connettono al server e non deve essere uguale al nome del gestore code sul server. Tutti i client devono avere un identificativo client, anche se non viene utilizzato per il controllo di identità. Consultare “Identificativo client” a pagina 127.

ID utente
La classe `MqttClient` (`MQTTClient_create` o `MQTTAsync_create` in C) imposta l'ID utente client come attributo di `MqttConnectOptions` (`MqttClient_ConnectOptions` in C). L'ID utente non deve essere univoco per un client.

Certificato digitale client
Il certificato digitale del client è memorizzato nel keystore del client. L'ubicazione del keystore dipende dal client:

- **Java**

  Impostare l'ubicazione e le proprietà del keystore client richiamando il metodo `setSSLProperties` di `MqttConnectOptions` e passando le proprietà del keystore. Vedi SSL Modifications to Example.java. Lo strumento **keytool** gestisce le chiavi e i keystore Java.

- **C**

  `MQTTClient_create` o `MQTTAsync_create` imposta le proprietà del keystore come attributi di `MQTTClient_SSLOptions ssl_opts`. Lo strumento **openSSL** crea e gestisce le chiavi e i keystore a cui accede il client MQTT per C.

- **Android**

  Gestire un keystore del dispositivo Android dal menu **Impostazioni** > **Sicurezza** . Caricare nuovi certificati dalla scheda SD.

Impostare l'identità del server memorizzandone la chiave privata nel keystore del server:

IBM WebSphere MQ
Il keystore del server MQTT è un attributo del canale di telemetria a cui è connesso il client.

Impostare l'ubicazione del keystore e gli attributi con IBM WebSphere MQ Explorer, o con il comando **DEFINE CHANNEL** ; consultare DEFINE CHANNEL (MQTT). Più canali possono condividere un keystore.

Autenticazione

Un client MQTT può autenticare il server MQTT a cui si connette e il server può autenticare il client che si connette ad esso.

Un client autentica un server con il protocollo SSL. Un server MQTT autentica un client con il protocollo SSL, con una password o con entrambi.

Se il client autentica il server, ma il server non autentica il client, il client è spesso noto come client anonimo. È comune stabilire una connessione client anonima su SSL e quindi autenticare il client con una password codificata dalla sessione SSL. È molto più comune autenticare un client con una password che con un certificato client, a causa del problema di distribuzione e gestione del certificato. È probabile che si trovino certificati client utilizzati in dispositivi di valore elevato come bancomat e macchine chip - and - pin e in dispositivi personalizzati, come contatori elettrici intelligenti.

Autenticazione server da un client
Un client MQTT verifica che sia connesso al server corretto autenticando il certificato del server con il protocollo SSL. Questa forma di verifica ti è familiare, quando navighi su un sito web tramite il protocollo HTTPS.

Il server invia il certificato pubblico, firmato da un'autorità di certificazione, al client. Il client utilizza la chiave pubblica dell'autorità di certificazione per verificare la firma dell'autorità di certificazione sul certificato del server. Verifica inoltre che il certificato sia aggiornato. Tali controlli stabiliscono che il certificato è valido.

I certificati dell'autorità di certificazione, spesso denominati certificati root, sono memorizzati nel truststore del client:

- **Java**

  Richiamare il metodo `setSSLProperties` di `MqttConnectOptions` e inoltrare le proprietà del truststore per impostare l'ubicazione e le proprietà del truststore client. Vedi SSL Modifications to Example.java. Gestire certificati e truststore con lo strumento **keytool** .

- **C**

`MQTTClient_create` o `MQTTAsync_create` impostare le proprietà truststore come attributi di `MQTTClient_SSLOptions ssl_opts`. Gestire certificati e truststore con lo strumento **openSSL** .

- **Android**

  Gestire un truststore del dispositivo Android dal menu **Impostazioni** > **Sicurezza** . Caricare nuovi certificati root dalla scheda SD.

Autenticazione client da un server

Un server MQTT verifica che sia connesso al client corretto autenticando il certificato client con il protocollo SSL o autenticando l'identità client con una password.

Autentica il client con la password inviata dal client al server in un'intestazione MQTT protocol . Il server potrebbe scegliere di autenticare l'identificativo client, l'ID utente o il certificato con la parola d'ordine. Dipende dal server. Di solito, il server autentica l'ID utente. Verificare le parole d'ordine su una connessione SSL che è stata protetta verificando il server, per evitare di inviare le parole d'ordine in chiaro.

- **IBM WebSphere MQ**

  IBM WebSphere MQ autentica un certificato client con il protocollo SSL. Memorizzare i certificati root nel keystore IBM WebSphere MQ Telemetry . È possibile autenticare solo un certificato client come parte dell'autenticazione SSL reciproca. Ovvero, è necessario fornire al client il certificato pubblico del server e fornire al server il certificato pubblico del client.

  IBM WebSphere MQ Telemetry utilizza la stessa memorizzazione sia per il certificato privato che per quello pubblico e per altri certificati pubblici, come i certificati root forniti dalle autorità di certificazione.

  Impostare l'ubicazione del keystore e gli attributi con IBM WebSphere MQ Explorer, o con il comando **`DEFINE CHANNEL`** ; consultare DEFINE CHANNEL (MQTT). Più canali possono condividere un keystore.

  IBM WebSphere MQ autentica l'ID utente del client, o l'identificativo del client, richiamando il servizio di autenticazione e autorizzazione Java (JAAS).

  Configurare JAAS in una stanza di configurazione `MQXRConfig` memorizzata nel file `jaas.config` . Il file è memorizzato nella directory `qmgrs\`*`QmgrName`*`\mqxr` nel percorso dati IBM WebSphere MQ .

  Controlla l'autenticità del client scrivendo un metodo `login` per `JAASLoginModule`. Consultare “Configurazione del canale di telemetria JAAS” a pagina 114.

  IBM WebSphere MQ Telemetry passa al metodo `JAASLoginModule.login` i parametri seguenti:

  – ID utente
  – Password
  – Identificativo client
  – Identificativo di rete
  – Nome canale
  – ValidPrompts

Authorization

L'autorizzazione non fa parte di MQTT protocol. Viene fornito dai server MQTT . Ciò che è autorizzato dipende da ciò che fa il server. I server MQTT sono broker di pubblicazione / sottoscrizione e le regole di autorizzazione MQTT utili controllano quali client possono connettersi al server e quali argomenti possono essere pubblicati o sottoscritti da un client. Se un client MQTT può gestire il server, più regole di autorizzazione controllano quali client possono gestire diversi aspetti del server.

Il numero di possibili clienti è enorme, quindi non è possibile autorizzare ogni cliente separatamente. Un server MQTT avrà un mezzo per raggruppare i client per profili o gruppi.

L'identità di un client, dal punto di vista dell'accesso e dell'autorizzazione, non è qualcosa che è univoco per un client MQTT . Non equiparare l'identità di un client con l'identificativo client. Possono essere gli stessi, ma sono comunemente diversi. Ad esempio, è probabile che tu abbia un nome utente comune a diversi servizi e alcuni di questi servizi cooperano in "SSO (single sign - on)". È probabile che un server MQTT a scala aziendale richiami un servizio di autorizzazione che offre identità e autorizzazioni comuni per diverse applicazioni.

IBM WebSphere MQ

IBM WebSphere MQ dispone di un servizio di autorizzazione collegabile. Il servizio di autorizzazione predefinito fornito su Windows e Linux è OAM (object authority manager). Consultare Controllo dell'accesso agli oggetti utilizzando OAM su sistemi UNIX, Linux e Windows. Associa gli ID utente e i gruppi del sistema operativo alle operazioni sugli oggetti IBM WebSphere MQ , come ad esempio argomenti e code.

È possibile configurare un canale di telemetria per accedere a IBM WebSphere MQ con un ID utente fisso. Questo è il modo in cui è impostato il canale di esempio. Oppure è possibile accedere a IBM WebSphere MQ con l'ID utente impostato dal client MQTT . L'autorizzazione dei client MQTT ad accedere agli oggetti WebSphere MQ descrive i metodi di impostazione di IBM WebSphere MQ Telemetry per ottenere un controllo di accesso client generico, medio e dettagliato.

Attività correlate

“Creazione ed esecuzione di Applicazione C di esempio client MQTT sicuro” a pagina 85
In base a un esempio Windows , puoi essere operativo con l'applicazione C di esempio sicura su qualsiasi sistema operativo per cui puoi compilare l'origine C. Verificare che sia possibile eseguire l'applicazione C di esempio su IBM MessageSight o IBM WebSphere MQ come server MQTT.

“Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro” a pagina 55
In base a un esempio Windows , puoi essere attivo e in esecuzione con l'applicazione Java di esempio sicura su IBM MessageSight o IBM WebSphere MQ come server MQTT. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile"

“Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77
Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

“Connessione dell'applicazione Java di esempio del client MQTT su Android su SSL” a pagina 63
Essere operativi con il client Android MQTT di esempio connesso a IBM WebSphere MQ su SSL.

“Autenticazione di un'applicazione Java client di MQTT con JAAS” a pagina 72
Scopri come autenticare un client con JAAS. Completare i passaggi in questa attività per modificare il programma di esempio `JAASLoginModule.java` e configurare IBM WebSphere MQ per autenticare un'app Java client MQTT con JAAS.

Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro

In base a un esempio Windows , puoi essere attivo e in esecuzione con l'applicazione Java di esempio sicura su IBM MessageSight o IBM WebSphere MQ come server MQTT. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile"

Prima di iniziare

1. È necessario avere accesso al server MQTT version 3.1 che supporta MQTT protocol su SSL.
2. Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
3. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile". Consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.
4. I canali SSL devono essere avviati.

Informazioni su questa attività

Come illustrazione, questo articolo mostra come compilare ed eseguire il Applicazione di esempio client MQTT Java sicuro su Windows dalla riga comandi.

Proteggere il canale SSL con le chiavi firmate CA o le chiavi autofirmate.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 su SSL Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare "Introduzione ai server MQTT" a pagina 136.
2. Opzionale: Installare un kit di sviluppo Java (JDK) versione 7 o successiva.

   Versione 7 è richiesto per eseguire il comando **keytool** per certificare i certificati. Se non si certificheranno i certificati, non si ha bisogno del JDK Versione 7.
3. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.

   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella MQTT. Il percorso a questa cartella viene indicato qui come *sdkroot*.

   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *sdkroot*. L'espansione crea una struttura ad albero di directory che inizia da *sdkroot*\SDK.
4. Creare ed eseguire gli script per generare coppie di chiavi e certificati e configurare IBM WebSphere MQ come server MQTT .

   Seguire la procedura riportata in "Generazione di chiavi e certificati" a pagina 95 per creare ed eseguire gli script. Gli script sono elencati anche in "Script di esempio per configurare i certificati SSL per Windows" a pagina 58.
5. Verificare che i canali SSL siano in esecuzione e che siano impostati come previsto.

   Su IBM WebSphere MQ, immettere il seguente comando in una finestra di comando:

   - 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. Creare gli script per creare ed eseguire il Applicazione di esempio client MQTT Javasicuro.

   a) Creare ed eseguire `ssjavaclient.bat` per verificare un canale SSL protetto con certificati autofirmati.

   b) Creare ed eseguire `cajavaclient.bat` per verificare un canale SSL protetto con certificati firmati dall'autorità di certificazione.

Script per eseguire il client MQTT secure Java

Eseguire gli script in "Script di esempio per configurare i certificati SSL per Windows" a pagina 58 prima di eseguire questi script.

ClientMQTT sicuro Java con certificati autofirmati.
Eseguire questo script con i certificati autofirmati creati eseguendo lo 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

Eseguire il client MQTT sicuro Java con i certificati firmati dell'autorità di certificazione.
Eseguire questo script con i certificati firmati dell'autorità di certificazione creati eseguendo lo 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

Concetti correlati

“Sicurezza MQTT” a pagina 52
Tre concetti sono fondamentali per la sicurezza MQTT : identità, autenticazione e autorizzazione. L'identità consiste nel denominare il client che viene autorizzato e a cui viene fornita l'autorizzazione. L'autenticazione consiste nel dimostrare l'identit ... del client e l'autorizzazione riguarda la gestione dei diritti assegnati al client.

Attività correlate

“Generazione di chiavi e certificati” a pagina 95
Seguire questa procedura per generare chiavi e certificati per i client Java e C, incluse le app Android e iOS e i server IBM WebSphere MQ e IBM MessageSight .

“Connessione dell'applicazione Java di esempio del client MQTT su Android su SSL” a pagina 63
Essere operativi con il client Android MQTT di esempio connesso a IBM WebSphere MQ su SSL.

“Autenticazione di un'applicazione Java client di MQTT con JAAS” a pagina 72
Scopri come autenticare un client con JAAS. Completare i passaggi in questa attività per modificare il programma di esempio `JAASLoginModule.java` e configurare IBM WebSphere MQ per autenticare un'app Java client MQTT con JAAS.

Script di esempio per configurare i certificati SSL per Windows

** **

I file di comando di esempio creano i certificati e gli archivi certificati come descritto nei passi nell'attività. Inoltre, l'esempio imposta il gestore code del client MQTT per utilizzare l'archivio certificati del server. L'esempio elimina e ricrea il gestore code richiamando lo script `SampleMQM.bat` fornito con IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` imposta i nomi e i percorsi sui certificati e gli altri parametri richiesti dai comandi **keytool** e **openSSL**. Le impostazioni sono descritte nei commenti nello 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.
```

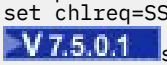
```
@rem If you then export the server certificate it is self-signed.
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The 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
```


```
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

I comandi nello script `cleancert.bat` eliminano il gestore code client MQTT per assicurare che l'archivio di certificati server non sia bloccato ed eliminare quindi tutti i keystore e i certificati creati dagli script di sicurezza di esempio.

```
@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

I comandi nello script `genkeys.bat` creano coppie di chiavi per la CA (Certificate Authority) privata, il server e un client.

```
@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

I comandi nello script `sscerts.bat` esportano i certificati autofirmati del client e del server dai relativi keystore e importano il certificato del server nel truststore del client e il certificato del client nel keystore del server. Il server non ha un truststore. I comandi creano un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script importa il certificato root CA (certificate authority) nei keystore privati. Il certificato root CA è necessario per creare la catena di chiavi tra il certificato root e il certificato firmato. Lo script `cacerts.bat` script esporta le richieste di certificato client e server dai loro keystore. Lo script firma le richieste di certificato con la chiave della CA (certificate authority) privata nel keystore `cajkskeystore.jks` e importa quindi nuovamente i certificati firmati negli stessi keystore da cui erano arrivate le richieste. L'importazione crea la catena di certificati con il certificato root CA. Lo script crea un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script elenca i keystore e i certificati nella directory dei certificati. Crea quindi il gestore code di esempio MQTT e configura i canali di telemetria sicuri.

```
@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
```

Connessione dell'applicazione Java di esempio del client MQTT su Android su SSL

Essere operativi con il client Android MQTT di esempio connesso a IBM WebSphere MQ su SSL.

Prima di iniziare

Questo articolo presume che tu stia eseguendo almeno Android API livello 14 (ICS 4.0). I livelli precedenti avevano un keystore, ma solo le app di sistema potevano accedervi.

1. È necessario avere accesso al server MQTT version 3.1 che supporta MQTT protocol su SSL.
2. Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
3. Se si sta verificando la connessione su una periferica Android iniziale, potrebbe essere necessaria una scheda SD per trasferire il certificato alla periferica.
4. Se si sta eseguendo il test della connessione su un'unità Android virtuale, configurare una scheda SD per l'unità virtuale.
5. I canali SSL devono essere avviati.

Informazioni su questa attività

Completare questa attività per eseguire Applicazione di esempio del client MQTT Java per Android su SSL. Una connessione SSL riuscita stabilisce un canale crittografato sicuro tra il dispositivo Android e il server MQTT . L'identità del server è autenticata.

Con Android, puoi autenticare il server con SSL. Puoi anche autenticare il tuo dispositivo, anche se l'applicazione di esempio non lo supporta. Per autenticare il dispositivo, utilizza l'API KeyChaino JAAS per autenticare l'ID client, l'indirizzo IP del client o il nome utente e la password forniti dall'app MQTT Android .

Qualsiasi certificato X.509 installato nel truststore Android deve essere firmato da un'autorità di certificazione. Nell'esempio, si crea un'autorità di certificazione, che firma il certificato che si installa nel dispositivo Android . Numerosi certificati root sono preinstallati nelle unità Android .

È necessario creare un blocco sul dispositivo Android prima di installare un certificato attendibile. Il blocco impedisce a qualcuno di installare i certificati sul dispositivo a tua insaputa.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 su SSL Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.
2. Esegui l'applicazione di esempio del client MQTT for Android "`MQTTExerciser`" su un canale MQTT non protetto. Consultare “Introduzione a Client MQTT per Java su Android” a pagina 18.

   Si utilizza nuovamente l'app per verificare il canale sicuro.

   Se è stata avviata un'unità virtuale Android , lasciarla in esecuzione.
3. Opzionale: Installare un kit di sviluppo Java (JDK) versione 7 o successiva.

   Versione 7 è richiesto per eseguire il comando **keytool** per certificare i certificati. Se non si certificheranno i certificati, non si ha bisogno del JDK Versione 7.
4. Creare ed eseguire gli script per generare coppie di chiavi e certificati e configurare IBM WebSphere MQ come server MQTT .

   Seguire la procedura riportata in “Generazione di chiavi e certificati” a pagina 95 per creare ed eseguire gli script. Gli script sono elencati anche in “Script di esempio per configurare i certificati SSL per Windows” a pagina 67.

   Sono necessari il certificato pubblico dell'autorità di certificazione e il keystore del server. Non sono necessari certificati client o certificati in formato `.pem` o `.p12` .
5. Verificare che i canali SSL siano in esecuzione e che siano impostati come previsto.

   Su IBM WebSphere MQ, immettere il seguente comando in una finestra di comando:

   - 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. Installare il certificato dell'autorità di certificazione nel truststore Android .

   Il file dell'autorità di certificazione nell'esempio è `cacert.cer`.

   a) Rinominare il certificato in `cacert.crt`

   b) Copiare il certificato nella memoria interna root o sulla scheda SD.

      Per un'unità virtuale in esecuzione, aprire Eclipse o eseguire l'ADB (Android Debug Bridge) per copiare il certificato nell'unità virtuale:

      Eclipse

      i) Eseguire Eclipsee aprire la prospettiva DDMS.

ii) Nella vista principale, aprire la finestra **Esplora file** .

iii) Aprire la directory `mnt/sdcard` .

iv) Trascinare il file `cacert.crt` nella directory `mnt/sdcard` .

ADB

i) Aprire una finestra di comandi e impostare la directory corrente su `android-sdk\platform-tools` nella directory di installazione android; ad esempio, `C:\Program Files\Android\android-sdk\platform-tools`.

ii) Copiare il certificato nella directory `mnt/sdcard` :

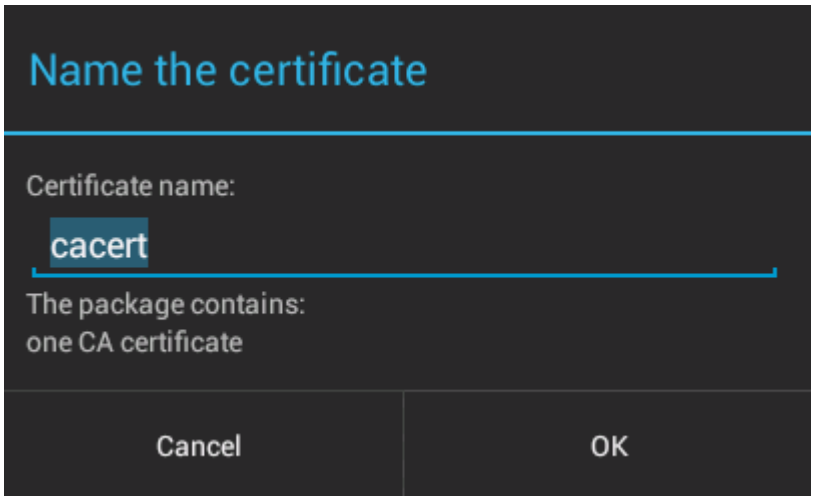
```
adb push %cacert% /mnt/sdcard/cacert.crt
```

7. Installare il certificato nel truststore del certificato sulla perifericaAndroid .

   Il certificato deve avere una clausola Basic Constraints con il valore `Subject Type=CA`.

   a) Sbloccare il dispositivo e fare clic sul pulsante **widget** .

   b) Fare clic su **Impostazioni** > **Sicurezza** > **Memoria credenziali** > **Installa dalla scheda SD**.



   c) Verificare che il nome del file del certificato sia corretto e fare clic su **OK**.

   Nota: Se non è stato definito un blocco per il dispositivo, viene ora richiesto da Android di impostare un blocco.

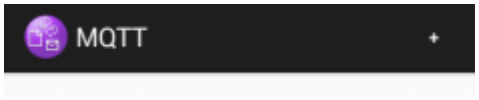
8. Confermare che il certificato è installato sulla periferica.

   a) Fai clic su **Credenziali attendibili** > **Utente** e attendi per alcuni minuti circa la visualizzazione del tuo certificato nell'elenco di certificati utente.

9. Rieseguire l'applicazione `MQTTExerciser` e connettersi a un canale MQTT configurato per i client SSL anonimi.

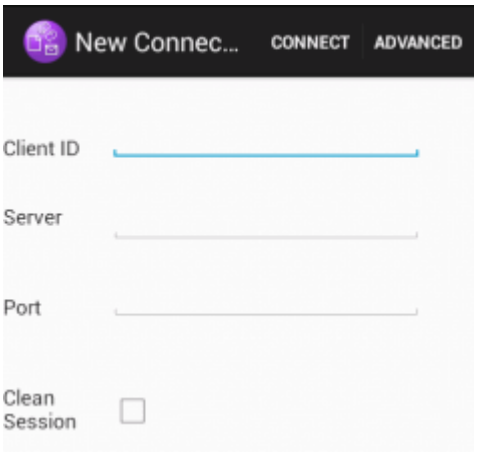
   a) Aprire il Applicazione di esempio del client MQTT Java per Android.

Questa finestra viene visualizzata nel dispositivo Android :



b) Collegarsi a un MQTT server.

i) Fare clic su **+** per aprire una nuova connessione MQTT .



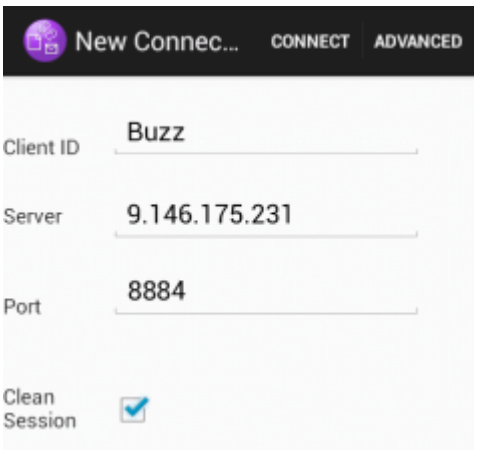
ii) Immettere un qualsiasi identificativo univoco nel campo **ID client** . Siate pazienti, le sequenze di tasti possono essere lente.

iii) Immettere nel campo **Server** l'indirizzo IP del server MQTT .

Questo è il server scelto nel primo passo principale. L'indirizzo IP non deve essere `127.0.0.1`

iv) Immettere il numero di porta della connessione MQTT .

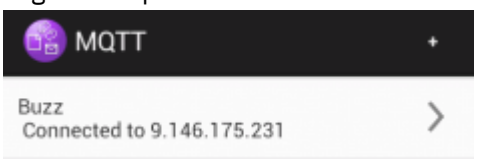
Impostare il numero di porta su 8884, impostato dalla variabile `%sslportopt%` negli script di esempio. Questo è il numero di porta del canale MQTT configurato per client SSL anonimi eseguendo gli script di esempio nel passo “4” a pagina 64.



v) Selezionare la scheda **Avanzate** e selezionare l'opzione **SSL** . Fare clic su **Salva**.

vi) Fare clic su **Connetti**.

Se la connessione ha esito positivo, viene visualizzato un messaggio "Connessione in corso" seguito da questa finestra:



Risultati

L'app `MQTTExerciser` richiede un po' più di tempo per connettersi e scambiare messaggi, ma altrimenti non si comporta in modo diverso rispetto alla connessione su una connessione non sicura.

Concetti correlati

“Sicurezza MQTT” a pagina 52
Tre concetti sono fondamentali per la sicurezza MQTT : identità, autenticazione e autorizzazione. L'identità consiste nel denominare il client che viene autorizzato e a cui viene fornita l'autorizzazione. L'autenticazione consiste nel dimostrare l'identit ... del client e l'autorizzazione riguarda la gestione dei diritti assegnati al client.

Attività correlate

“Generazione di chiavi e certificati” a pagina 95
Seguire questa procedura per generare chiavi e certificati per i client Java e C, incluse le app Android e iOS e i server IBM WebSphere MQ e IBM MessageSight .

“Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro” a pagina 55
In base a un esempio Windows , puoi essere attivo e in esecuzione con l'applicazione Java di esempio sicura su IBM MessageSight o IBM WebSphere MQ come server MQTT. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile"

“Autenticazione di un'applicazione Java client di MQTT con JAAS” a pagina 72
Scopri come autenticare un client con JAAS. Completare i passaggi in questa attività per modificare il programma di esempio `JAASLoginModule.java` e configurare IBM WebSphere MQ per autenticare un'app Java client MQTT con JAAS.

Script di esempio per configurare i certificati SSL per Windows

** **

I file di comando di esempio creano i certificati e gli archivi certificati come descritto nei passi nell'attività. Inoltre, l'esempio imposta il gestore code del client MQTT per utilizzare l'archivio certificati del server. L'esempio elimina e ricrea il gestore code richiamando lo script `SampleMQM.bat` fornito con IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` imposta i nomi e i percorsi sui certificati e gli altri parametri richiesti dai comandi **`keytool`** e **openSSL**. Le impostazioni sono descritte nei commenti nello 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

I comandi nello script `cleancert.bat` eliminano il gestore code client MQTT per assicurare che l'archivio di certificati server non sia bloccato ed eliminare quindi tutti i keystore e i certificati creati dagli script di sicurezza di esempio.

```
@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

I comandi nello script `genkeys.bat` creano coppie di chiavi per la CA (Certificate Authority) privata, il server e un client.

```
@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

I comandi nello script `sscerts.bat` esportano i certificati autofirmati del client e del server dai relativi keystore e importano il certificato del server nel truststore del client e il certificato del client nel keystore del server. Il server non ha un truststore. I comandi creano un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script importa il certificato root CA (certificate authority) nei keystore privati. Il certificato root CA è necessario per creare la catena di chiavi tra il certificato root e il certificato firmato. Lo script `cacerts.bat` script esporta le richieste di certificato client e server dai loro keystore. Lo script firma le richieste di certificato con la chiave della CA (certificate authority) privata nel keystore `cajkskeystore.jks` e importa quindi nuovamente i certificati firmati negli stessi keystore da cui erano arrivate le richieste. L'importazione crea la catena di certificati con il certificato root CA. Lo script crea un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script elenca i keystore e i certificati nella directory dei certificati. Crea quindi il gestore code di esempio MQTT e configura i canali di telemetria sicuri.

```
@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
```

Autenticazione di un'applicazione Java client di MQTT con JAAS

Scopri come autenticare un client con JAAS. Completare i passaggi in questa attività per modificare il programma di esempio `JAASLoginModule.java` e configurare IBM WebSphere MQ per autenticare un'app Java client MQTT con JAAS.

Prima di iniziare

1. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile". Consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client.
2. Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
3. È necessario avere accesso agli esempi MQXR `JAASLoginModule` e `JAASPrincipal` Java in un'installazione IBM WebSphere MQ . Gli esempi si trovano nel percorso `%MQ_FILE_PATH% \mqxr\samples.`
4. Completare la procedura su Windows o Linux; gli esempi sono tratti da Windows.
5. Per completare il passo “1” a pagina 73, è necessario disporre dell'autorizzazione per creare il gestore code `MQXR_SAMPLE_QM` su IBM WebSphere MQ.

Informazioni su questa attività

Nell'attività, vengono emessi i parametri di identificazione del client MQTT `Sample` dalla propria versione di `JAASLoginModule`. La scrittura dei parametri del client comporta la modifica del programma di esempio `JAASLoginModule` e la configurazione di IBM WebSphere MQ per caricare la propria versione di `JAASLoginModule`.

Procedura

1. Completare la procedura in “Compila ed esegui tutte le applicazioni MQTT di esempio client Java da Eclipse” a pagina 16 per eseguire il client Paho MQTT `Sample` .

   L'obiettivo è preparare un ambiente di sviluppo per sviluppare e verificare l'autenticazione JAAS . È necessario un ambiente di sviluppo Java per adattare il modulo di autenticazione JAAS . Nell'esempio, si esegue il client Paho di esempio per Java per verificare la configurazione JAAS . Per semplicità, utilizzare lo stesso ambiente di sviluppo per modificare sia il client di esempio che il modulo di login JAAS di esempio. In alternativa, verificare il modulo di login JAAS con il client MQTT per C o qualsiasi altro client MQTT .

2. Opzionale: Aggiungere un parametro nome utente e password all'esempio MQTT Paho .

   Nota: Se il client Paho per Java include i parametri nome utente e password, questo passo non è necessario. Controllare il sito di download per un aggiornamento. Consultare IBM messaging community downloads, altrimenti modificare la copia di `Sample.java`.

   a) Aprire l'explorer di package nel pacchetto `org.eclipse.paho.sample.mqttv3app` nel progetto di esempi Paho .

   b) Fare clic con il pulsante destro del mouse su `Sample.java` **Copia** > **Incolla**. Nella finestra **Conflitto nome** , immettere il nome `SampleForJAAS`.

   c) Aggiungi le seguenti righe di codice al metodo `main` .

      i) Dopo la riga "`boolean ssl = false;`", dichiarare le variabili `userName` e `password` :

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

      Per motivi di compatibilità con i server MQTT più vecchi, per impostazione predefinita non impostare i parametri password e nome utente.

      ii) Dopo la riga, "`case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;`", analizzare i due nuovi parametri di input:

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

iii) Prima della riga, "`if (action.equals("publish")) {`", aggiungere `userName` e `password` agli argomenti del costruttore `Sample` :

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

d) Aggiungere `userName` e `password` al costruttore di `Sample`.

Modifica:

```
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) Aggiungere le seguenti linee di codice al costruttore `Sample` .

Dopo la riga "`conOpt.setCleanSession(clean);`", impostare le variabili `userName` e `password` nell'oggetto `conOpt` nel metodo `Sample` :

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

f) Nei metodi `publish` e `subscribe` , modificare le seguenti righe di codice:

Modificare la linea "`client.connect();`" in

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

3. Crea un progetto Java , `JAASSample`, per il tuo esempio JAAS .
   a) Nello spazio di lavoro Eclipse , aprire la procedura guidata **Nuovo progetto Java** : fare clic su **File** > **Nuovo** > **Progetto Java**.
   b) Nel campo **Nome del progetto** , immettere `JAASSample`.
   c) Nelle opzioni JRE, selezionare `J2SE-1.5` come JRE dell'ambiente di esecuzione e fare clic su **Avanti**.

      Il JRE deve corrispondere al JRE eseguito dal server IBM WebSphere MQ . IBM WebSphere MQ Version 7.5 esegue `J2SE-1.5`.
   d) Nella finestra **Impostazioni Java** , selezionare la scheda **Librerie** e fare clic su **Aggiungi JAR esterni ...**. Passare alla`%MQ_FILE_PATH%\mqxr\lib` e selezionare **MQXR.jar**; fare clic su **Fine**.
4. Importare le classi JAAS samples `JAASLoginModule` e `JAASPrincipal` .
   a) Fare clic con il tasto destro del mouse su `JAASSample` in Esplora package **Importa ...** > **Generale** > **File system** e fare clic su **Avanti**.
   b) Passare a `%MQ_FILE_PATH%\mqxr\samples` e selezionare `JAASLoginModule.java` e `JAASPrincipal.java`; fare clic su **Fine**.
   c) Selezionare e fare clic con il tasto destro del mouse su entrambi i file Java in Esplora package, **Refactor ...** > **Sposta**.
   d) Nella finestra **Sposta** , verificare che `JAASSample` sia selezionato come destinazione per entrambi gli elementi e fare clic su **Crea package ...**
   e) Immettere `samples` nel campo **Nome** della procedura guidata **Nuovo pacchetto Java** ; fare clic su **Fine** > **OK**

   Eclipse crea le classi Java importate con una serie di avvertenze relative ai valori non utilizzati.
5. Ridenominare la classe `JAASLoginModule`

Rinominare la classe in modo che sia più semplice distinguerla dalla classe JAASLoginModule di esempio fornita con IBM WebSphere MQ.

a) Fare clic con il tasto destro del mouse su JAASloginModule.java in Esplora package **Refactor ...** > **Rinomina**.

b) Nella finestra **Rinomina unità di compilazione** , modificare il campo **Nuovo nome** da JAASloginModule a MyJAASloginModule; fare clic su **Fine**.

6. Modificare la classe MyJAASloginModule per emettere in output il contenuto dei campi di callback.

a) Aggiungere la seguente riga di codice a MyJAASloginModule.java.

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

Inserire le righe appena prima dell'istruzione:"if (true) loggedIn = true;"

b) Premere CTRL+Shift+O per riorganizzare le importazioni e salvare il file.

7. Rinominare JAASPrincipal in MyJAASPrincipal.

Ridenominare la classe per evitare confusione con la classe JAASPrincipal di esempio. Nell'esempio, lasciare inalterato il contenuto della classe MyJAASPrincipal .

8. Fornire all'ID utente che sta eseguendo i processi del gestore code le autorizzazioni read e execute per le proprie classi JAAS .

a) In Esplora risorse di Windows , aprire la directory dello spazio di lavoro Eclipse .
Nell'esempio, l'ubicazione dello spazio di lavoro Eclipse è rappresentata dalla variabile Eclipse , *workspace_loc*.

b) Individuare la directory che contiene le classi MyJAASLoginModule e MyJAASPrincipal .

Il percorso della directory è *workspace_loc*\JAASSample\bin\samples

c) Selezionare e quindi fare clic con il tasto destro del mouse su entrambe le classi, quindi fare clic su **Proprietà**; fare clic sulla scheda **Sicurezza** nella finestra **Proprietà** .

d) Fare clic su **Aggiungi ...**, immettere il nome oggetto mqme fare clic su **Controlla nomi** per verificarlo; fare clic su **OK**.

e) Selezionare **mqm** nell'elenco di **Nomi gruppo o utente** e selezionare **read and execute** e **read** nell'elenco di autorizzazioni per mqm; fare clic su OK.

9. Configurare IBM WebSphere MQ per eseguire la classe MyJAASLoginModule .

a) Aggiungere un file service.env alla configurazione di IBM WebSphere MQ per definire i percorsi classe per caricare la propria classe MyJAASLoginModule .

Creare un file service.env con la seguente istruzione del percorso classe nella directory *WMQ_DATA_PATH* :

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

Dove *user.dir* è la directory root per i file di classe compilati nello spazio di lavoroEclipse . La directory *WMQ_DATA_PATH* contiene la directory qmgrs . Vedere Ulteriori variabili di ambiente.

Suggerimento: CLASSPATH=*user.dir*\JAASSample\bin potrebbe essere l'unica istruzione nel file service.env

b) Aggiungere una sezione, MyJAASStanza, al file jaas.config per identificare la propria classe MyJAASLoginModule relativa ai percorsi classe nel file service.env .

jaas.config è nella directory mqxr del gestore code; *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.

La stanza è:

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

c) Configurare un canale IBM WebSphere MQ Telemetry con il nome della stanza di configurazione JAAS .

Eseguire il seguente comando da una finestra comandi:

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

Il comando collega il canale MyJAAS al MyJAASStanza nel file jaas.config . Non specificando un'opzione MCAUSER o specificando l'opzione USECLTID nella definizione di canale, il canale autorizza l'accesso alle risorse del gestore code con il nome utente fornito dal programma client MQTT . Nell'esempio, il nome utente fornito dal client è impostato su "Guest". Le autorizzazioni esistenti per Guest impostate dal file di comandi SampleMQM vengono utilizzate anche in questo esempio.

10. Riavviare il servizio IBM WebSphere MQ Telemetry per leggere i nuovi dati di configurazione.
Per riavviare il servizio IBM WebSphere MQ Telemetry , avviare il gestore code o il servizio da IBM WebSphere MQ Exploreroppure eseguire i seguenti comandi per la configurazione di esempio:

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

11. Eseguire il programma Sample .

Per configurare una configurazione di esecuzione per SampleForJAAS seguire la stessa procedura del passo “1” a pagina 73, con le seguenti modifiche:

a) Impostare il numero di porta su 1890 in modo che corrisponda alla configurazione del canale MQTT .

b) Aggiungere i parametri -u Guest -z password alle password nella scheda **(x) = Argomenti** per le configurazioni Subscriber e Publisher create per il programma Sample

I programmi di esempio vengono eseguiti senza alcuna modifica nell'output, ad eccezione del fatto che il numero di porta è ora 1890 piuttosto che 1883.

Nella directory *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM , aprire il file mqxr.stdout . L'output da MyJAASLoginModule viene scritto in mqxr.stdout:

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

Operazioni successive

Se il proprio esempio non funziona, leggere l'argomento per la risoluzione dei problemi per JAAS; “Risoluzione del problema: modulo di login di JAAS non richiamato dal servizio di telemetria” a pagina 184e provare questi suggerimenti per il debug.

1. Aggiungere -verbose ai parametri in *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties. Da questo log, è possibile vedere se la classe è stata caricata correttamente.

   L'output viene scritto in *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr.
2. Ricercare in *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log le eccezioni generate in MyJAASLoginModule. Ad esempio, se si tenta di emettere un passwordnull, che è un array di caratteri e non una stringa, viene generata un'eccezione.

3. Consultare *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log. Se il nome utente in userName non è autorizzato ad accedere alle risorse del gestore code e il canale è configurato senza l'opzione MCAUSER o USECLTID , gli eventuali errori vengono riportati qui.
4. Verificare che il nome della stanza in *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config sia uguale al nome nel canale MQTT configurato per la porta a cui il client Sample sta tentando di connettersi.
5. Verificare che il percorso nella sezione corrisponda al percorso della classe MyJAASLoginModule in Eclipse; ad esempio:

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

6. Per eliminare se l'errore si trova nel percorso classe nel file service.env in *WMQ_DATA_PATH* non viene rilevato correttamente, modificare la linea "set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%" in %MQ_FILE_PATH% \mqxr\bin\controlMQXR.BAT in modo da includere il percorso classe. È anche possibile eseguire l'echo del percorso classe. Tuttavia, il percorso di classe non contiene il percorso di classe impostato in service.env, quindi funziona solo se si modifica il file controlMQXR.BAT .

Concetti correlati

“Sicurezza MQTT” a pagina 52
Tre concetti sono fondamentali per la sicurezza MQTT : identità, autenticazione e autorizzazione. L'identità consiste nel denominare il client che viene autorizzato e a cui viene fornita l'autorizzazione. L'autenticazione consiste nel dimostrare l'identit ... del client e l'autorizzazione riguarda la gestione dei diritti assegnati al client.

“Configurazione del canale di telemetria JAAS” a pagina 114
Configurare JAAS per autenticare il Nome utente inviato dal client.

Attività correlate

“Risoluzione del problema: modulo di login di JAAS non richiamato dal servizio di telemetria” a pagina 184
Verificare se il modulo di login JAAS non viene richiamato dal servizio di telemetria (MQXR) e configurare JAAS per correggere il problema.

“Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro” a pagina 55
In base a un esempio Windows , puoi essere attivo e in esecuzione con l'applicazione Java di esempio sicura su IBM MessageSight o IBM WebSphere MQ come server MQTT. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile"

“Connessione dell'applicazione Java di esempio del client MQTT su Android su SSL” a pagina 63
Essere operativi con il client Android MQTT di esempio connesso a IBM WebSphere MQ su SSL.

Informazioni correlate

Variabili di ambiente aggiuntive

Connessione di MQTT messaging client per JavaScript su SSL e WebSockets

Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

Prima di iniziare

1. È necessario avere accesso a un server MQTT version 3 che supporta MQTT protocol over WebSockets.
2. Il browser deve supportare SSL e WebSocket protocol. Consultare “Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL” a pagina 173.
3. I canali SSL devono essere avviati.

Informazioni su questa attività

Completare questa attività per eseguire il client di messaggistica MQTT per le pagine di esempio JavaScript su SSL. L'attività indirizza l'utente a “Generazione di chiavi e certificati” a pagina 95 per creare certificati e configurare IBM WebSphere MQ.

Proteggere il canale SSL con le chiavi firmate CA o le chiavi autofirmate.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare MQTT protocol su sicurezza WebSockets.

   - IBM MessageSighte le release di IBM WebSphere MQ Versione 7.5.0.1 e successive, eseguire questa operazione.
2. Opzionale: Installare un kit di sviluppo Java (JDK) alla versione 7 o successiva.

   Se si sta configurando un sistema di test e si desidera utilizzare i certificati autofirmati, è necessario utilizzare il comando JDK Versione 7 **keytool** per certificare i certificati. Se si sta impostando un sistema di produzione e si stanno inviando richieste di firma del certificato (CSR) a un'autorità di certificazione esterna, non è necessario il JDK Versione 7.
3. Creare ed eseguire gli script per generare coppie di chiavi e certificati e configurare IBM WebSphere MQ come server MQTT .

   Seguire la procedura riportata in “Generazione di chiavi e certificati” a pagina 95 per creare ed eseguire gli script. Gli script sono elencati anche in “Script di esempio per configurare i certificati SSL per Windows” a pagina 80.

   Il nome comune del certificato server deve corrispondere al nome DNS del canale server. Alcuni browser accettano certificati che contengono un elenco di nomi comuni; ad esempio:

   ```
   "CN=localhost, CN=*.example.com"
   ```

   Altri browser accettano solo un nome comune. Ad esempio, Firefox, fino alla versione 18, accetta solo un nome comune. Le versioni successive potrebbero essere diverse.
4. Verificare che i canali SSL siano in esecuzione e che siano impostati come previsto.

   Su IBM WebSphere MQ, immettere il seguente comando in una finestra di comando:

   - 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
     ```
5. Installare i certificati nell'archivio certificati del browser.

   Per l'esempio, scegliere uno dei seguenti certificati server:

   a. Per un certificato server autofirmato, il certificato è `srvcertselfsigned.cer`.

   b. Per un certificato server firmato dalla propria autorità di certificazione privata, il certificato è `cacert.cer`.

   c. Per un certificato server firmato da un'autorità di certificazione esterna, verificare che il certificato root dell'autorità di certificazione sia già installato nella memorizzazione certificato.

   A seconda del supporto disponibile nel proprio browser, installare `cacert.cer` nell'elenco delle autorità di certificazione root attendibili. Consultare “Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL” a pagina 173.

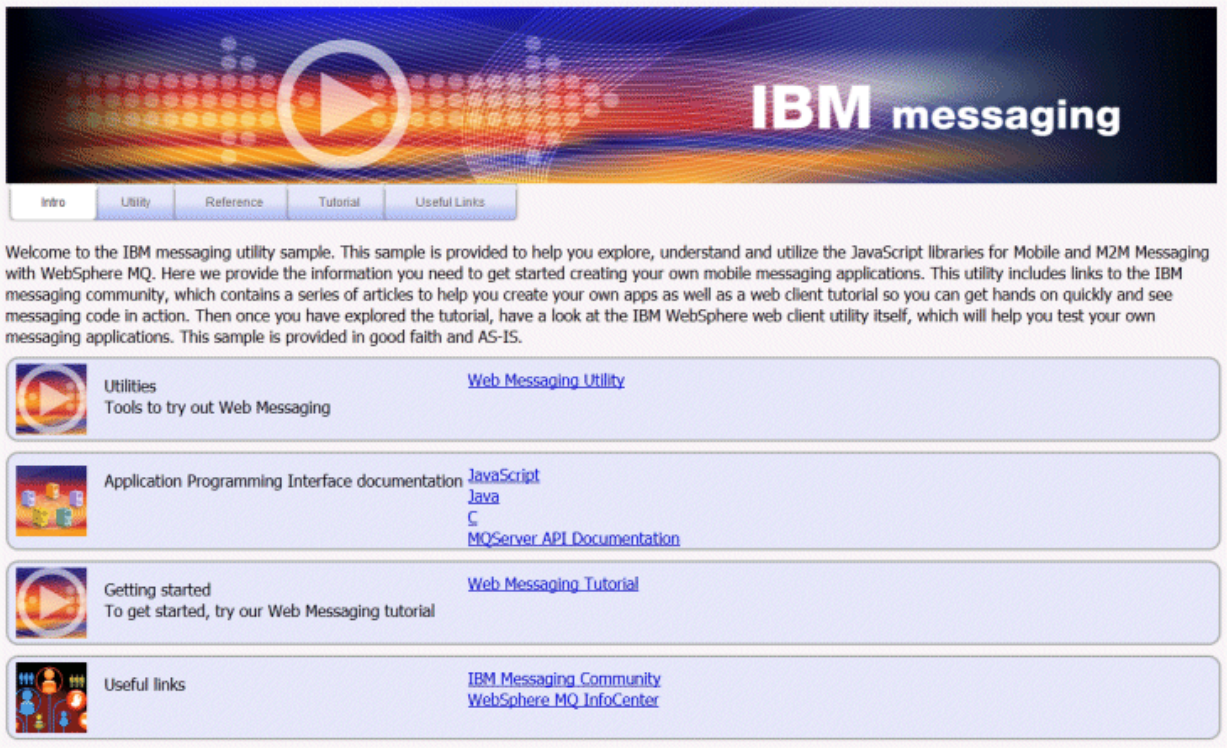
6. Opzionale: Autenticare il client.

   Nell'esempio si installa `cacert.cer` per autenticare il server e codificare il canale, ma non per autenticare il client. Per autenticare il client è necessario installare il keystore del client, `cltkeystore.p12`, nel browser. Non tutti i browser supportano i client di autenticazione. Consultare “Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL” a pagina 173.
7. Connettersi al canale WebSockets sicuro.

   Aprire il browser e digitare l'URL del canale WebSockets nella barra degli indirizzi; nell'esempio:

   ```
   https://localhost:8886
   ```

   IBM WebSphere MQ risponde con la prima pagina di Esempio di client di messaggistica MQTT JavaScript pagine.



   Se la connessione non riesce e sono stati eseguiti gli script di esempio per impostare il gestore code MQTT di esempio, provare a connettersi al canale WebSockets normale sulla porta 1886. L'esito positivo sulla porta 1886 isola l'errore nella connessione SSL.

   ```
   https://localhost:1886
   ```

Concetti correlati

“MQTT messaging client per JavaScript e le applicazioni web” a pagina 116

“Come programmare le app di messaggistica in JavaScript” a pagina 120

Attività correlate

“Generazione di chiavi e certificati” a pagina 95

Seguire questa procedura per generare chiavi e certificati per i client Java e C, incluse le app Android e iOS e i server IBM WebSphere MQ e IBM MessageSight .

“Introduzione a MQTT messaging client per JavaScript” a pagina 24

È possibile iniziare a utilizzare MQTT messaging client per JavaScript visualizzando la home page di esempio del client di messaggistica e sfogliando le risorse a cui si collega. Per visualizzare questa home page, configurare un server MQTT per accettare le connessioni da Esempio di client di messaggistica MQTT JavaScript pagine, quindi immettere l'URL configurato sul server in un browser Web. MQTT messaging client per JavaScript viene avviato automaticamente sul dispositivo e viene visualizzata la home page di esempio del client di messaggistica. Questa pagina contiene collegamenti ai programmi di

utilità, alla documentazione dell'interfaccia di programmazione, a un'esercitazione e ad altre informazioni utili.

Riferimenti correlati

“Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL” a pagina 173
Ci sono differenze nella capacità tra browser diversi, su piattaforme diverse. La comprensione di tali differenze ti aiuta a configurare le tue app, le autorità di certificazione (CA) e i certificati client per la connessione utilizzando MQTT messaging client per JavaScript su SSL e WebSockets.

Script di esempio per configurare i certificati SSL per Windows

** **

I file di comando di esempio creano i certificati e gli archivi certificati come descritto nei passi nell'attività. Inoltre, l'esempio imposta il gestore code del client MQTT per utilizzare l'archivio certificati del server. L'esempio elimina e ricrea il gestore code richiamando lo script `SampleMQM.bat` fornito con IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` imposta i nomi e i percorsi sui certificati e gli altri parametri richiesti dai comandi **keytool** e **openSSL**. Le impostazioni sono descritte nei commenti nello 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

I comandi nello script `cleancert.bat` eliminano il gestore code client MQTT per assicurare che l'archivio di certificati server non sia bloccato ed eliminare quindi tutti i keystore e i certificati creati dagli script di sicurezza di esempio.

```
@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

I comandi nello script `genkeys.bat` creano coppie di chiavi per la CA (Certificate Authority) privata, il server e un client.

```
@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

I comandi nello script `sscerts.bat` esportano i certificati autofirmati del client e del server dai relativi keystore e importano il certificato del server nel truststore del client e il certificato del client

nel keystore del server. Il server non ha un truststore. I comandi creano un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script importa il certificato root CA (certificate authority) nei keystore privati. Il certificato root CA è necessario per creare la catena di chiavi tra il certificato root e il certificato firmato. Lo script `cacerts.bat` script esporta le richieste di certificato client e server dai loro keystore. Lo script firma le richieste di certificato con la chiave della CA (certificate authority) privata nel keystore `cajkskeystore.jks` e importa quindi nuovamente i certificati firmati negli stessi keystore da cui erano arrivate le richieste. L'importazione crea la catena di certificati con il certificato root CA. Lo script crea un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script elenca i keystore e i certificati nella directory dei certificati. Crea quindi il gestore code di esempio MQTT e configura i canali di telemetria sicuri.

```
@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
```

Creazione ed esecuzione di Applicazione C di esempio client MQTT sicuro

In base a un esempio Windows , puoi essere operativo con l'applicazione C di esempio sicura su qualsiasi sistema operativo per cui puoi compilare l'origine C. Verificare che sia possibile eseguire l'applicazione C di esempio su IBM MessageSight o IBM WebSphere MQ come server MQTT.

Prima di iniziare

1. È necessario avere accesso al server MQTT version 3.1 che supporta MQTT protocol su SSL.
2. Se esiste un firewall tra il client e il server, verificare che non blocchi il traffico MQTT .
3. Le versioni binarie del client per le librerie C vengono fornite per diversi sistemi operativi. Per alcuni di questi sistemi operativi, la versione sicura del client non viene fornita come file binario. Per tali sistemi operativi, è necessario seguire le istruzioni riportate in “Creazione del client MQTT per le librerie C” a pagina 31.
4. Per la risoluzione dei problemi, il supporto IBM potrebbe richiedere l'esecuzione del client MQTT per C su una piattaforma di riferimento.
5. I canali SSL devono essere avviati.

Per una panoramica delle piattaforme supportate e di riferimento, consultare Requisiti di sistema per IBM Mobile Messaging and M2M Pacchetto client. Per i dettagli su cosa è supportato per il client C, consultare le sezioni pertinenti di Requisiti di sistema per WebSphere MQ V7.5 Telemetria.

Informazioni su questa attività

Come illustrazione, questo articolo mostra come compilare ed eseguire il Applicazione C di esempio client MQTT sicuro su Windows dalla riga comandi. Nella figura, Microsoft Visual Studio 2010 viene utilizzato per compilare il client. È possibile modificare gli script della riga comandi per compilare ed eseguire l'app di esempio su altri sistemi operativi, ad esempio Linux e iOS.

Nota:

Gli script Windows forniti in questo articolo presumono che si crei l'intero pacchetto OpenSSL dall'origine. Se si sceglie di utilizzare le librerie precompilate fornite da IBM , si potrebbe anche preferire di ottenere una release binaria precompilata di OpenSSL. Le librerie precompilate non sono disponibili per l'utilizzo con iOS.

Proteggere il canale SSL con le chiavi firmate CA o le chiavi autofirmate.

Procedura

1. Scegliere un server MQTT a cui è possibile connettere l'app client.

   Il server deve supportare il protocollo MQTT version 3.1 su SSL Tutti i server MQTT da IBM , inclusi IBM WebSphere MQ e IBM MessageSight. Consultare “Introduzione ai server MQTT” a pagina 136.
2. Opzionale: Installare un kit di sviluppo Java (JDK) versione 7 o successiva.

   Versione 7 è richiesto per eseguire il comando **keytool** per certificare i certificati. Se non si certificheranno i certificati, non si ha bisogno del JDK Versione 7.
3. Installare un ambiente di sviluppo C sulla piattaforma in cui si sta eseguendo la creazione.

   I makefile negli esempi in questo argomento sono destinati ai seguenti strumenti:

   - iOS Per iOS, su Apple Mac con OS X 10.8.2 con gli strumenti di sviluppo iOS da Xcode.
   - Linux Per Linux, gcc versione 4.4.6 da Red Hat Enterprise Linux versione 6.2.

     Il livello minimo supportato della libreria C `glibc` è 2.12e del kernel Linux è 2.6.32.
   - Windows Per Microsoft Windows, Visual Studio versione 10.0.
4. Scarica Mobile Messaging and M2M Pacchetto client e installa l'SDK MQTT .

   Non c'è alcun programma di installazione; si espande semplicemente il file scaricato.

   a. Scarica Mobile Messaging and M2M Pacchetto client.
   b. Creare una cartella in cui si installerà l'SDK.

      È opportuno denominare la cartella `MQTT`. Il percorso a questa cartella viene indicato qui come *`sdkroot`*.
   c. Espandere il contenuto del file Mobile Messaging and M2M Pacchetto client compresso in *`sdkroot`*. L'espansione crea una struttura ad albero di directory che inizia da *`sdkroot`*\SDK.
5. Opzionale: Seguire la procedura riportata in “Creazione del client MQTT per le librerie C” a pagina 31.

   Eseguire questa operazione solo se l'SDK MQTT non include la libreria del client C sicuro per il sistema operativo di destinazione.

   - Windows Le librerie sono `mqttv3cs.lib` per compilazione e `mqttv3cs.dll` per esecuzione.
   - Linux La libreria è `libmqttv3cs.so`
   - iOS La libreria è `libmqttv3cs.a`
6. Creare ed eseguire gli script per generare coppie di chiavi e certificati e configurare IBM WebSphere MQ come server MQTT .

Seguire la procedura riportata in “Generazione di chiavi e certificati” a pagina 95 per creare ed eseguire gli script. Gli script sono elencati anche in “Script di esempio per configurare i certificati SSL per Windows” a pagina 89.

7. Verificare che i canali SSL siano in esecuzione e che siano impostati come previsto.

   Su IBM WebSphere MQ, immettere il seguente comando in una finestra di comando:

   - 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. Creare gli script per creare ed eseguire il Applicazione C di esempio client MQTTsicuro.
   a) Creare ed eseguire `sscclient.bat` per verificare un canale SSL protetto con certificati autofirmati.
   b) Creare ed eseguire `cacclient.bat` per verificare un canale SSL protetto con certificati firmati dall'autorità di certificazione.

Risultati

I risultati sono simili all'esecuzione del client non protetto.

```
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. Sottoscrittore sicuro

```
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. Publisher sicuro

Script per eseguire il Applicazione C di esempio client MQTT sicuro

Eseguire gli script in “Script di esempio per configurare i certificati SSL per Windows” a pagina 89 prima di eseguire questi script.

Sicuro Applicazione C di esempio client MQTT con certificati autofirmati.
Eseguire questo script con i certificati autofirmati creati eseguendo lo 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`

Esegui l'applicazione C di esempio del client sicuro MQTT con i certificati firmati dell'autorità di certificazione.

Eseguire questo script con i certificati firmati dell'autorità di certificazione creati eseguendo lo 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`

Concetti correlati

“Sicurezza MQTT” a pagina 52

Tre concetti sono fondamentali per la sicurezza MQTT : identità, autenticazione e autorizzazione. L'identità consiste nel denominare il client che viene autorizzato e a cui viene fornita l'autorizzazione. L'autenticazione consiste nel dimostrare l'identit ... del client e l'autorizzazione riguarda la gestione dei diritti assegnati al client.

Attività correlate

“Generazione di chiavi e certificati” a pagina 95

Seguire questa procedura per generare chiavi e certificati per i client Java e C, incluse le app Android e iOS e i server IBM WebSphere MQ e IBM MessageSight .

Script di esempio per configurare i certificati SSL per Windows

Esempio

I file di comando di esempio creano i certificati e gli archivi certificati come descritto nei passi nell'attività. Inoltre, l'esempio imposta il gestore code del client MQTT per utilizzare l'archivio certificati del server. L'esempio elimina e ricrea il gestore code richiamando lo script `SampleMQM.bat` fornito con IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` imposta i nomi e i percorsi sui certificati e gli altri parametri richiesti dai comandi **keytool** e **openSSL**. Le impostazioni sono descritte nei commenti nello 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

I comandi nello script `cleancert.bat` eliminano il gestore code client MQTT per assicurare che l'archivio di certificati server non sia bloccato ed eliminare quindi tutti i keystore e i certificati creati dagli script di sicurezza di esempio.

```
@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

I comandi nello script `genkeys.bat` creano coppie di chiavi per la CA (Certificate Authority) privata, il server e un client.

```
@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

I comandi nello script `sscerts.bat` esportano i certificati autofirmati del client e del server dai relativi keystore e importano il certificato del server nel truststore del client e il certificato del client nel keystore del server. Il server non ha un truststore. I comandi creano un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script importa il certificato root CA (certificate authority) nei keystore privati. Il certificato root CA è necessario per creare la catena di chiavi tra il certificato root e il certificato firmato. Lo script `cacerts.bat` script esporta le richieste di certificato client e server dai loro keystore. Lo script firma le richieste di certificato con la chiave della CA (certificate authority) privata nel keystore `cajkskeystore.jks` e importa quindi nuovamente i certificati firmati negli stessi keystore da cui erano arrivate le richieste. L'importazione crea la catena di certificati con il certificato root CA. Lo script crea un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script elenca i keystore e i certificati nella directory dei certificati. Crea quindi il gestore code di esempio MQTT e configura i canali di telemetria sicuri.

```
@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
```

Generazione di chiavi e certificati

Seguire questa procedura per generare chiavi e certificati per i client Java e C, incluse le app Android e iOS e i server IBM WebSphere MQ e IBM MessageSight .

Prima di iniziare

1. È necessario disporre di una copia del comando **keytool** . Non tutte le versioni di **keytool** supportano la conversione di keystore da JKS (Java keystore) a PKCS (Public Key Cryptographic System) o la firma di richieste di certificati. L'esempio utilizza il comando **keytool** in JDK Versione 7.0, che supporta entrambe queste capacità.
2. Se si intende generare chiavi e certificati per il client per C, che sono in formato PEM (Privacy - Enhanced Mail), è necessario disporre di una copia del comando **openSSL** . Segui i passi in “Creazione del client MQTT per le librerie C” a pagina 31 per creare il pacchetto di openSSL .
3. Modificare i valori di parametro nello script `initcert.bat` per soddisfare le proprie esigenze. In particolare, è possibile scegliere di omettere i parametri della password per evitare di scrivere le password. Il comando **keytool** richiede di specificare le password mancanti.

Informazioni su questa attività

È necessario disporre di chiavi e certificati per creare connessioni SSL sicure tra i client e i server MQTT . Questa attività mostra due modi differenti per creare le chiavi e i certificati richiesti: autofirmati e firmati dalla propria autorità di certificazione. Il metodo che si segue dipende da come si intende gestire i keystore e i certificati.

Per utilizzare i certificati firmati da un'autorità di certificazione esterna, sostituire il passo di firma in `cacerts.bat`con l'invio delle richieste di certificato a un'autorità di certificazione esterna. L'autorità di certificazione potrebbe restituire un certificato intermedio e un certificato root in aggiunta al certificato firmato. Seguire le istruzioni fornite dalla CA esterna in cui installare i certificati restituiti.

Il server IBM WebSphere MQ ricerca i certificati solo nell'archivio certificati specificato nei parametri di configurazione del canale di telemetria. Non effettua ulteriori ricerche nel negozio JSE `cacerts` . Un client Java ricerca i certificati nel truststore specificato. Se non si specifica un truststore, lo si ricerca nel keystore `cacerts` nella directory JSE `jre\lib\security` . I client Android ricercano i certificati nell'archivio certificati predefinito sulla periferica Android . Le app client C e le app iOS effettuano la ricerca solo negli archivi di certificato specificati dall'applicazione.

I clienti Android e Java ricercano i certificati attendibili in un truststore preconfigurato. I certificati root CA vengono memorizzati nell'archivio certificati attendibili Android e nell'archivio JSE `jre\lib\security\cacerts` . Se il certificato root della CA che ha certificato il certificato server è già installato nel truststore preconfigurato, non definire un truststore client. L'unica configurazione richiesta è quella di impostare la porta TCP/IP per il canale del server MQTT protetto.

Gli strumenti per creare chiavi e certificati, e gestire tutti i diversi formati, non sono semplici da utilizzare. Hanno numerosi parametri da gestire e **openSSL** richiede un file di configurazione, `openssl.cnf`e parametri della riga comandi. Nessuno strumento fornisce tutte le funzioni richieste per gestire le chiavi e i certificati per le applicazioni in esecuzione su C e Java. I canali di telemetria in IBM WebSphere MQ richiedono un keystore JKS, pertanto gli esempi utilizzano principalmente gli strumenti di certificato Java , **ikeyman** e **keytool**. Tuttavia, gli strumenti Java non supportano il formato PEM, richiesto per le applicazioni client C. Per creare keystore in formato PEM, eseguire lo strumento **openSSL** . Lo strumento **openSSL** converte i keystore dal formato PKCS12 al formato PEM e **keytool** converte i keystore tra il formato JKS e il formato PKCS12 . Non è richiesto alcun file `openssl.cnf` per la conversione del keystore. Hai bisogno di **openSSL** solo se prevedi di creare applicazioni client C o iOS . Se si preferisce

utilizzare **openSSL**, è possibile utilizzarlo per firmare i certificati invece di firmare i certificati con **keytool**.

Procedura

1. Aprire una finestra comandi per eseguire i seguenti script.
2. Creare ed eseguire lo script `initcert.bat` per impostare i parametri richiesti per eseguire i client di esempio sicuri MQTT .
3. Creare ed eseguire lo script `cleancert.bat` per cancellare l'ambiente pronto per creare nuovi keystore e certificati.
4. Creare ed eseguire lo script `genkeys.bat` per generare le coppie chiave richieste.
5. Eseguire una di queste opzioni:
   - Creare ed eseguire lo script `sscerts.bat` per creare certificati autofirmati.
   - Creare ed eseguire lo script `cacerts.bat` per creare catene di certificati firmati dall'autorità di certificazione.
6. Creare ed eseguire lo script `mqcerts.bat` per creare il gestore code `MQXR_SAMPLE_QM` e configurare i relativi canali di telemetria.

Attività correlate

“Creazione ed esecuzione di Applicazione C di esempio client MQTT sicuro” a pagina 85
In base a un esempio Windows , puoi essere operativo con l'applicazione C di esempio sicura su qualsiasi sistema operativo per cui puoi compilare l'origine C. Verificare che sia possibile eseguire l'applicazione C di esempio su IBM MessageSight o IBM WebSphere MQ come server MQTT.

“Creazione ed esecuzione di Applicazione di esempio client MQTT Java sicuro” a pagina 55
In base a un esempio Windows , puoi essere attivo e in esecuzione con l'applicazione Java di esempio sicura su IBM MessageSight o IBM WebSphere MQ come server MQTT. Puoi eseguire un'applicazione di Client MQTT per Java su qualsiasi piattaforma con JSE 1.5 o superiore che sia "Java Compatibile"

Script di esempio per configurare i certificati SSL per Windows

Esempio

I file di comando di esempio creano i certificati e gli archivi certificati come descritto nei passi nell'attività. Inoltre, l'esempio imposta il gestore code del client MQTT per utilizzare l'archivio certificati del server. L'esempio elimina e ricrea il gestore code richiamando lo script `SampleMQM.bat` fornito con IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` imposta i nomi e i percorsi sui certificati e gli altri parametri richiesti dai comandi **keytool** e **openSSL**. Le impostazioni sono descritte nei commenti nello 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

I comandi nello script `cleancert.bat` eliminano il gestore code client MQTT per assicurare che l'archivio di certificati server non sia bloccato ed eliminare quindi tutti i keystore e i certificati creati dagli script di sicurezza di esempio.

```
@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

I comandi nello script `genkeys.bat` creano coppie di chiavi per la CA (Certificate Authority) privata, il server e un client.

```
@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

I comandi nello script `sscerts.bat` esportano i certificati autofirmati del client e del server dai relativi keystore e importano il certificato del server nel truststore del client e il certificato del client nel keystore del server. Il server non ha un truststore. I comandi creano un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script importa il certificato root CA (certificate authority) nei keystore privati. Il certificato root CA è necessario per creare la catena di chiavi tra il certificato root e il certificato firmato. Lo script `cacerts.bat` script esporta le richieste di certificato client e server dai loro keystore. Lo script firma le richieste di certificato con la chiave della CA (certificate authority) privata nel keystore `cajkskeystore.jks` e importa quindi nuovamente i certificati firmati negli stessi keystore da cui erano arrivate le richieste. L'importazione crea la catena di certificati con il certificato root CA. Lo script crea un truststore client in formato PEM dal truststore JKS client.

```
@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

Lo script elenca i keystore e i certificati nella directory dei certificati. Crea quindi il gestore code di esempio MQTT e configura i canali di telemetria sicuri.

```
@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
```

Identificazione, autorizzazione e autenticazione del client MQTT

Il servizio di telemetria (MQXR) pubblica o sottoscrive argomenti WebSphere MQ per conto dei client MQTT, utilizzando i canali MQTT. L'amministratore WebSphere MQ configura l'identità del canale MQTT utilizzata per l'autorizzazione WebSphere MQ . L'amministratore può definire un'identità comune per il canale oppure utilizzare il Nome utente o ClientIdentifier di un client connesso al canale.

Il servizio di telemetria (MQXR) può autenticare il client utilizzando il Nome utente fornito dal client o utilizzando un certificato client. Il Nome utente viene autenticato utilizzando una password fornita dal client.

Per riassumere: l'identificazione del client è la selezione dell'identità del client. In base al contesto, il client viene identificato da ClientIdentifier, Username, un'identità client comune creata dall'amministratore o un certificato client. L'identificativo client utilizzato per il controllo di autenticità non deve essere lo stesso identificativo utilizzato per l'autorizzazione.

I programmi client MQTT impostano il Nome utente e la Password inviati al server utilizzando un canale MQTT. Possono anche impostare le proprietà SSL richieste per codificare e autenticare la connessione. L'amministratore decide se autenticare il canale MQTT e come autenticare il canale.

Per autorizzare un client MQTT ad accedere agli oggetti IBM WebSphere MQ , autorizzare ClientIdentifiero Username del client oppure autorizzare un'identità client comune. Per consentire a un client di connettersi a IBM WebSphere MQ, autenticare il Nome utenteo utilizzare un certificato client. Configurare JAAS per autenticare il Nome utentee configurare SSL per autenticare un certificato client.

Se imposti una Password sul client, crittografa la connessione utilizzando VPN o configura il canale MQTT per utilizzare SSL, per mantenere la password privata.

È difficile gestire i certificati client. Per questo motivo, se i rischi associati all'autenticazione della password sono accettabili, l'autenticazione della password viene spesso utilizzata per autenticare i client.

Se esiste un modo sicuro per gestire e memorizzare il certificato client, è possibile fare affidamento sull'autenticazione del certificato. Tuttavia, raramente i certificati possono essere gestiti in modo sicuro nei tipi di ambienti in cui viene utilizzata la telemetria. Invece, l'autenticazione delle unità che utilizzano i certificati client è completata dall'autenticazione delle password client sul server. A causa della complessità aggiuntiva, l'utilizzo dei certificati client è limitato alle applicazioni altamente sensibili. L'uso di due forme di autenticazione è chiamato autenticazione a due fattori. È necessario conoscere uno dei fattori, ad esempio una password, e l'altro, ad esempio un certificato.

In un'applicazione altamente sensibile, ad esempio un dispositivo con chip e pin, il dispositivo viene bloccato durante la fabbricazione per evitare la manomissione dell'hardware e del software interni. Un certificato client sicuro, limitato nel tempo, viene copiato sul dispositivo. Il dispositivo viene distribuito nell'ubicazione in cui deve essere utilizzato. Un'ulteriore autenticazione viene eseguita ogni volta che il dispositivo viene utilizzato, utilizzando una parola d'ordine o un altro certificato da una smart card.

Identità e autorizzazione client MQTT

Utilizzare ClientIdentifier, Usernameo un'identità client comune per l'autorizzazione ad accedere agli oggetti WebSphere MQ .

L'amministratore IBM WebSphere MQ ha tre scelte per selezionare l'identità del canale MQTT. L'amministratore effettua la scelta quando definisce o modifica il canale MQTT utilizzato dal client. L'identità viene utilizzata per autorizzare l'accesso agli argomenti IBM WebSphere MQ . Le diverse opzioni sono:

1. L'identificativo del client.
2. Un'identità fornita dall'amministratore per il canale.
3. Il Nome utente passato dal client MQTT.

Nome utente è un attributo della classe di opzioni MqttConnect. Deve essere impostato prima che il client si colleghi al servizio. Il valore predefinito è null.

Utilizzare il comando IBM WebSphere MQ **setmqaut** per scegliere quali oggetti e quali azioni sono autorizzati ad essere utilizzati dall'identità associata al canale MQTT. Ad esempio, per autorizzare un'identità del canale, `MQTTClient`, fornita dall'amministratore del gestore code, `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
```

Informazioni correlate

Autorizzazione dei client MQTT ad accedere agli oggetti WebSphere MQ

Autenticazione client MQTT utilizzando una parola d'ordine

Autenticare il `Nome utente` utilizzando la password client. È possibile autenticare il client utilizzando un'identità diversa rispetto all'identità utilizzata per autorizzare il client a pubblicare e sottoscrivere argomenti.

Il servizio di telemetria (MQXR) utilizza JAAS per autenticare il client `Username`. JAAS utilizza la `Password` fornita dal client MQTT.

L'amministratore IBM WebSphere MQ decide se autenticare il `Nome utente`o non autenticarlo affatto, configurando il canale MQTT a cui si connette un client. I client possono essere assegnati a canali diversi e ciascun canale può essere configurato per autenticare i propri client in modi differenti. Utilizzando JAAS, è possibile configurare quali metodi devono autenticare il client e quali possono facoltativamente autenticare il client.

La scelta dell'identità per l'autenticazione non influisce sulla selezione dell'identità per l'autorizzazione. È possibile impostare un'identità comune per l'autorizzazione per comodità di gestione, ma autenticare ciascun utente per utilizzare tale identità. La seguente procedura descrive la procedura per autenticare i singoli utenti per utilizzare un'identità comune:

1. L'amministratore IBM WebSphere MQ imposta l'identità del canale MQTT su qualsiasi nome, ad esempio `MQTTClientUser`, utilizzando Esplora risorse di IBM WebSphere MQ .
2. L'amministratore IBM WebSphere MQ autorizza `MQTTClient` a pubblicare e sottoscrivere qualsiasi argomento:

   ```
   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. Lo sviluppatore dell'applicazione client MQTT crea un oggetto `MqttConnectOptions` e imposta `Nome utente` e `Password` prima di connettersi al server.
4. Lo sviluppatore della sicurezza crea un JAAS `LoginModule` per autenticare il `nome utente` con `Password` e lo include nel file di configurazione JAAS .
5. L'amministratore IBM WebSphere MQ configura il canale MQTT per autenticare `UserName` del client utilizzando JAAS.

Autenticazione client MQTT tramite SSL

Le connessioni tra il client MQTT e il gestore code vengono sempre avviate dal client MQTT. Il client MQTT è sempre il client SSL. L'autenticazione client del server e l'autenticazione server del client MQTT sono entrambe facoltative.

Fornire al client un certificato digitale firmato privato, è possibile autenticare il client MQTT su IBM WebSphere MQ. L'amministratore IBM WebSphere MQ può forzare i clienti MQTT ad autenticarsi al gestore code utilizzando SSL. È possibile richiedere l'autenticazione client solo come parte dell'autenticazione reciproca.

Come alternativa all'utilizzo di SSL, alcuni tipi di VPN (Virtual Private Network), ad esempio IPsec, autenticano gli endpoint di una connessione TCP/IP. VPN codifica ogni pacchetto IP che passa attraverso la rete. Una volta che tale connessione VPN è stabilita, è stata creata una rete attendibile. È possibile connettere i client MQTT ai canali di telemetria utilizzando TCP/IP sulla rete VPN.

L'autenticazione client che utilizza SSL si basa sul fatto che il client abbia un segreto. Il segreto è la chiave privata del client nel caso di un certificato autofirmato o una chiave fornita da un'autorità di certificazione. La chiave viene utilizzata per firmare il certificato digitale del client. Chiunque sia in possesso della corrispondente chiave pubblica può verificare il certificato digitale. I certificati possono essere attendibili o, se sono concatenati, è possibile risalire attraverso una catena di certificati a un certificato root attendibile. La verifica del client invia al server tutti i certificati presenti nella catena di certificati fornita dal client. Il server controlla la catena di certificati finché non trova un certificato attendibile. Il certificato attendibile è il certificato pubblico generato da un certificato autofirmato oppure un certificato root in genere emesso da un'autorità di certificazione. Come ultimo passo facoltativo, il certificato attendibile può essere confrontato con un elenco di revoche di certificati in tempo reale.

Il certificato attendibile potrebbe essere emesso da un'autorità di certificazione ed essere già incluso nell'archivio certificati JRE. Potrebbe essere un certificato autofirmato oppure un certificato che è stato aggiunto al keystore del canale di telemetria come certificato attendibile.

Nota: Il canale di telemetria include un keystore/truststore combinato che contiene sia le chiavi private per uno o più canali di telemetria, sia eventuali certificati pubblici necessari per autenticare i client. Poiché un canale SSL deve disporre di un keystore ed è lo stesso file del truststore del canale, non viene mai fatto riferimento all'archivio certificati JRE. L'implicazione è che se l'autenticazione di un client richiede un certificato root CA, è necessario posizionare il certificato root nel keystore per il canale, anche se il certificato root CA si trova già nell'archivio certificati JRE. All'archivio certificati JRE non viene mai fatto riferimento.

Occorre considerare le minacce che l'autenticazione client deve contrastare e i ruoli che il client e il server ricoprono nel contrastare le minacce. L'autenticazione del certificato client da sola non è sufficiente a impedire l'accesso non autorizzato a un sistema. Se qualcun altro ha accesso al dispositivo client, tale dispositivo non effettua necessariamente azioni autorizzate dal titolare del certificato. Non fare mai affidamento su una singola difesa contro attacchi indesiderati. Utilizzare almeno un approccio di autenticazione a due fattori e integrare il possesso di un certificato con la conoscenza di informazioni private. Ad esempio, utilizzare JAAS e autenticare il client utilizzando una password emessa dal server.

La minaccia principale al certificato client è che esso cada nelle mani sbagliate. Il certificato è contenuto in un keystore protetto da password nel client. Come viene collocato nel keystore? In che modo il client MQTT ottiene la password sul keystore? Quanto è sicura la protezione con password? I dispositivi di telemetria sono spesso facili da rimuovere e possono essere oggetto di attacchi in privato. Il dispositivo deve essere a prova di manomissione? La distribuzione e la protezione dei certificati sul lato client è notoriamente difficile: si chiama problema di gestione chiavi.

Una minaccia secondaria è che il dispositivo venga utilizzato erroneamente per accedere ai server in modi non intenzionali. Ad esempio, se l'applicazione MQTT viene manomessa, potrebbe essere possibile utilizzare un punto debole della configurazione del server utilizzando l'identità del client autenticato.

Per autenticare un client MQTT tramite SSL, configurare il canale di telemetria e il client.

-
-

Configurazione client MQTT per l'autenticazione client mediante SSL

Per autenticare il client MQTT utilizzando SSL, il client si connette a un canale di telemetria utilizzando SSL. Deve specificare una porta TCP che corrisponde a un canale di telemetria configurato per autenticare i client SSL.

Ad esempio, sul client:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

La JVM del client deve utilizzare la factory standard del socket da JSSE. Se si utilizza Java ME, è necessario verificare che il pacchetto JSSE sia caricato. Se si utilizza Java SE, JSSE è stato incluso con JRE dalla versione Java 1.4.1.

La connessione SSL richiede l'impostazione di un numero di proprietà SSL prima della connessione. È possibile impostare le proprietà trasmettendole alla JVM utilizzando lo switch `-D` oppure è possibile impostare le proprietà utilizzando il metodo `MqttConnectionOptions.setSSLProperties` .

Se si carica un factory socket non standard, richiamando il metodo `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, il modo in cui le impostazioni SSL vengono trasmesse al socket di rete viene definito dall'applicazione.

Aggiungere il certificato digitale del client, firmato utilizzando la chiave privata del client o da una CA, al keystore protetto da password sul client. Se il certificato ha una catena di chiavi, è possibile aggiungere i certificati dalla catena di chiavi all'archivio. Quando il server verifica il certificato client, utilizza i certificati inviati dal client per la corrispondenza con i certificati nel relativo keystore. Sta cercando la prima corrispondenza nella catena di chiavi con un certificato che ha. Il resto della catena di chiavi viene ignorato.

Il client MQTT invia tutti i certificati nel keystore al server. Se il server autentica una delle catene di chiavi inviate dal client, il client viene autenticato.

È inoltre possibile utilizzare le suite di cifratura SSL per l'autenticazione client. Questo è un elenco alfabetico delle suite di cifratura SSL attualmente supportate:

- `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 Se si prevede di utilizzare le suite di cifratura SHA-2 , consultare “Requisiti di sistema per l'utilizzo delle suite di cifratura SHA-2 con client MQTT” a pagina 172.

Concetti correlati

“Configurazione del client MQTT per l'autenticazione di canale mediante SSL” a pagina 107
Per autenticare il canale di telemetria utilizzando SSL, il client deve connettersi al canale di telemetria utilizzando SSL. Deve specificare una porta che corrisponde a un canale di telemetria configurato per SSL. La configurazione deve includere un keystore protetto da passphrase che contenga il certificato digitale firmato privatamente del server.

Autenticazione del canale di telemetria mediante SSL

Le connessioni tra il client MQTT e il gestore code vengono sempre avviate dal client MQTT. Il client MQTT è sempre il client SSL. L'autenticazione client del server e l'autenticazione server del client MQTT sono entrambe facoltative.

Il client tenta sempre di autenticare il server, a meno che il client non sia configurato in modo da utilizzare una CipherSpec che supporta la connessione anonima. Se l'autenticazione non riesce, la connessione non viene stabilita.

Come alternativa all'utilizzo di SSL, alcuni tipi di VPN (Virtual Private Network), ad esempio IPsec, autenticano gli endpoint di una connessione TCP/IP. VPN codifica ogni pacchetto IP che passa attraverso la rete. Una volta che tale connessione VPN è stabilita, è stata creata una rete attendibile. È possibile connettere i client MQTT ai canali di telemetria utilizzando TCP/IP sulla rete VPN.

L'autenticazione server tramite SSL autentica il server al quale si sta per inviare informazioni riservate. Il client esegue i controlli che corrispondono ai certificati inviati dal server, rispetto ai certificati collocati nel relativo truststore o nel relativo `cacerts` archivio JRE.

L'archivio certificati JRE è un file JKS, `cacerts`. Si trova in `JRE InstallPath\lib\security\`. È installato con la password predefinita `changeit`. È possibile memorizzare certificati attendibili nell'archivio certificati JRE o nel truststore del client. Non è possibile utilizzare entrambi gli archivi. Utilizzare il truststore client se si desidera tenere i certificati pubblici che il client ritiene attendibili separati dai certificati utilizzati da altre applicazioni Java. Utilizzare l'archivio certificati JRE se si desidera utilizzare un archivio certificati comune per tutte le applicazioni Java in esecuzione sul client. Se si decide di utilizzare l'archivio certificati JRE, riesaminare i certificati in esso contenuti per assicurarsi che siano attendibili.

È possibile modificare la configurazione JSSE fornendo un diverso provider di trust. È possibile personalizzare un provider di trust per eseguire diversi controlli su un certificato. In alcuni ambienti OGSi che hanno utilizzato il client MQTT, l'ambiente fornisce un provider di attendibilità differente.

Per autenticare il canale di telemetria utilizzando SSL, configurare il server e il client

•

Concetti correlati

“Configurazione del client MQTT per l'autenticazione di canale mediante SSL” a pagina 107
Per autenticare il canale di telemetria utilizzando SSL, il client deve connettersi al canale di telemetria utilizzando SSL. Deve specificare una porta che corrisponde a un canale di telemetria configurato per SSL. La configurazione deve includere un keystore protetto da passphrase che contenga il certificato digitale firmato privatamente del server.

Configurazione del client MQTT per l'autenticazione di canale mediante SSL

Per autenticare il canale di telemetria utilizzando SSL, il client deve connettersi al canale di telemetria utilizzando SSL. Deve specificare una porta che corrisponde a un canale di telemetria configurato per SSL. La configurazione deve includere un keystore protetto da passphrase che contenga il certificato digitale firmato privatamente del server.

Ad esempio, sul client:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

La JVM del client deve utilizzare la factory standard del socket da JSSE. Se si utilizza Java ME, è necessario verificare che il pacchetto JSSE sia caricato. Se si utilizza Java SE, JSSE è stato incluso con JRE dalla versione Java 1.4.1.

La connessione SSL richiede l'impostazione di un numero di proprietà SSL prima della connessione. È possibile impostare le proprietà trasmettendole alla JVM utilizzando lo switch `-D` oppure è possibile impostare le proprietà utilizzando il metodo `MqttConnectionOptions.setSSLProperties` .

Se si carica un factory socket non standard, richiamando il metodo `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, il modo in cui le impostazioni SSL vengono trasmesse al socket di rete viene definito dall'applicazione.

Codificare il client per connettersi al canale di telemetria utilizzando SSL e configurare il client per considerare attendibile un certificato server in uno dei tre seguenti modi:

Utilizzo di un certificato server firmato da un'autorità di certificazione nota nell'archivio `cacerts` .

Nessuna configurazione aggiuntiva, se il server invia tutte le chiavi intermedie nella catena di certificati. Si consiglia di esaminare i certificati nell'archivio `cacerts` del client JRE e modificare la password nell'archivio `cacerts`

Altri certificati

Memorizzare i certificati accreditati nel truststore sul client. È necessario memorizzare almeno uno dei certificati nella catena di certificati nel truststore, impostare i parametri truststore in `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Utilizzo di un gestore sicuro personalizzato

Implementare un provider di attendibilità e trasmettergli il nome dell'algoritmo utilizzato. Impostare il nome della classe provider e l'algoritmo da utilizzare in `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

È anche possibile utilizzare suite di cifratura SSL per l'autenticazione del canale. Questo è un elenco alfabetico delle suite di cifratura SSL attualmente supportate:

- `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 Se si prevede di utilizzare le suite di cifratura SHA-2 , consultare “Requisiti di sistema per l'utilizzo delle suite di cifratura SHA-2 con client MQTT” a pagina 172.

Concetti correlati

“Configurazione client MQTT per l'autenticazione client mediante SSL” a pagina 104
Per autenticare il client MQTT utilizzando SSL, il client si connette a un canale di telemetria utilizzando SSL. Deve specificare una porta TCP che corrisponde a un canale di telemetria configurato per autenticare i client SSL.

Privacy di pubblicazione sui canali di telemetria

La riservatezza delle pubblicazioni MQTT inviate in entrambe le direzioni attraverso i canali di telemetria è protetta utilizzando SSL per codificare le trasmissioni sulla connessione.

I client MQTT che si collegano ai canali di telemetria utilizzano SSL per proteggere la riservatezza delle pubblicazioni trasmesse sul canale utilizzando la crittografia a chiave simmetrica. Poiché gli endpoint non sono autenticati, non puoi considerare attendibile la sola crittografia del canale. Combina la protezione della privacy con il server o l'autenticazione reciproca.

Come alternativa all'utilizzo di SSL, alcuni tipi di VPN (Virtual Private Network), ad esempio IPsec, autenticano gli endpoint di una connessione TCP/IP. VPN codifica ogni pacchetto IP che passa attraverso la rete. Una volta che tale connessione VPN è stabilita, è stata creata una rete attendibile. È possibile connettere i client MQTT ai canali di telemetria utilizzando TCP/IP sulla rete VPN.

Per una configurazione tipica, che codifica il canale e autentica il server, consultare “Autenticazione del canale di telemetria mediante SSL” a pagina 106.

La crittografia delle connessioni SSL senza autenticare il server espone la connessione ad attacchi man - in - the - middle. Anche se le informazioni scambiate sono protette contro le intercettazioni, non sai con chi le stai scambiando. A meno che tu non controlli la rete, sei esposto a qualcuno che intercetta le tue trasmissioni IP e si maschera come l'endpoint.

Puoi creare una connessione SSL crittografata, senza autenticare il server, utilizzando uno scambio di chiavi Diffie - Hellman CipherSpec che supporta SSL anonimo. Il segreto master, condiviso tra client e server e utilizzato per codificare le trasmissioni SSL, viene stabilito senza scambiare un certificato server firmato privatamente.

Poiché le connessioni anonime non sono sicure, la maggior parte delle implementazioni SSL non utilizzano per impostazione predefinita CipherSpecsanonime. Se una richiesta client per la connessione SSL viene accettata da un canale di telemetria, il canale deve avere un keystore protetto da una passphrase. Per impostazione predefinita, poiché le implementazioni SSL non utilizzano CipherSpecsanonime, il keystore deve contenere un certificato firmato privatamente che il client può autenticare.

Se si utilizza CipherSpecsanonimo, il keystore del server deve esistere, ma non deve contenere alcun certificato firmato privatamente.

Un altro modo per stabilire una connessione crittografata consiste nel sostituire il provider di attendibilità sul client con la propria implementazione. Il provider di attendibilità non autentica il certificato del server, ma la connessione viene codificata.

Configurazione SSL di client e canali di telemetria MQTT

I client MQTT e il servizio WebSphere MQ Telemetry (MQXR) utilizzano JSSE (Java Secure Socket Extension) per connettere i canali di telemetria utilizzando SSL. I client MQTT C e il daemon WebSphere MQ Telemetry per dispositivi non supportano SSL.

Configurare SSL per autenticare il canale di telemetria e il client MQTT e codificare il trasferimento dei messaggi tra client e il canale di telemetria.

Come alternativa all'utilizzo di SSL, alcuni tipi di VPN (Virtual Private Network), ad esempio IPsec, autenticano gli endpoint di una connessione TCP/IP. VPN codifica ogni pacchetto IP che passa attraverso la rete. Una volta che tale connessione VPN è stabilita, è stata creata una rete attendibile. È possibile connettere i client MQTT ai canali di telemetria utilizzando TCP/IP sulla rete VPN.

È possibile configurare la connessione tra un client MQTT Java e un canale di telemetria per utilizzare il protocollo SSL su TCP/IP. Ciò che è protetto dipende dalla modalità di configurazione di SSL per l'utilizzo di JSSE. A partire dalla configurazione più sicura, è possibile configurare tre diversi livelli di sicurezza:

1. Consenti la connessione solo ai client MQTT attendibili. Connettere un solo client MQTT ad un canale di telemetria attendibile. Crittografare i messaggi tra il client e il gestore code; consultare “Autenticazione client MQTT tramite SSL” a pagina 103.
2. Connettere un solo client MQTT ad un canale di telemetria attendibile. Crittografare i messaggi tra il client e il gestore code; consultare “Autenticazione del canale di telemetria mediante SSL” a pagina 106.
3. Crittografare i messaggi tra il client e il gestore code; consultare “Privacy di pubblicazione sui canali di telemetria” a pagina 109.

Parametri di configurazione JSSE

Modificare i parametri JSSE per modificare il modo in cui è configurata una connessione SSL. I parametri di configurazione JSSE sono disposti in tre serie:

1. IBM WebSphere MQ Canale di telemetria
2. Client MQTT Java
3. JRE

Configurare i parametri del canale di telemetria utilizzando IBM WebSphere MQ Explorer. Impostare i parametri del client Java MQTT nell'attributo `MqttConnectionOptions.SSLProperties` . Modificare i parametri di sicurezza JRE modificando i file nella directory di sicurezza JRE sul client e sul server.

IBM WebSphere MQ canale di telemetria

Impostare tutti i parametri SSL del canale di telemetria utilizzando WebSphere MQ Explorer.

`ChannelName`

`ChannelName` è un parametro obbligatorio su tutti i canali.

Il nome del canale identifica il canale associato ad un numero di porta particolare. Denominare i canali per gestire le serie di client MQTT.

`PortNumber`

`PortNumber` è un parametro facoltativo su tutti i canali. Il valore predefinito è 1883 per i canali TCP e 8883 per quelli SSL.

Il numero di porta TCP/IP associato a questo canale. I client MQTT sono connessi a un canale specificando la porta definita per il canale. Se il canale ha proprietà SSL, il client deve connettersi utilizzando il protocollo SSL; ad esempio:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

`NomeKeyFile`

`KeyFileKeyFile` è un parametro obbligatorio per i canali SSL. Deve essere omesso per i canali TCP.

`KeyFileNome` è il percorso del keystore Java contenente certificati digitali forniti. Utilizzare JKS, JCEKS o PKCS12 come tipo di keystore sul server.

Identificare il tipo di keystore utilizzando una delle seguenti estensioni file:

```
.jks
.jceks
.p12
.pkcs12
```

Si presume che un keystore con qualsiasi altra estensione file sia un keystore JKS.

È possibile combinare un tipo di keystore sul server con altri tipi di keystore sul client.

Inserire il certificato privato del server nel keystore. Il certificato è noto come certificato server. Il certificato può essere autofirmato o parte di una concatenazione di certificati firmata da un'autorità di firma.

Se si sta utilizzando una catena di certificati, inserire i certificati associati nel keystore del server.

Il certificato del server e tutti i certificati nella sua catena di certificati vengono inviati ai client per autenticare l'identità del server.

Se `ClientAuth` è stato impostato su `Required`, il keystore deve contenere i certificati necessari per autenticare il client. Il client invia un certificato autofirmato, o una catena di certificati, e il client viene autenticato dalla prima verifica di questo materiale rispetto a un certificato nel keystore. Utilizzando una catena di certificati, un certificato può verificare molti client, anche se vengono emessi con certificati client differenti.

PassPhrase

`PassPhrase` è un parametro obbligatorio per i canali SSL. Deve essere omesso per i canali TCP.

La passphrase viene utilizzata per proteggere il keystore.

ClientAuth

`ClientAuth` è un parametro SSL facoltativo. L'impostazione predefinita è nessuna autenticazione client. Deve essere omesso per i canali TCP.

Impostare `ClientAuth` se si desidera che il servizio di telemetria (MQXR) autentichi il client, prima di consentire al client di connettersi al canale di telemetria.

Se si imposta `ClientAuth`, il client deve connettersi al server utilizzando SSL e autenticare il server. In risposta all'impostazione di `ClientAuth`, il client invia il proprio certificato digitale al server e qualsiasi altro certificato nel relativo keystore. Il suo certificato digitale è noto come certificato client. Questi certificati vengono autenticati rispetto a quelli contenuti nel keystore del canale e nell'archivio JRE `cacerts` .

CipherSuite

`CipherSuite` è un parametro SSL facoltativo. Viene utilizzato il valore predefinito per provare tutti i CipherSpecsabilitati. Deve essere omesso per i canali TCP.

Se si desidera utilizzare una particolare CipherSpec, impostare `CipherSuite` sul nome della CipherSpec che deve essere utilizzata per stabilire la connessione SSL.

Il servizio di telemetria e il client MQTT negoziano un CipherSpec comune da tutti i CipherSpecs abilitati ad ogni estremità. Se una specifica CipherSpec viene specificata in una o in entrambe le estremità della connessione, deve corrispondere alla CipherSpec nell'altra estremità.

Installare ulteriori cifrature aggiungendo ulteriori fornitori a JSSE.

FIPS (Federal Information Processing Standards)

FIPS è un'impostazione facoltativa. Per impostazione predefinita non è impostato.

Nel pannello delle proprietà del gestore code o utilizzando **runmqsc**, impostare SSLFIPS. SSLFIPS specifica se devono essere utilizzati solo algoritmi certificati FIPS.

Elenco nomi revoche

L'elenco nomi di revoca è un'impostazione facoltativa. Per impostazione predefinita non è impostato.

Nel pannello delle proprietà del gestore code o utilizzando **runmqsc**, impostare SSLCRLNL. SSLCRLNL specifica un elenco nomi degli oggetti delle informazioni di autenticazione utilizzati per fornire le posizioni di revoca dei certificati.

Non viene utilizzato nessun altro parametro del gestore code che imposta proprietà SSL.

Client MQTT Java

Impostare le proprietà SSL per il client Java in `MqttConnectionOptions.SSLProperties`; ad esempio:

```
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);
```

I nomi e i valori di specifiche proprietà sono descritti nella documentazione dell'API per `MqttConnectOptions`. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

Protocollo

`Protocollo` è facoltativo.

Il protocollo viene selezionato in negoziazione con il server di telemetria. Se si richiede un protocollo specifico, è possibile selezionarne uno. Se il server di telemetria non supporta il protocollo, la connessione ha esito negativo.

ContextProvider

`ContextProvider` è facoltativo.

KeyStore

`KeyStore` è facoltativo. Configurarlo se `ClientAuth` è impostato sul server per forzare l'autenticazione del client.

Inserire il certificato digitale del client, firmato utilizzando la chiave privata, nel keystore. Specificare il percorso e la password del keystore. Il tipo e il fornitore sono facoltativi. JKS è il tipo predefinito e IBMJCE è il fornitore predefinito.

Specificare un provider keystore differente per fare riferimento a una classe che aggiunge un nuovo provider keystore. Inoltrare il nome dell'algoritmo utilizzato dal provider keystore per istanziare il `KeyManagerFactory` impostando il nome del gestore chiavi.

TrustStore

`TrustStore` è facoltativo. È possibile inserire tutti i certificati attendibili nell'archivio JRE `cacerts` .

Configurare il truststore se si desidera avere un truststore diverso per il client. È possibile che non si configuri il truststore se il server utilizza un certificato emesso da una CA nota che ha già il certificato root memorizzato in `cacerts`.

Aggiungere il certificato firmato pubblicamente del server o il certificato root al truststore e specificare il percorso del truststore e la password. JKS è il tipo predefinito e IBMJCE è il fornitore predefinito.

Specificare un provider truststore differente per fare riferimento a una classe che aggiunge un nuovo provider truststore. Inoltrare il nome dell'algoritmo utilizzato dal provider truststore per creare un'istanza di `TrustManagerFactory` impostando il nome del gestore sicuro.

JRE

Altri aspetti della protezione Java che influiscono sul comportamento di SSL sia sul client che sul server sono configurati nel JRE. I file di configurazione su Windows si trovano in *Java Installation Directory*`\jre\lib\security`. Se si sta utilizzando il JRE fornito con IBM WebSphere MQ , il percorso è come mostrato nella seguente tabella:

Tabella 3. Percorsi file per piattaforma per i file di configurazione SSL JRE	
Piattaforma	**Percorso file**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux per System x a 32 bit	*WMQ Installation Directory*/java/jre/lib/security
Altre piattaforme UNIX and Linux	*WMQ Installation Directory*/java/jre64/jre/lib/security

Autorità di certificazione note

Il file `cacerts` contiene i certificati root delle autorità di certificazione note. `cacerts` viene utilizzato per impostazione predefinita, a meno che non si specifichi un truststore. Se si utilizza l'archivio `cacerts` o non si fornisce un truststore, è necessario esaminare e modificare l'elenco di firmatari in `cacerts` per soddisfare i propri requisiti di sicurezza.

È possibile aprire `cacerts` utilizzando il comando WebSphere MQ `strmqikm`.che esegue il programma di utilità IBM Key Management. Aprire `cacerts` come file JKS, utilizzando la password `changeit`. Modificare la password per proteggere il file.

Configurazione delle classi di sicurezza

Utilizzare il file `java.security` per registrare ulteriori provider di sicurezza e altre proprietà di sicurezza predefinite.

Autorizzazioni

Utilizzare il file `java.policy` per modificare le autorizzazioni concesse alle risorse. `javaws.policy` concede le autorizzazioni a `javaws.jar`

Livello di crittografia

Alcuni JRE vengono forniti con una crittografia di potenza ridotta. Se non è possibile importare le chiavi nei keystore, la causa potrebbe essere una codifica di livello ridotto. Prova ad avviare **ikeyman** utilizzando il comando **strmqikm** oppure scarica i file con giurisdizione limitata da IBM developer kits, Security information.

Importante: Il tuo paese di origine potrebbe avere restrizioni sull'importazione, il possesso, l'uso o la riesportazione in un altro paese, del software di crittografia. Prima di scaricare o utilizzare i file delle politiche illimitate, è necessario controllare le leggi del proprio Paese. Controllare le relative normative e le relative politiche relative all'importazione, al possesso, all'utilizzo e alla riesportazione del software di crittografia, per determinare se è consentito.

Modificare il provider di attendibilità per consentire al client di connettersi a qualsiasi server

L'esempio illustra come aggiungere un provider di attendibilità e farvi riferimento dal codice client MQTT. L'esempio non esegue alcuna autenticazione del client o del server. La connessione SSL risultante viene codificata senza essere autenticata.

Il frammento di codice in Figura 20 a pagina 113 imposta il provider e il gestore sicuro `AcceptAllProviders` per il client 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. Frammento di codice client 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

Configurazione del canale di telemetria JAAS

Configurare JAAS per autenticare il `Nome utente` inviato dal client.

L'amministratore WebSphere MQ configura quali canali MQTT richiedono l'autenticazione client utilizzando JAAS. Specificare il nome di una configurazione JAAS per ogni canale che deve eseguire l'autenticazione JAAS . I canali possono utilizzare la stessa configurazione JAAS oppure possono utilizzare configurazioni JAAS differenti. Le configurazioni sono definite in *WMQData directory*`\qmgrs\`*qMgrName*`\mqxr\jaas.config`.

Il file `jaas.config` è organizzato per nome di configurazione JAAS . Sotto ogni nome di configurazione c'è un elenco di configurazioni di login; consultare Figura 24 a pagina 115.

JAAS fornisce quattro moduli di login standard. I moduli di login NT e UNIX standard hanno un valore limitato.

Modulo JndiLogin
Esegue l'autenticazione rispetto a un servizio directory configurato in JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Autentica utilizzando i protocolli Kerberos .

NTLoginModule
Autentica utilizzando le informazioni di sicurezza NT per l'utente corrente.

Modulo UnixLogin
Autentica utilizzando le informazioni di sicurezza UNIX per l'utente attuale.

Il problema nell'utilizzare `NTLoginModule` o `UnixLoginModule` è che il servizio di telemetria (MQXR) viene eseguito con l'identità `mqm` e non con l'identità del canale MQTT. `mqm` è l'identità passata a `NTLoginModule` o `UnixLoginModule` per l'autenticazione e non l'identità del client.

Per risolvere questo problema, scrivere il proprio modulo di login o utilizzare gli altri moduli di login standard. Un esempio `JAASLoginModule.java` viene fornito con WebSphere MQ Telemetry. È un'implementazione dell'interfaccia `javax.security.auth.spi.LoginModule` . Utilizzarla per sviluppare il proprio metodo di autenticazione.

Tutte le nuove classi LoginModule fornite devono trovarsi nel percorso classi del servizio di telemetria (MQXR). Non posizionare le classi nelle directory WebSphere MQ che si trovano nel percorso di classe. Creare le proprie directory e definire l'intero percorso classe per il servizio di telemetria (MQXR).

È possibile aumentare il percorso classe utilizzato dal servizio di telemetria (MQXR) impostando il percorso classe nel file `service.env` . `CLASSPATH` deve essere in maiuscolo e l'istruzione del percorso classe può contenere solo valori letterali. Non è possibile utilizzare variabili in CLASSPATH; ad esempio, `CLASSPATH=%CLASSPATH%` non è corretto. Il servizio di telemetria (MQXR) imposta il proprio percorso classi. Il CLASSPATH definito in `service.env` viene aggiunto ad esso.

Il servizio di telemetria (MQXR) fornisce due callback che restituiscono `Username` e la `Password` per un client connesso al canale MQTT. `Nome utente` e `Password` sono impostati nell'oggetto `MqttConnectOptions` . Consultare Figura 25 a pagina 116 per un esempio di come accedere a `Nome utente` e `Password`.

Esempi

Un esempio di file di configurazione JAAS con una configurazione denominata, `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. File `jaas.config` di esempio

Un esempio di modulo di accesso JAAS codificato per ricevere il `Nome utente` e `Password` forniti da un client 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. Metodo `JAASLoginModule.Login()` di esempio

Concetti di programmazione client

I concetti descritti in questa sezione consentono di comprendere il client Java per la versione 3.1 di MQTT protocol. I concetti completano la documentazione API che accompagna il package `com.ibm.micro.client.mqttv3`.

`com.ibm.micro.client.mqttv3` contiene le classi che forniscono i metodi pubblici per le implementazioni Java del protocollo MQTT versione 3.1 . Il package `com.ibm.micro.client.mqttv3` e i package di accompagnamento che implementano il protocollo per Java SE e ME, vengono forniti con l'installazione di IBM WebSphere MQ Telemetry.

Per sviluppare ed eseguire un client MQTT , è necessario copiare o installare questi pacchetti sulla periferica client. Non è necessario installare un runtime client separato.

Le condizioni di licenza per client sono associate al server a cui si stanno collegando i client.

Il client Java è un'implementazione di riferimento della versione 3.1 di MQTT protocol. È possibile implementare i propri client in diverse lingue adatte a diverse piattaforme di dispositivi. Fare riferimento a MQ Telemetry Transport format and protocol per i dettagli.

La documentazione API client per il pacchetto `com.ibm.micro.client.mqttv3` non fa alcun presupposto su quale server è connesso il client. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT. Il funzionamento del client potrebbe differire leggermente quando si è connessi a server differenti. Le descrizioni che seguono descrivono il comportamento del client quando è connesso al servizio di telemetria IBM WebSphere MQ (MQXR).

MQTT messaging client per JavaScript e le applicazioni web

Fino a poco tempo fa, la programmazione di applicazioni web e la creazione di applicazioni di messaggistica erano discipline separate. Non importa dove si trovi la tua esperienza precedente, ci sono vantaggi significativi nell'utilizzare JavaScript e la messaggistica insieme. Quando codificate la vostra app di messaggistica come un'applicazione web, può essere estratta ed eseguita su qualsiasi browser

aggiornato. Se si modifica l'app, viene eseguito il pull dell'ultima versione ogni volta che il browser viene aggiornato. Il browser si occupa anche della sicurezza e della trasmissione affidabile dei messaggi.

Come l'utilizzo di un'applicazione web facilita la distribuzione dell'applicazione

Se hai esperienza nello sviluppo e nella distribuzione di applicazioni di messaggistica tradizionali su (ad esempio) IBM WebSphere MQ, potresti avere familiarità con il seguente processo di distribuzione:

1. L'amministratore di sistema installa o integra la libreria client.
2. L'amministratore di sistema fa in modo che l'app di messaggistica venga distribuita agli utenti finali e installata sui loro sistemi locali.
3. Quando il codice viene modificato, l'amministratore di sistema ripete i passi precedenti (in modo che la gestione delle modifiche sia complessa).

Se codificate la vostra app di messaggistica come un'applicazione web, questo è il processo di distribuzione:

1. L'amministratore del sistema serve l'app web e la libreria client in un URL.
2. Il browser dell'utente finale inserisce insieme l'app web e la libreria client.
3. Quando il codice cambia, la versione aggiornata viene rilevata quando il browser viene aggiornato (quindi la gestione delle modifiche è semplice).

Perché potresti voler utilizzare la messaggistica direttamente dal browser nelle tue app web

Se hai esperienza nella programmazione di applicazioni in JavaScript, potresti essere interessato a conoscere i benefici forniti dai sistemi di messaggistica come IBM WebSphere MQ:

- Se si inviano e si ricevono messaggi tramite un sistema di messaggistica, tale sistema è responsabile della consegna dei messaggi.
- Poiché il sistema di messaggistica si occupa della consegna, la tua applicazione web può "sparare e dimenticare". Questo semplifica notevolmente la logica di programmazione. Se i messaggi vengono consegnati per te, il tuo codice non deve controllare che siano arrivati. L'app non deve più gestire la conferma di ricezione o salvare i messaggi non recapitati e riprovare in un secondo momento.
- I sistemi di messaggistica forniscono la messaggistica basata sugli eventi. La tua app client non deve più inviare una richiesta e quindi eseguire continuamente il polling per una risposta. Invece, il server di messaggi invia un messaggio all'applicazione client quando si verifica un evento interessante. Ciò significa anche che la tua app client viene avvisata non appena si verifica l'evento, piuttosto che attendere la prossima volta che l'app esegue il polling del server.
- La messaggistica basata sugli eventi riduce inoltre in modo massiccio il carico sul dispositivo che ospita l'app del client, il traffico di rete tra il browser e il server di messaggistica e il carico sul server di messaggistica. Ciò è sempre più importante, poiché sempre più sistemi sono in esecuzione su dispositivi mobili e si collegano attraverso reti wireless.

Come i pezzi si adattano insieme

MQTT messaging client per JavaScript include una libreria client e un'app web di esempio che la utilizza. Codificate la vostra app web che usa la libreria. L'app web e la libreria client vengono quindi resi disponibili all'URL scelto, ad esempio da un gestore code MQ (come nel diagramma riportato di seguito) o da un server delle applicazioni. Il browser estrae l'applicazione web e la libreria client e l'applicazione web utilizza quindi il browser per connettersi e scambiare messaggi con un server MQTT come IBM WebSphere MQ Telemetry o IBM MessageSight.

Questi sono i flussi:

1. Ogni istanza del browser aggiorna la propria connessione all'URL su cui è disponibile l'applicazione Web e una versione aggiornata dell'app Web e della libreria client viene caricata nel browser.

2. L'applicazione web si connette a un gestore code, utilizzando MQTT su WebSocket protocole sottoscrive un argomento di interesse.
3. Il gestore code utilizza la stessa connessione per inviare messaggi che corrispondono alla sottoscrizione all'app Web.

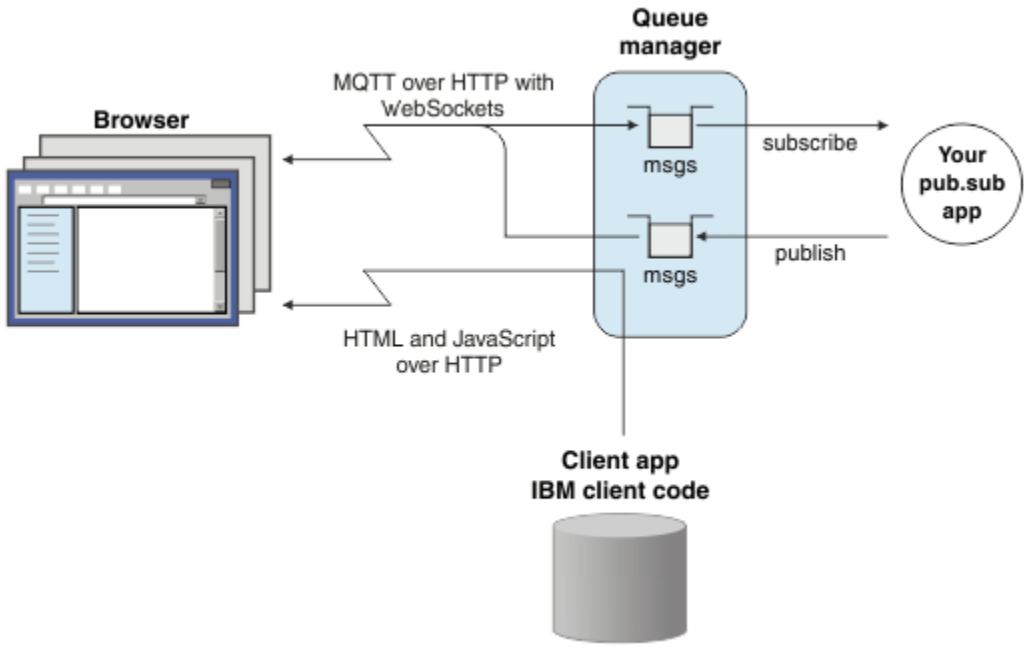


Figura 26. Utilizzo di MQTT messaging client per JavaScript con la messaggistica di pubblicazione e sottoscrizione

L'applicazione web contiene la logica dell'applicazione e l'URL del server MQTT. Quando viene aperta in un browser, l'app si connette al server MQTT, crea le sottoscrizioni di cui ha bisogno, quindi attende di ricevere gli avvisi basati su eventi e agire su di essi.

L'applicazione web si connette utilizzando MQTT come protocollo di trasporto, in esecuzione su WebSockets. La maggior parte dei browser moderni può creare connessioni WebSockets . Utilizzando WebSockets, l'applicazione web può passare i messaggi attraverso i firewall che accettano HTTP e WebSocket protocole può inviare pacchetti di dati (noti come "frame") proprio come utilizzando TCP su IP.

Quando un messaggio inviato dall'applicazione web arriva nel server MQTT, l'applicazione lato server lo vede solo come un messaggio. Non sa che il messaggio proviene da un browser.

Gestione e controllo di un server MQTT

Il server MQTT gestisce la complessità lato server della messaggistica. Garantisce la consegna dei messaggi che riceve dall'applicazione web e ospita l'applicazione di pubblicazione e sottoscrizione che risponde all'app web. Per qualsiasi server MQTT, è necessario completare la seguente procedura:

- Creare un server.
- Selezionare una porta.
- Definire un nuovo canale MQTT.
- Configurare la propria applicazione web client per connettersi alla porta scelta attraverso il nuovo canale MQTT.

Devi anche fornire l'eseguibile dell'applicazione web JavaScript al browser. Se si utilizza IBM WebSphere MQ Telemetry, per impostazione predefinita il server MQTT esegue questa operazione utilizzando lo stesso canale MQTT utilizzato dall'app web per connettersi al server MQTT. Se stai provando MQTT,

questo può aiutarti ad essere operativo rapidamente. Per l'utilizzo di produzione, in modo particolare in ambienti ad alta velocità di trasmissione, potresti preferire di servire l'eseguibile dell'applicazione web JavaScript su un canale separato, utilizzando un server delle applicazioni dedicato come WebSphere Application Server.

Nota: Poiché è progettato per ambienti ad alta velocità di trasmissione, IBM MessageSight si aspetta che tu lo faccia.

Ad esempio, se si utilizza IBM WebSphere MQ Telemetry, utilizzare la procedura guidata IBM WebSphere MQ Explorer **Nuovo canale di telemetria** per effettuare le seguenti operazioni:

1. Creare un server.
2. Selezionare una porta (1883 per impostazione predefinita).
3. Definire un nuovo canale MQTT.
4. Configurare la propria applicazione web client per connettersi alla porta scelta attraverso il nuovo canale MQTT.

L'eseguibile dell'applicazione web JavaScript viene (facoltativamente) servito anche tramite il gestore code sullo stesso canale. A tale scopo, il gestore code deve supportare sia MQTT che HTTP. Se si dispone già di un gestore code configurato per MQTT, è possibile utilizzare lo strumento della riga comandi MQSC per modificare il protocollo nella definizione del canale per supportare sia MQTT che HTTP. Vedere ALTER CHANNEL.

L'applicazione web e la libreria del client MQTT messaging client per JavaScript sono archiviate su disco in una struttura definita dal server delle applicazioni o dal gestore code. Se si utilizza IBM WebSphere MQ Telemetry, l'app web e la libreria client vengono memorizzati nella seguente struttura di directory:

```
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)
```

L'applicazione web di esempio e la libreria client sono memorizzati nella directory *MQINSTALL*/mqxr/SDK/WebContent . Il materiale in questa directory viene fornito da tutti i gestori code. Se non vuoi che i tuoi utenti vedano e utilizzino tutto questo materiale, dovresti creare la tua propria versione dell'app. Per rendere questa app, o la tua app di sostituzione, disponibile su specifici gestori code, inserisci l'applicazione nella directory *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent . Per selezionare l'app e le classi JavaScript associate da utilizzare in un URL, il gestore code cerca prima nella propria directory WebContent , quindi nella directory WebContent globale. Nella precedente struttura di directory di esempio, il gestore code serve *your_client_app* e la copia globale delle classi JavaScript .

Per arrestare il gestore code che serve i file eseguibili dell'app Web o per modificare il punto in cui il gestore code cerca i file eseguibili, configurare la proprietà **webcontentpath** e aggiungerla al file mqxr.properties . Vedere Proprietà MQXR.

Concetti correlati

“Come programmare le app di messaggistica in JavaScript” a pagina 120

Attività correlate

“Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77

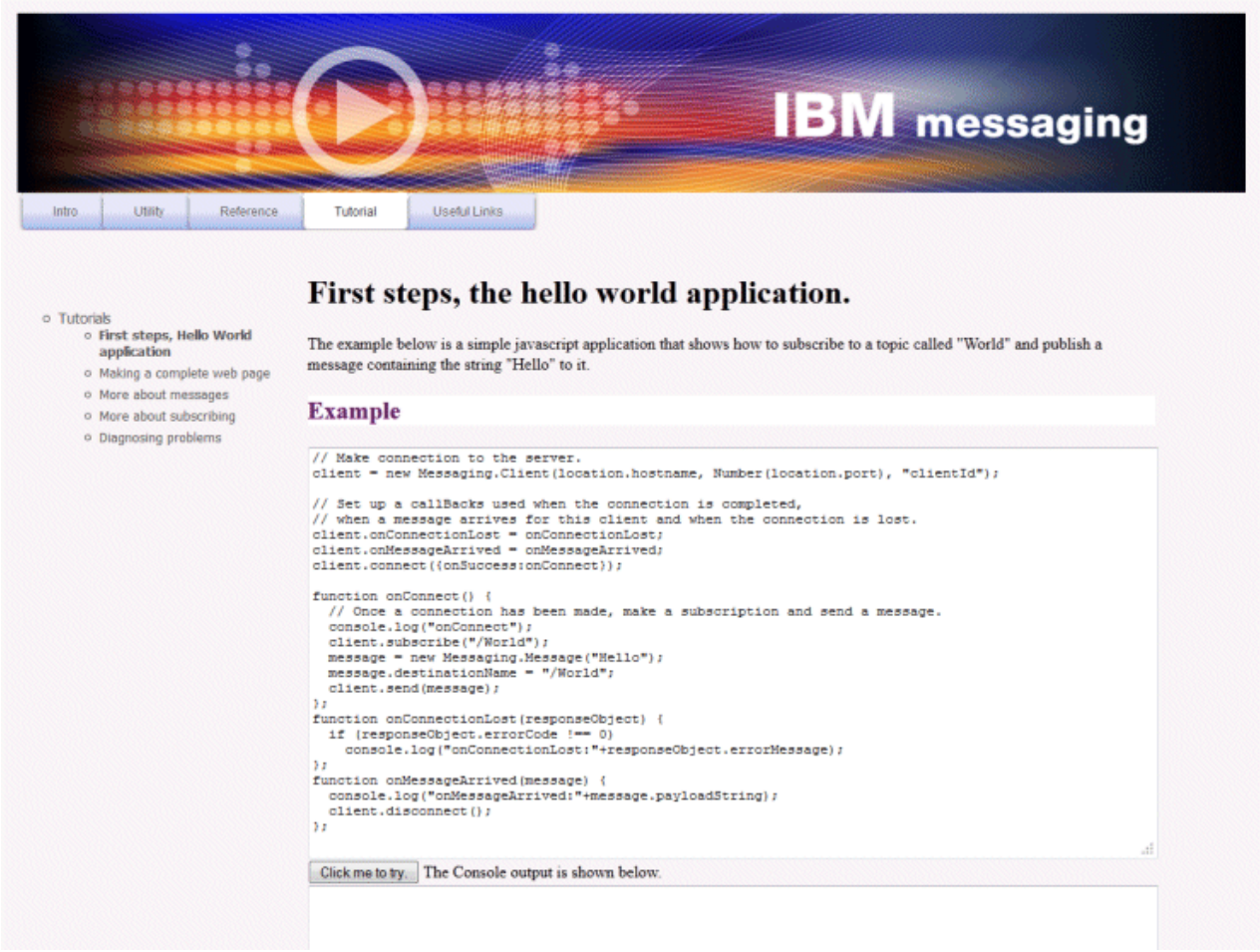
Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

“Introduzione a MQTT messaging client per JavaScript” a pagina 24
È possibile iniziare a utilizzare MQTT messaging client per JavaScript visualizzando la home page di esempio del client di messaggistica e sfogliando le risorse a cui si collega. Per visualizzare questa home page, configurare un server MQTT per accettare le connessioni da Esempio di client di messaggistica MQTT JavaScript pagine, quindi immettere l'URL configurato sul server in un browser Web. MQTT messaging client per JavaScript viene avviato automaticamente sul dispositivo e viene visualizzata la home page di esempio del client di messaggistica. Questa pagina contiene collegamenti ai programmi di utilità, alla documentazione dell'interfaccia di programmazione, a un'esercitazione e ad altre informazioni utili.

Come programmare le app di messaggistica in JavaScript

MQTT messaging client per JavaScript include un'esercitazione che illustra come creare una semplice app web di pubblicazione e sottoscrizione. Esplorando il codice dell'applicazione "Primi passi, Hello world", è possibile ottenere una comprensione di base della meccanica della programmazione di app web per la messaggistica.

Se la tua esperienza finora è stata principalmente nello sviluppo e distribuzione di applicazioni di messaggistica tradizionali, potresti trovare utile anche la sezione “Suggerimenti per la codifica JavaScript” a pagina 121 . Se sei uno sviluppatore esperto di JavaScript che non ha esperienza nella messaggistica, troverai una breve introduzione ai concetti chiave della messaggistica nella sezione “Informazioni di base sulla messaggistica” a pagina 123 .



Suggerimenti per la codifica JavaScript

Se si è abituati a sviluppare applicazioni di messaggistica, ma non si hanno applicazioni Web, potrebbero essere utili i seguenti suggerimenti:

Impacchettamento del codice per ogni evento in un callback onSuccess

Quando si codifica un'applicazione di messaggistica, si codificano i seguenti eventi nel seguente ordine:

1. connect
2. sottoscrivere
3. pubblicazione
4. ricezione messaggio

L'API MQTT messaging client per JavaScript è completamente asincrona, il che significa che il tuo thread dell'applicazione non si blocca in attesa che le chiamate come la connessione o la sottoscrizione diventino effettive. Invece, queste chiamate segnalano il loro completamento richiamando un callback `onSuccess` o `onFailure` . Per essere certi che ogni evento sia stato completato prima che venga attivato l'evento successivo, devi impacchettare il codice per ogni evento in un callback `onSuccess` . Ad esempio, l'applicazione JavaScript potrebbe ritornare dall'effettuare la chiamata di connessione prima che la connessione sia stata creata. Per essere certi che la connessione si sia verificata prima della sottoscrizione, è necessario inserire il codice di sottoscrizione in un callback `onSuccess` per la connessione.

Il codice dell'applicazione "First steps, Hello world" utilizza questo approccio.

Integrazione del codice dell'applicazione all'interno della markup HTML

Di seguito è riportato una pagina JavaScript di esempio:

Ecco l'origine della pagina precedente, per mostrare come il codice dell'applicazione è incorporato all'interno del contrassegno 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>
```

Informazioni di base sulla messaggistica

Di seguito sono riportate alcune informazioni di messaggistica di background per gli sviluppatori di applicazioni Web che non hanno esperienza con la messaggistica:

Messaggistica asincrona e fire - and - forget.

Il protocollo MQTT supporta la consegna garantita e i trasferimenti fire - and - forget. Nel protocollo, la consegna del messaggio è asincrona: l'app passa il messaggio all'API client e non intraprende ulteriori azioni per garantire che il messaggio venga consegnato. Questo approccio è noto come *fire - and - forget*. Quando una risposta è disponibile, viene inviata automaticamente alla app.

La consegna asincrona libera l'app da qualsiasi connessione server e dall'attesa di messaggi. Il modello di interazione è simile all'email, ma ottimizzata per la programmazione delle applicazioni.

Consultare anche la sezione "Protocollo MQTT" di “Introduzione a MQTT” a pagina 5

Una panoramica della messaggistica di pubblicazione e sottoscrizione.

Il fornitore di informazioni è denominato *publisher*. Un editore fornisce informazioni su un argomento, senza che sia necessario conoscere le applicazioni interessate a tali informazioni. Un publisher sceglie un *argomento*, che è un contenitore per i messaggi su un oggetto specifico. Il publisher genera quindi ogni parte di informazioni per tale oggetto come messaggio, denominato *pubblicazione*, e le pubblica nell'argomento associato.

L'utente delle informazioni è chiamato *sottoscrittore*. Un sottoscrittore crea una *sottoscrizione* per un argomento a cui è interessato. Quando un nuovo messaggio viene inviato all'argomento, il messaggio viene inoltrato a tutti i sottoscrittori dell'argomento. I sottoscrittori possono effettuare più sottoscrizioni e possono ricevere informazioni da diversi publisher.

Vedere anche Introduzione alla IBM WebSphere MQ messaggistica di pubblicazione / sottoscrizione

La corrispondenza tra sottoscrizioni e argomenti.

Se si utilizza IBM WebSphere MQ come server MQTT, è necessario comprendere come IBM WebSphere MQ specifica gli argomenti. In IBM WebSphere MQ, un publisher crea un messaggio e lo pubblica con una stringa di argomenti che meglio si adatta all'oggetto della pubblicazione. Per ricevere pubblicazioni, un sottoscrittore crea una sottoscrizione con una stringa di argomenti corrispondente al modello per selezionare gli argomenti di pubblicazione. Il gestore code consegna le pubblicazioni ai sottoscrittori che hanno sottoscrizioni che corrispondono all'argomento della pubblicazione e sono autorizzati a ricevere le pubblicazioni.

Generalmente gli argomenti sono organizzati gerarchicamente, in strutture ad albero degli argomenti, utilizzando il carattere ' /' per creare argomenti secondari nella stringa di argomento. Gli argomenti sono nodi nella struttura ad albero degli argomenti. Gli argomenti possono essere nodi foglia senza ulteriori argomenti secondari o nodi intermedi con argomenti secondari. I sottoscrittori possono utilizzare caratteri jolly per sottoscrivere più di un argomento alla volta. Ad esempio, una sottoscrizione a `/sport/tennis` riceve solo i messaggi inviati al sottoargomento tennis, mentre una sottoscrizione a `/sport/#` riceve i messaggi inviati a qualsiasi sottoargomento di `/sport`.

Vedere anche Argomenti, Alberi degli argomentie Schemi dei wildcard.

Concetti correlati

“MQTT messaging client per JavaScript e le applicazioni web” a pagina 116

Attività correlate

“Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77
Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

“Introduzione a MQTT messaging client per JavaScript” a pagina 24
È possibile iniziare a utilizzare MQTT messaging client per JavaScript visualizzando la home page di esempio del client di messaggistica e sfogliando le risorse a cui si collega. Per visualizzare questa home page, configurare un server MQTT per accettare le connessioni da Esempio di client di messaggistica MQTT JavaScript pagine, quindi immettere l'URL configurato sul server in un browser Web. MQTT messaging client per JavaScript viene avviato automaticamente sul dispositivo e viene visualizzata la home page di esempio del client di messaggistica. Questa pagina contiene collegamenti ai programmi di

utilità, alla documentazione dell'interfaccia di programmazione, a un'esercitazione e ad altre informazioni utili.

Callback e sincronizzazione nelle app client MQTT

Il modello di programmazione client MQTT utilizza ampiamente i thread. I thread disaccoppiano un'applicazione client MQTT , per quanto possibile, dai ritardi nella trasmissione dei messaggi verso e dal server. Le pubblicazioni, i token di consegna e gli eventi di connessione persi vengono consegnati ai metodi in una classe di callback che implementa `MqttCallback`.

Callback

L'interfaccia `MqttCallback` dispone di tre metodi di callback; vedi un'implementazione di esempio in Callback.java.

`connectionLost(java.lang.Throwable cause)`

`connectionLost` viene chiamato quando un errore di comunicazione porta al rilascio della connessione. Viene anche richiamato se il server elimina la connessione come risultato di un errore sul server dopo che la connessione è stata stabilita. Gli errori del server vengono registrati nel log degli errori del gestore code. Il server elimina la connessione al client e il client richiama `MqttCallback.connectionLost`.

Gli unici errori remoti generati come eccezioni sullo stesso thread dell'app client sono le eccezioni da `MqttClient.connect`. Gli errori rilevati dal server una volta stabilita la connessione vengono riportati al metodo di callback `MqttCallback.connectionLost` come `throwables`.

Gli errori tipici del server che risultano in `connectionLost` sono errori di autorizzazione. Ad esempio, il server di telemetria tenta di pubblicare su un argomento per conto di un client che non è autorizzato a pubblicare sull'argomento. Tutto ciò che risulta in un codice condizione `MQCC_FAIL` restituito al server di telemetria può causare l'eliminazione della connessione.

`deliveryComplete(MqttDeliveryToken token)`

`deliveryComplete` viene chiamato dal cliente MQTT per passare un token di consegna all'applicazione del client; vedi “Token di consegna” a pagina 128. Utilizzando il token di consegna, il callback può accedere al messaggio pubblicato con il metodo `token.getMessage`.

Quando il callback dell'applicazione restituisce il controllo al client MQTT dopo essere stato richiamato dal metodo `deliveryComplete` , la consegna viene completata. Fino al completamento della consegna, i messaggi con QoS 1 o 2 vengono conservati dalla classe di persistenza.

La chiamata a `deliveryComplete` è un punto di sincronizzazione tra l'applicazione e la classe di persistenza. Il metodo `deliveryComplete` non viene mai richiamato due volte per lo stesso messaggio.

Quando il callback dell'applicazione viene restituito da `deliveryComplete` al client MQTT , il client richiama `MqttClientPersistence.remove` per i messaggi con QoS 1 o 2. `MqttClientPersistence.remove` elimina la copia memorizzata localmente del messaggio pubblicato.

Da una prospettiva di elaborazione della transazione, la chiamata a `deliveryComplete` è una transazione a fase singola che esegue il commit della consegna. Se l'elaborazione non riesce durante il callback, al riavvio del client `MqttClientPersistence.remove` viene richiamato di nuovo per eliminare la copia locale del messaggio pubblicato. Il callback non viene richiamato di nuovo. Se si utilizza il callback per memorizzare un log dei messaggi consegnati, non è possibile sincronizzare il log con il client MQTT . Se si desidera memorizzare un log in modo affidabile, aggiornare il log nella classe `MqttClientPersistence` .

Il token di consegna e il messaggio sono indicati dal thread dell'applicazione principale e dal client MQTT . Il client MQTT annulla il riferimento all'oggetto `MqttMessage` quando la consegna è completata e l'oggetto token di consegna quando il client si disconnette. L'oggetto `MqttMessage` può essere raccolto nel garbage collector dopo il completamento della consegna se l'applicazione client lo annulla. Il token di consegna può essere un raccoglitore dati inutilizzati dopo che la sessione è stata disconnessa.

È possibile ottenere gli attributi `MqttDeliveryToken` e `MqttMessage` dopo la pubblicazione di un messaggio. Se si tenta di impostare gli attributi `MqttMessage` dopo che il messaggio è stato pubblicato, il risultato non è definito.

Il client MQTT continua ad elaborare le conferme di consegna se il cliente si riconnette alla precedente sessione con lo stesso `ClientIdentifier`; consultare "Ripulisci sessioni" a pagina 126. L'applicazione client MQTT deve impostare `MqttClient.CleanSession` su `false` per la sessione precedente e impostarla su `false` nella nuova sessione. Il client MQTT crea nuovi token di consegna e oggetti messaggio nella nuova sessione per le consegne in sospeso. Recupera gli oggetti utilizzando la classe `MqttClientPersistence` . Se il client dell'applicazione ha ancora riferimenti ai vecchi token di consegna e messaggi, annullarli. La richiamata dell'applicazione viene richiamata nella nuova sessione per tutte le distribuzioni avviate nella sessione precedente e completate in questa sessione.

Il callback dell'applicazione viene richiamato dopo la connessione del client dell'applicazione, quando viene completata una consegna in sospeso. Prima che il client dell'applicazione si colleghi, è possibile richiamare le consegne in sospeso utilizzando il metodo `MqttClient.getPendingDeliveryTokens` .

Nota che l'applicazione client ha originariamente creato l'oggetto del messaggio pubblicato e il suo array di byte di payload. Il client MQTT fa riferimento a questi oggetti. L'oggetto messaggio restituito dal token di consegna nel metodo `token.getMessage` non è necessariamente lo stesso oggetto messaggio creato dal client. Se una nuova istanza client MQTT ricrea il token di distribuzione, la classe `MqttClientPersistence` ricrea l'oggetto `MqttMessage` . Per coerenza `token.getMessage` restituisce `null` se `token.isCompleted` è `true`, indipendentemente dal fatto che l'oggetto del messaggio sia stato creato dal client dell'applicazione o dalla classe `MqttClientPersistence` .

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` viene richiamato quando arriva una pubblicazione per il client che corrisponde a un argomento di sottoscrizione. `topic` è l'argomento di pubblicazione, non il filtro di sottoscrizione. I due possono essere diversi se il filtro contiene caratteri jolly.

Se l'argomento corrisponde a più sottoscrizioni create dal client, il client riceve più copie della pubblicazione. Se un client pubblica su un argomento a cui è anche sottoscrittore, riceve una copia della propria pubblicazione.

Se un messaggio viene inviato con un QoS di 1 o 2, il messaggio viene memorizzato dalla classe `MqttClientPersistence` prima che il client MQTT chiami `messageArrived`. `messageArrived` funziona come `deliveryComplete`: viene richiamato una sola volta per una pubblicazione e la copia locale della pubblicazione viene rimossa da `MqttClientPersistence.remove` quando `messageArrived` ritorna al client MQTT . Il client MQTT elimina i riferimenti all'argomento e al messaggio quando `messageArrived` ritorna al client MQTT . L'argomento e gli oggetti del messaggio vengono raccolti nel raccoglitore dati inutilizzati, se il client delle applicazioni non ha mantenuto un riferimento agli oggetti.

Callback, thread e sincronizzazione dell'applicazione client

Il client MQTT richiama un metodo callback su un thread separato al thread dell'applicazione principale. L'applicazione client non crea un thread per il callback, ma viene creata dal client MQTT .

Il client MQTT sincronizza i metodi di callback. Viene eseguita una sola istanza del metodo callback alla volta. La sincronizzazione facilita l'aggiornamento di un oggetto che indica quali pubblicazioni sono state consegnate. Un'istanza di `MqttCallback.deliveryComplete` viene eseguita alla volta, quindi è sicuro aggiornare il conteggio senza ulteriore sincronizzazione. È anche il caso di una sola pubblicazione alla volta. Il codice nel metodo `messageArrived` può aggiornare un oggetto senza sincronizzarlo. Se si fa riferimento al conteggio o all'oggetto che si sta aggiornando, in un altro thread, sincronizzare il conteggio o l'oggetto.

Il token di consegna fornisce un meccanismo di sincronizzazione tra il thread dell'applicazione principale e la distribuzione di una pubblicazione. Il metodo `token.waitForCompletion` attende il completamento della consegna di una pubblicazione specifica o la scadenza di un timeout facoltativo.

È possibile utilizzare `token.waitForCompletion` in un paio di semplici modi per elaborare una pubblicazione alla volta:

1. Per sospendere il client dell'applicazione fino a quando non viene completata la distribuzione della pubblicazione, vedi PubSync.java.
2. Per sincronizzarsi con il metodo `MqttCallback.deliveryComplete` . Solo quando `MqttCallback.deliveryComplete` ritorna al client MQTT `token.waitForCompletion` viene ripreso. Utilizzando questo meccanismo, è possibile sincronizzare il codice in esecuzione in `MqttCallback.deliveryComplete` prima che il codice venga eseguito nel thread dell'applicazione principale.

Cosa fare se si desidera pubblicare senza attendere la consegna di ogni pubblicazione, ma si desidera una conferma quando tutte le pubblicazioni sono state consegnate? Se si esegue la pubblicazione su un singolo thread, l'ultima pubblicazione da inviare è anche l'ultima da consegnare.

Sincronizzazione delle richieste inviate al server

Tabella 4. Comportamento di sincronizzazione dei metodi che risultano in richieste al server.

Questa tabella elenca i metodi nel client Java MQTT che inviano una richiesta al server. Per ciascun metodo, la tabella descrive le condizioni in cui il metodo attende o restituisce e il tempo di attesa del metodo.

Metodo	Sincronizzazione	Intervallo di timeout
`MqttClient.Connect`	Attende che venga stabilita una connessione con il server.	30 secondi per impostazione predefinita o come impostato da un parametro.
`MqttClient.Disconnect`	Attende che il client MQTT termini il lavoro che deve eseguire e che la sessione TCP/IP si scolleghi.	30 secondi per impostazione predefinita o come impostato da un parametro.
`MqttClient.Subscribe`	Attende il completamento della richiesta di sottoscrizione.	30 secondi per impostazione predefinita o come impostato da un parametro.
`MqttClient.UnSubscribe`	Attende il completamento della richiesta di annullamento della sottoscrizione.	30 secondi per impostazione predefinita o come impostato da un parametro.
`MqttClient.Publish`	Ritorna immediatamente al thread dell'applicazione dopo aver passato la richiesta al client MQTT .	Nessuna.
`MqttDeliveryToken.waitForCompletion`	Attende la restituzione del token di consegna.	Indefinito per impostazione predefinita o come impostato da un parametro.

Ripulisci sessioni

Il client MQTT e il servizio di telemetria (MQXR) mantengono le informazioni sullo stato della sessione. Le informazioni sullo stato vengono utilizzate per assicurare "almeno una volta" e "esattamente una volta" la consegna e "esattamente una volta" la ricezione delle pubblicazioni. Lo stato della sessione include anche le sottoscrizioni create da un client MQTT. È possibile scegliere di eseguire un client MQTT con o

senza mantenere le informazioni sullo stato tra le sessioni. Modificare la modalità di pulizia della sessione impostando `MqttConnectOptions.cleanSession` prima della connessione.

Quando connetti un'applicazione client MQTT utilizzando il metodo `MqttClient.connect` , il client identifica la connessione utilizzando l'identificativo client e l'indirizzo del server. Il server verifica se le informazioni sulla sessione sono state salvate da una precedente connessione al server. Se una sessione precedente esiste ancora e `cleanSession=true`, le informazioni sulla sessione precedente sul client e sul server vengono cancellate. Se `cleanSession=false` la sessione precedente viene ripresa. Se non esiste alcuna sessione precedente, viene avviata una nuova sessione.

Nota: L'amministratore WebSphere MQ può chiudere forzatamente una sessione aperta ed eliminare tutte le informazioni sulla sessione. Se il client riapre la sessione con `cleanSession=false`, viene avviata una nuova sessione.

Pubblicazioni

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di collegare il client, tutte le distribuzioni di pubblicazione in sospeso per il client vengono rimosse quando il client si connette.

L'impostazione di ripulitura della sessione non ha alcun effetto sulle pubblicazioni inviate con `QoS=0`. Per `QoS=1` e `QoS=2`, l'utilizzo di `cleanSession=true` potrebbe comportare la perdita di una pubblicazione.

Sottoscrizione

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di connettere il client, tutte le vecchie sottoscrizioni per il client vengono rimosse quando il client si connette. Qualsiasi nuova sottoscrizione effettuata dal client durante la sessione viene rimossa quando si disconnette.

Se si imposta `MqttConnectOptions.cleanSession` su `false` prima della connessione, tutte le sottoscrizioni create dal client vengono aggiunte a tutte le sottoscrizioni esistenti per il client prima della connessione. Tutte le sottoscrizioni restano attive alla disconnessione del client.

Un altro modo di comprendere il modo in cui l'attributo `cleanSession` influenza le sottoscrizioni è considerarlo un attributo modale. Nella sua modalità predefinita, `cleanSession=true`, il client crea sottoscrizioni e riceve pubblicazioni solo nell'ambito della sessione. Nella modalità alternativa, `cleanSession=false`, le sottoscrizioni sono durevoli. Il client può connettersi e disconnettersi e le sue sottoscrizioni rimangono attive. Quando il client si riconnette, riceve tutte le pubblicazioni non consegnate. Mentre è connesso, può modificare l'insieme di sottoscrizioni attive per suo conto.

È necessario impostare la modalità `cleanSession` prima della connessione; la modalità dura per l'intera sessione. Per modificarne l'impostazione, è necessario disconnettere e riconnettere il client. Se si modificano le modalità dall'utilizzo di `cleanSession=false` a `cleanSession=true`, tutte le sottoscrizioni precedenti per il client e tutte le pubblicazioni non ricevute vengono eliminate.

Identificativo client

L'identificativo client è una stringa di 23 byte che identifica un client MQTT . L'identificativo client deve essere univoco tra tutti i client che si connettono al server e non deve essere uguale al nome del gestore code sul server. All'interno di questi vincoli, è possibile utilizzare qualsiasi stringa di identificazione. È importante disporre di una procedura per l'assegnazione di identificativi client e di un mezzo per la configurazione di un client con l'identificativo scelto.

L'identificativo client viene utilizzato nella gestione di un sistema MQTT . Con potenzialmente centinaia di migliaia di clienti da amministrare, è necessario essere in grado di identificare rapidamente un particolare cliente. Si supponga, ad esempio, che un dispositivo non abbia funzionato correttamente e che l'utente riceva una notifica, forse da un cliente che squilla un help desk. Come il cliente identifica il dispositivo e come si correla tale identificazione con il server che di solito è connesso al client? È necessario consultare un database che associa ogni periferica a un identificativo client e a un server? Il nome della periferica identifica il server a cui è collegata? Quando si sfogliano le connessioni client MQTT , ogni

connessione viene etichettata con l'identificativo client. È necessario ricercare una tabella per associare un identificativo client a una periferica fisica?

L'identificativo del client identifica una particolare periferica, un utente o un'applicazione in esecuzione sul client? Se un cliente sostituisce un dispositivo difettoso con uno nuovo, il nuovo dispositivo ha lo stesso identificativo del vecchio dispositivo? Assegnare un nuovo identificatore? Se si modifica un dispositivo fisico, ma si conserva lo stesso identificativo, le pubblicazioni in sospeso e le sottoscrizioni attive vengono automaticamente trasferite al nuovo dispositivo.

Come si garantisce che gli identificatori client siano univoci? Oltre a un sistema per la generazione di identificativi univoci, è necessario disporre di un processo affidabile per l'impostazione dell'identificativo sul client. Forse il dispositivo client è una "black-box", senza interfaccia utente. Si produce il dispositivo con un identificativo client, ad esempio utilizzando il relativo indirizzo MAC? Oppure si dispone di un processo di installazione e configurazione del software che configura il dispositivo prima che venga attivato?

È possibile creare un identificativo client dall'indirizzo MAC della periferica a 48 bit, per mantenere l'identificativo breve e univoco. Se la dimensione di trasmissione non è un problema critico, è possibile utilizzare i restanti 17 byte per semplificare l'amministrazione dell'indirizzo.

Token di consegna

Quando un client pubblica un argomento, viene creato un nuovo token di consegna. Utilizzare il token di consegna per monitorare la distribuzione di una pubblicazione o per bloccare l'app client fino al completamento della distribuzione.

Il token è un oggetto `MqttDeliveryToken` . Viene creato richiamando il metodo MqttTopic.publish () e viene conservato dal client MQTT fino a quando la sessione client non viene disconnessa e la consegna non viene completata.

Il normale utilizzo del token è quello di verificare se la consegna è completa. Blocca l'applicazione client fino al completamento della distribuzione utilizzando il token restituito per richiamare `token.waitForCompletion`. In alternativa, fornire un handler `MqttCallBack` . Quando il client MQTT ha ricevuto tutti i riconoscimenti previsti come parte della consegna della pubblicazione, richiama `MqttCallBack.deliveryComplete` passando il token di consegna come parametro.

Fino a quando la consegna non è completa, puoi ispezionare la pubblicazione utilizzando il token di consegna restituito chiamando `token.getMessage`.

Consegne completate

Il completamento delle consegne è asincrono e dipende dalla qualità del servizio associato alla pubblicazione.

Al massimo una volta

`QoS=0`

La consegna è completa immediatamente al ritorno da `MqttTopic.publish`. `MqttCallback.deliveryComplete` viene richiamato immediatamente.

Almeno una volta

`QoS=1`

Il recapito è completo quando è stato ricevuto un riconoscimento alla pubblicazione dal gestore code. `MqttCallback.deliveryComplete` viene richiamato quando viene ricevuto il riconoscimento. Il messaggio potrebbe essere consegnato più di una volta prima che venga richiamato `MqttCallback.deliveryComplete` , se le comunicazioni sono lente o inaffidabili.

Esattamente una volta

`QoS=2`

La consegna è completa quando il client riceve un messaggio di completamento che indica che la pubblicazione è stata pubblicata per i sottoscrittori. `MqttCallback.deliveryComplete` viene

richiamato non appena viene ricevuto il messaggio di pubblicazione. Non attende il messaggio di completamento.

In rare circostanze, la tua applicazione client potrebbe non tornare al client MQTT da `MqttCallback.deliveryComplete` normalmente. Sai che la consegna è stata completata, perché `MqttCallback.deliveryComplete` è stato chiamato. Se il client riavvia la stessa sessione, `MqttCallback.deliveryComplete` non viene richiamato di nuovo.

Consegne incomplete

Se la distribuzione non è completa dopo la disconnessione della sessione client, è possibile connettere nuovamente il client e completare la distribuzione. È possibile completare la consegna di un messaggio solo se il messaggio è stato pubblicato in una sessione con l'attributo `MqttConnectionOptions` impostato su `false`.

Creare il client utilizzando lo stesso identificativo client e lo stesso indirizzo server, quindi connettersi, impostando nuovamente l'attributo `cleanSession` `MqttConnectionOptions` su `false` . Se si imposta `cleanSession` su `true`, i token di consegna in sospeso vengono eliminati.

È possibile verificare se sono presenti recapiti in sospeso chiamando `MqttClient.getPendingDeliveryTokens`. È possibile chiamare `MqttClient.getPendingDeliveryTokens` prima di collegare il client.

Ultimo testamento e pubblicazione testamentaria

Se una connessione client MQTT termina in modo imprevisto, è possibile configurare WebSphere MQ Telemetry per inviare una pubblicazione "Last Will and Testament". Predefinire il contenuto della pubblicazione e l'argomento a cui inviarla. L'"ultima volontà e testamento" è una proprietà di connessione. Crearlo prima di collegare il cliente.

Creare un argomento per l'ultimo testamento. È possibile creare un argomento come `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Impostare un "ultimo testamento" utilizzando il metodo `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` .

Considerare la creazione di una data / ora nel messaggio `lastWillPayload` . Includere altre informazioni sul client che consentono di identificare il client e le circostanze della connessione. Passare l'oggetto `MqttConnectionOptions` al costruttore `MqttClient` .

Impostare `lastWillQos` su 1 o 2, per rendere il messaggio persistente in IBM WebSphere MQe per garantire la consegna. Per conservare le informazioni sull'ultima connessione persa, impostare `lastWillRetained` su `true`.

La pubblicazione "Last Will and Testament" viene inviata ai sottoscrittori se la connessione termina in modo imprevisto. Viene inviato se la connessione termina senza che il client chiami il metodo `MqttClient.disconnect` .

Per monitorare le connessioni, completare la pubblicazione "Last Will and Testament" con altre pubblicazioni per registrare le connessioni e le disconnessioni programmate.

Persistenza del messaggio nei client MQTT

I messaggi di pubblicazione vengono resi persistenti se vengono inviati con una QoS (quality of service) di "almeno una volta" o "esattamente una volta". È possibile implementare il proprio meccanismo di persistenza sul client oppure utilizzare il meccanismo di persistenza predefinito fornito con il client. La persistenza funziona in entrambe le direzioni, per le pubblicazioni inviate al o dal client.

In MQTT, la persistenza del messaggio ha due aspetti: il modo in cui il messaggio viene trasferito e se viene accodato nel server MQTT come un messaggio persistente.

1. Il client MQTT accoppia la persistenza del messaggio con la QoS (quality of service). A seconda della qualità del servizio scelta per un messaggio, il messaggio viene reso persistente. La persistenza del messaggio è necessaria per implementare la QoS (quality of service) richiesta.

   Se si specifica "al massimo una volta", `QoS=0`, il client elimina il messaggio non appena viene pubblicato. Se si verifica un errore nell'elaborazione upstream del messaggio, il messaggio non viene inviato di nuovo. Anche se il client rimane attivo, il messaggio non viene inviato di nuovo. Il funzionamento dei messaggi `QoS=0` è lo stesso dei messaggi non persistenti veloci IBM WebSphere MQ .

   Se un messaggio viene pubblicato da un client con QoS di 1 o 2, viene reso persistente. Il messaggio viene memorizzato localmente e scartato dal client solo quando non è più necessario per garantire la consegna "almeno una volta", `QoS=1`o "esattamente una volta", `QoS=2`.
2. Se un messaggio è contrassegnato come QoS 1 o 2, viene accodato come un messaggio persistente. Se è contrassegnato come `QoS=0`, viene accodato come messaggio non persistente. In IBM WebSphere MQ i messaggi non persistenti vengono trasferiti tra gestori code "esattamente una volta", a meno che il canale dei messaggi non abbia l'attributo `NPMSPEED` impostato su FAST.

Una pubblicazione persistente viene memorizzata sul cliente fino a quando non viene ricevuta da un'applicazione client. Per `QoS=2`, la pubblicazione viene eliminata dal client quando il callback dell'applicazione restituisce il controllo. Per `QoS=1` l'applicazione potrebbe ricevere nuovamente la pubblicazione, se si verifica un errore. Per `QoS=0`, il callback riceve la pubblicazione non più di una volta. Potrebbe non ricevere la pubblicazione se si verifica un errore o se il client è disconnesso al momento della pubblicazione.

Quando si sottoscrive un topic, è possibile ridurre il QoS con cui il sottoscrittore riceve i messaggi in modo che corrisponda alle sue capacità di persistenza. Le pubblicazioni create in un QoS superiore vengono inviate con il QoS più elevato richiesto dal sottoscrittore.

Memorizzazione dei messaggi

L'implementazione dell'archiviazione dei dati su piccoli dispositivi varia molto. Il modello di salvataggio temporaneo dei messaggi persistenti nell'archivio gestito dal client MQTT potrebbe essere troppo lento o richiedere troppa memoria. Nei dispositivi mobili, il sistema operativo mobile potrebbe fornire un servizio di archiviazione ideale per i messaggi MQTT .

Per fornire flessibilità nel soddisfare i vincoli delle periferiche di piccole dimensioni, il client MQTT ha due interfacce di persistenza. Le interfacce definiscono le operazioni coinvolte nella memorizzazione dei messaggi persistenti. Le interfacce sono descritte nella documentazione API per Client MQTT per Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT. È possibile implementare le interfacce per adattarsi a una periferica. Il client MQTT in esecuzione su Java SE ha un'implementazione predefinita delle interfacce che memorizzano i messaggi persistenti nel filesystem. Utilizza il pacchetto `java.io` . Il client dispone anche di una implementazione predefinita per Java ME, `MqttDefaultMIDPPersistence`.

Classi di persistenza

`MqttClientPersistence`
: Passare un'istanza dell'implementazione di `MqttClientPersistence` al client MQTT come parametro del costruttore `MqttClient` . Se si omette il parametro `MqttClientPersistence` dal costruttore `MqttClient` , il client MQTT memorizza i messaggi persistenti utilizzando la classe `MqttDefaultFilePersistence` o `MqttDefaultMIDPPersistence`.

`MqttPersistable`
: `MqttClientPersistence` ottiene e inserisce `MqttPersistable` oggetti utilizzando una chiave di memoria. È necessario fornire un'implementazione di `MqttPersistable` e l'implementazione di `MqttClientPersistence` se non si utilizza `MqttDefaultFilePersistence` o `MqttDefaultMIDPPersistence`.

MqttDefaultFilePersistence

Il client MQTT fornisce la classe MqttDefaultFilePersistence . Se crei un'istanziazione di MqttDefaultFilePersistence nella tua applicazione client, puoi fornire la directory per memorizzare i messaggi persistenti come un parametro del costruttore MqttDefaultFilePersistence .

In alternativa, il client MQTT può creare un'istanza di MqttDefaultFilePersistence e inserire i file in una directory predefinita. Il nome della directory è *client identifier*-tcp *hostname portnumber*. "\", "\\", "/", ":" e " " vengono rimossi dalla stringa del nome directory.

Il percorso della directory è il valore della proprietà di sistema rcp.data. Se rcp.data non è impostato, il percorso è il valore della proprietà di sistema usr.data.

rcp.data è una proprietà associata all'installazione di OSGi o Eclipse Rich Client Platform (RCP).

usr.data è la directory in cui è stato avviato il comando Java che ha avviato l'applicazione.

MqttDefaultMIDPPersistence

MqttDefaultMIDPPersistence ha un costruttore predefinito e nessun parametro. Utilizza il package javax.microedition.rms.RecordStore per memorizzare i messaggi.

Pubblicazioni

Le pubblicazioni sono istanze di MqttMessage associate a una stringa di argomento. Il client MQTT può creare pubblicazioni da inviare a IBM WebSphere MQe sottoscrivere argomenti su IBM WebSphere MQ MQ per ricevere pubblicazioni.

Un MqttMessage ha un array di byte come payload. Cercare di mantenere i messaggi il più piccoli possibile. La lunghezza massima del messaggio consentita dal protocollo di MQTT è 250 MB.

Di solito, un programma client MQTT utilizza java.lang.String o java.lang.StringBuffer per manipolare il contenuto del messaggio. Per comodità, la classe MqttMessage ha un metodo toString per convertire il suo payload in una stringa. Per creare il payload dell'array di byte da un java.lang.String o java.lang.StringBuffer, utilizzare il metodo getBytes .

Il metodo getBytes converte una stringa nella serie di caratteri predefinita per la piattaforma. La serie di caratteri predefinita è generalmente UTF-8. Le pubblicazioni MQTT che contengono solo testo sono generalmente codificate in UTF-8. Utilizzare il metodo getBytes("UTF8") per sovrascrivere la serie di caratteri predefinita.

In IBM WebSphere MQ, una pubblicazione MQTT viene ricevuta come messaggio jms - bytes . Il messaggio include una cartella MQRFH2 contenente una cartella <mqtt>e una <mqps> . La cartella <mqtt> contiene clientId e qos, ma questo contenuto potrebbe cambiare in futuro.

Un MqttMessage ha tre attributi aggiuntivi: QoS (quality of service), se è conservato e se è un duplicato. L'indicatore duplicato viene impostato solo se la qualità del servizio è "almeno una volta" o "esattamente una volta". Se il messaggio è stato inviato in precedenza e non è stato riconosciuto abbastanza rapidamente dal client MQTT , il messaggio viene inviato di nuovo, con l'attributo duplicato impostato su true.

Pubblicazione

Per creare una pubblicazione in un'applicazione client MQTT , crea un MqttMessage. Imposta il payload, la qualità del servizio e se viene conservato e richiama il metodo MqttTopic.publish(MqttMessage message) ; viene restituito MqttDeliveryToken e il completamento della pubblicazione è asincrono.

In alternativa, il client MQTT può creare un oggetto messaggio temporaneo dai parametri sul metodo MqttTopic.publish(byte [] payload, int qos, boolean retained) quando crea una pubblicazione.

Se la pubblicazione ha una qualità del servizio "almeno una volta" o "esattamente una volta", QoS=1 o QoS=2, il client MQTT richiama l'interfaccia MqttClientPersistence . Richiama

MqttClientPersistence per memorizzare il messaggio prima di restituire un token di consegna all'applicazione.

L'applicazione può scegliere di bloccare fino a quando il messaggio non viene consegnato al server, utilizzando il metodo MqttDeliveryToken.waitForCompletion . In alternativa, l'applicazione può continuare senza bloccare. Se si desidera controllare se le pubblicazioni vengono consegnate, senza bloccare, registrare un'istanza di una classe di callback che implementa MqttCallback con il client MQTT . Il client MQTT richiama il metodo MqttCallback.deliveryComplete non appena la pubblicazione è stata consegnata. A seconda della qualità del servizio, la distribuzione potrebbe essere quasi immediata per QoS=0o potrebbe richiedere del tempo per QoS=2.

Utilizzare il metodo MqttDeliveryToken.isComplete per eseguire il polling se la consegna è completa. Mentre il valore di MqttDeliveryToken.isComplete è false, è possibile richiamare MqttDeliveryToken.getMessage per ottenere il contenuto del messaggio. Se il risultato della chiamata MqttDeliveryToken.isComplete è true, il messaggio è stato eliminato e la chiamata MqttDeliveryToken.getMessage genera un'eccezione di puntatore null. Non esiste alcuna sincronizzazione integrata tra MqttDeliveryToken.getMessage e MqttDeliveryToken.isComplete.

Se il client si disconnette prima di ricevere tutti i token di consegna in sospeso, una nuova istanza del client può interrogare i token di consegna in sospeso prima della connessione. Fino a quando il client non si connette, non vengono completate nuove consegne ed è sicuro chiamare MqttDeliveryToken.getMessage. Utilizzare il metodo MqttDeliveryToken.getMessage per scoprire quali pubblicazioni non sono state consegnate. I token di consegna in sospeso vengono eliminati se ci si connette con MqttConnectOptions.cleanSession impostato sul valore predefinito, true.

Sottoscrizione

Un gestore code o IBM MessageSight è responsabile della creazione di pubblicazioni da inviare a un sottoscrittore MQTT . Il gestore code verifica se il filtro argomenti in una sottoscrizione creata da un client MQTT corrisponde alla stringa argomenti in una pubblicazione. La corrispondenza può essere una corrispondenza esatta oppure può includere caratteri jolly. Prima che la pubblicazione venga inoltrata al sottoscrittore dal gestore code, il gestore code controlla gli attributi argomento associati alla pubblicazione. Segue la procedura di ricerca descritta in Sottoscrizione mediante una stringa di argomenti contenente caratteri jolly per identificare se un oggetto argomento di gestione concede all'utente l'autorizzazione alla sottoscrizione.

Quando il client MQTT riceve una pubblicazione con QoS (quality of service) "almeno una volta", richiama il metodo MqttCallback.messageArrived per elaborare la pubblicazione. Se la qualità del servizio della pubblicazione è "esattamente una volta", QoS=2, il client MQTT richiama l'interfaccia MqttClientPersistence per memorizzare il messaggio quando viene ricevuto. Viene quindi chiamato MqttCallback.messageArrived.

Qualità del servizio fornito da un client MQTT

Un client MQTT fornisce tre QoS (quality of service) per la distribuzione di pubblicazioni a WebSphere MQ e al client MQTT : "al più una volta", "almeno una volta" e "esattamente una volta". Quando un client MQTT invia una richiesta a IBM WebSphere MQ per creare una sottoscrizione, la richiesta viene inviata con la qualità del servizio "almeno una volta".

La qualità del servizio di una pubblicazione è un attributo di MqttMessage. È impostato dal metodo MqttMessage.setQos.

Il metodo MqttClient.subscribe può ridurre la qualità del servizio applicato alle pubblicazioni inviate a un client su un argomento. La qualità del servizio di una pubblicazione inoltrata a un sottoscrittore potrebbe essere diversa dalla qualità del servizio della pubblicazione. Il valore inferiore dei due valori viene utilizzato per inoltrare una pubblicazione.

Al massimo una volta
QoS=0

Il messaggio viene consegnato al massimo una volta oppure non viene consegnato. Il suo recapito sulla rete non è riconosciuto.

Il messaggio non è memorizzato. Se il client è disconnesso o se si verifica un errore nel server, il messaggio potrebbe andare perso.

`QoS=0` è la modalità di trasferimento più veloce. A volte si chiama "fuoco e dimenticate".

Il protocollo MQTT non richiede che i server inoltrino le pubblicazioni su `QoS=0` a un client. Se il client è disconnesso nel momento in cui il server riceve la pubblicazione, la pubblicazione potrebbe essere eliminata, a seconda del server. Il servizio di telemetria (MQXR) non elimina i messaggi inviati con `QoS=0`. Vengono memorizzati come messaggi non persistenti e vengono eliminati solo se il gestore code viene arrestato.

Almeno una volta

`QoS=1`

`QoS=1` è la modalità predefinita di trasferimento.

Il messaggio viene sempre consegnato almeno una volta. Se il mittente non riceve un riconoscimento, il messaggio viene inviato di nuovo con l'indicatore DUP impostato fino a quando non viene ricevuto un riconoscimento. Come risultato, il destinatario può essere inviato lo stesso messaggio più volte e potrebbe elaborarlo più volte.

Il messaggio deve essere memorizzato localmente sul mittente e sul destinatario fino a quando non viene elaborato.

Il messaggio viene eliminato dal destinatario dopo che questo ha elaborato il messaggio. Se il destinatario è un broker, il messaggio viene pubblicato per i relativi sottoscrittori. Se il destinatario è un client, il messaggio viene consegnato all'applicazione del sottoscrittore. Una volta eliminato il messaggio, il destinatario invia un riconoscimento al mittente.

Il messaggio viene eliminato dal mittente dopo aver ricevuto un riconoscimento dal destinatario.

Esattamente una volta

`QoS=2`

Il messaggio viene sempre consegnato esattamente una volta.

Il messaggio deve essere memorizzato localmente sul mittente e sul destinatario fino a quando non viene elaborato.

`QoS=2` è la modalità di trasferimento più sicura ma più lenta. Richiede almeno due coppie di trasmissioni tra il mittente e il ricevente prima che il messaggio venga eliminato dal mittente. Il messaggio può essere elaborato dal destinatario dopo la prima trasmissione.

Nella prima coppia di trasmissioni, il mittente trasmette il messaggio e riceve il riconoscimento dal destinatario che ha memorizzato il messaggio. Se il mittente non riceve un riconoscimento, il messaggio viene inviato di nuovo con l'indicatore DUP impostato fino a quando non viene ricevuto un riconoscimento.

Nella seconda coppia di trasmissioni, il mittente comunica al ricevente che può completare l'elaborazione del messaggio, `"PUBREL"`. Se il mittente non riceve un riconoscimento del messaggio `"PUBREL"` , il messaggio `"PUBREL"` viene inviato di nuovo fino a quando non viene ricevuto un riconoscimento. Il mittente elimina il messaggio salvato quando riceve il riconoscimento al messaggio `"PUBREL"`

Il destinatario può elaborare il messaggio nella prima o nella seconda fase, purché non rielabori il messaggio. Se il destinatario è un broker, pubblica il messaggio ai sottoscrittori. Se il destinatario è un client, consegna il messaggio all'applicazione del sottoscrittore. Il destinatario invia un messaggio di completamento al mittente che ha terminato l'elaborazione del messaggio.

Pubblicazioni conservate e client MQTT

Se si crea una sottoscrizione a un argomento che ha una pubblicazione conservata, la pubblicazione conservata più recente sull'argomento viene immediatamente inoltrata all'utente.

Utilizzare il metodo `MqttMessage.setRetained` per specificare se una pubblicazione su un argomento viene conservata o meno.

Per eliminare una pubblicazione conservata in IBM WebSphere MQ, eseguire il comando MQSC CLEAR TOPICSTR**CLEAR TOPICSTR** .

Se si crea una pubblicazione con un payload null, la pubblicazione vuota viene inoltrata ai sottoscrittori. Altri broker MQTT potrebbero non inoltrare una pubblicazione vuota ai sottoscrittori.

Se si pubblica una pubblicazione non conservata in un argomento che ha una pubblicazione conservata, la pubblicazione conservata non viene influenzata. I sottoscrittori correnti ricevono la nuova pubblicazione. I nuovi sottoscrittori ricevono prima la pubblicazione conservata, quindi le nuove pubblicazioni.

Quando si crea o si aggiorna una pubblicazione conservata, inviare la pubblicazione con un QoS o 1 o 2. Se lo si invia con un QoS pari a 0, IBM WebSphere MQ crea una pubblicazione conservata non persistente. La pubblicazione non viene conservata se il gestore code viene arrestato.

Utilizzare le pubblicazioni conservate per registrare l'ultimo valore di una misurazione. I nuovi sottoscrittori all'argomento conservato ricevono immediatamente il valore più recente della misurazione. Se non viene eseguita alcuna nuova misurazione dall'ultima sottoscrizione del sottoscrittore all'argomento della pubblicazione e se il sottoscrittore effettua nuovamente la sottoscrizione, il sottoscrittore riceve nuovamente la pubblicazione conservata più recente sull'argomento.

Sottoscrizione

Creare sottoscrizioni per registrare un interesse negli argomenti di pubblicazione utilizzando un filtro argomenti. Un client può creare più sottoscrizioni o una sottoscrizione contenente un filtro argomenti che utilizza caratteri jolly, per registrare un interesse in più argomenti. Le pubblicazioni sugli argomenti corrispondenti ai filtri vengono inviati al client. Le sottoscrizioni possono rimanere attive mentre un client è disconnesso. Le pubblicazioni vengono inviate al client quando si riconnette.

Creare sottoscrizioni utilizzando i metodi `MqttClient.subscribe` , passando uno o più filtri argomento e parametri QoS (quality of service). Il parametro QoS (quality of service) imposta la qualità massima del servizio che il sottoscrittore è pronto a utilizzare per ricevere un messaggio. I messaggi inviati a questo client non possono essere consegnati con una QoS (quality of service) superiore. La QoS (quality of service) è impostata sul valore inferiore del valore originale quando il messaggio è stato pubblicato e sul livello specificato per la sottoscrizione. La qualità del servizio predefinita per la ricezione dei messaggi è `QoS=1`, almeno una volta.

La richiesta di sottoscrizione viene inviata con `QoS=1`.

Le pubblicazioni vengono ricevute da un sottoscrittore quando il client MQTT richiama il metodo `MqttCallback.messageArrived` . Il metodo `messageArrived` inoltra anche la stringa di argomenti con cui il messaggio è stato pubblicato al sottoscrittore.

È possibile rimuovere una sottoscrizione o una serie o sottoscrizioni utilizzando i metodi `MqttClient.unsubscribe` .

Un comando WebSphere MQ può rimuovere una sottoscrizione. Elencare le sottoscrizioni utilizzando WebSphere MQ Explorer oppure utilizzando i comandi **runmqsc** o PCF. Tutte le sottoscrizioni dei client MQTT vengono denominate. Viene fornito un nome del formato: *`ClientIdentifier`*:*`Topic name`*

Se si utilizza il valore predefinito `MqttConnectOptions`o si imposta `MqttConnectOptions.cleanSession` su `true` prima di connettere il client, tutte le vecchie sottoscrizioni per il client vengono rimosse quando il client si connette. Qualsiasi nuova sottoscrizione effettuata dal client durante la sessione viene rimossa quando si disconnette.

Se si imposta `MqttConnectOptions.cleanSession` su `false` prima della connessione, tutte le sottoscrizioni create dal client vengono aggiunte a tutte le sottoscrizioni esistenti per il client prima della connessione. Tutte le sottoscrizioni restano attive alla disconnessione del client.

Un altro modo di comprendere il modo in cui l'attributo `cleanSession` influenza le sottoscrizioni è considerarlo un attributo modale. Nella sua modalità predefinita, `cleanSession=true`, il client crea sottoscrizioni e riceve pubblicazioni solo nell'ambito della sessione. Nella modalità alternativa, `cleanSession=false`, le sottoscrizioni sono durevoli. Il client può connettersi e disconnettersi e le sue sottoscrizioni rimangono attive. Quando il client si riconnette, riceve tutte le pubblicazioni non consegnate. Mentre è connesso, può modificare l'insieme di sottoscrizioni attive per suo conto.

È necessario impostare la modalità `cleanSession` prima della connessione; la modalità dura per l'intera sessione. Per modificarne l'impostazione, è necessario disconnettere e riconnettere il client. Se si modificano le modalità dall'utilizzo di `cleanSession=false` a `cleanSession=true`, tutte le sottoscrizioni precedenti per il client e tutte le pubblicazioni non ricevute vengono eliminate.

Le pubblicazioni che corrispondono alle sottoscrizioni attive vengono inviate al client non appena vengono pubblicate. Se il client è disconnesso, vengono inviati al client se si riconnette allo stesso server con lo stesso identificativo client e `MqttConnectOptions.cleanSession` impostato su `false`.

Le sottoscrizioni per un particolare cliente vengono identificate dall'identificativo client. È possibile riconnettere il client da un dispositivo client differente allo stesso server e continuare con la stessa sottoscrizione e ricevere pubblicazioni non consegnate.

Stringhe argomento e filtri argomento nei client MQTT

Le stringhe argomento e filtri argomento vengono utilizzati per la pubblicazione e la sottoscrizione. La sintassi delle stringhe argomento e dei filtri nei client MQTT è in gran parte la stessa delle stringhe argomento in IBM WebSphere MQ.

Le stringhe di argomenti vengono utilizzate per inviare pubblicazioni ai sottoscrittori. Creare una stringa di argomenti utilizzando il metodo `MqttClient.getTopic(java.lang.String topicString)`.

I filtri argomento vengono utilizzati per sottoscrivere argomenti e ricevere pubblicazioni. I filtri argomento possono contenere caratteri jolly. Con i caratteri jolly, è possibile sottoscrivere più argomenti. Creare un filtro argomenti utilizzando un metodo di sottoscrizione; ad esempio, `MqttClient.subscribe(java.lang.String topicFilter)`.

Stringhe argomento

La sintassi di una stringa di argomenti IBM WebSphere MQ è descritta in Stringhe di argomenti. La sintassi delle stringhe di argomenti MQTT è descritta nella classe `MqttClient` nella documentazione API per Client MQTT per Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT..

La sintassi di ciascun tipo di stringa argomento è quasi identica. Ci sono quattro differenze minori:

1. Le stringhe degli argomenti inviate a IBM WebSphere MQ dai client MQTT devono seguire le convenzioni per i nomi dei gestori code. In particolare, non le stringhe di argomenti non possono contenere trattini.
2. Le lunghezze massime differiscono. Le stringhe di argomenti IBM WebSphere MQ sono limitate a 10.240 caratteri. Un client MQTT può creare stringhe di argomenti fino a un massimo di 65535 byte.
3. Una stringa argomento creata da un client MQTT non può contenere un carattere null.
4. In WebSphere Message Broker, un livello di argomento null, `'...//...'` non era valido. I livelli di argomento null sono supportati da IBM WebSphere MQ.

A differenza della pubblicazione / sottoscrizione IBM WebSphere MQ , il protocollo `mqttv3` non ha un concetto di oggetto argomento di gestione. Non è possibile costruire una stringa argomento da un oggetto argomento e una stringa argomento. Tuttavia, una stringa di argomenti viene associata a un argomento di amministrazione in WebSphere MQ. Il controllo accessi associato all'argomento di gestione determina se una pubblicazione viene pubblicata nell'argomento o scartata. Gli attributi che vengono applicati a una pubblicazione quando viene inoltrata ai sottoscrittori, sono influenzati dagli attributi dell'argomento di gestione.

Filtri argomento

La sintassi di un filtro argomento IBM WebSphere MQ è descritta in Schema dei caratteri jolly basati sugli argomenti. La sintassi dei filtri argomento che puoi creare con un client MQTT è descritta nella classe `MqttClient` nella documentazione API per Client MQTT per Java. Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

La sintassi di ciascun tipo di filtro argomento è quasi identica. L'unica differenza sta nel modo in cui i broker MQTT interpretano un filtro argomento. In WebSphere Message Broker V6, un carattere jolly multilivello può essere utilizzato solo alla fine di un filtro argomenti. In WebSphere MQ, è possibile utilizzare un carattere jolly multilivello a qualsiasi livello nella struttura ad albero degli argomenti, ad esempio `USA/#/Dutchess County`.

Riferimento alla programmazione di client MQTT

Di seguito sono riportati i collegamenti a Mobile Messaging and M2M Pacchetto cliente alla documentazione dell'API client associata.

In Mobile Messaging and M2M Pacchetto client, le librerie client MQTT sono fornite con la documentazione API generata. È possibile scaricare il pacchetto client da IBM messaging community downloads.

Puoi visualizzare le copie in linea della documentazione API più recente seguendo questi collegamenti al progetto Eclipse Paho :

- Client MQTT per classi Java
- Libreria client MQTT per C
- Libreria client MQTT asincrona per C

Nota:

1. Collegare le applicazioni MQTT Java al package `org.eclipse.paho.client.mqttv3` piuttosto che a `com.ibm.micro.client.mqttv3`. esistente. Il pacchetto `com.ibm.micro.client.mqttv3` viene fornito per supportare le applicazioni MQTT Java esistenti.
2. V 7.5.0.1 Collegare le applicazioni client MQTT per la libreria C alla libreria `MQTTAsync` piuttosto che alla libreria `MQTTClient` . `MQTTClient` viene fornito per supportare le applicazioni MQTT esistenti per C.
3. MQTT messaging client per JavaScript richiede un server MQTT che supporti WebSockets. Ad esempio, IBM WebSphere MQ Version 7.5 e versioni successive lo fanno.

Introduzione ai server MQTT

I server di messaggistica che supportano il protocollo di trasporto MQTT sono disponibili da IBM e altri. Il server MQTT di base consente alle app e ai dispositivi mobili, supportati dalle librerie client MQTT , di scambiare messaggi. IBM WebSphere MQ e IBM MessageSight sono server MQTT di IBM. Oltre a fungere da server MQTT di base, scambiano anche i messaggi tra le applicazioni client MQTT e le applicazioni aziendali. Tutti i server MQTT da IBM supportano il protocollo MQTT version 3.1 e MQTT su WebSocket protocol.

Server MQTT correnti da IBM

IBM WebSphere MQ

- IBM WebSphere MQ fornisce la messaggistica di livello aziendale. Il componente di telemetria consente a IBM WebSphere MQ di agire anche come server MQTT .
- Ciò supporta le tue applicazioni mobili, M2M (machine-to-machine) e basate sui dispositivi e consente loro di scambiare messaggi con le app di messaggistica aziendale come le app IBM WebSphere MQ e JMS .
- L'installazione di IBM WebSphere MQ include una copia dell'SDK MQTT da IBM. Questo SDK fornisce applicazioni client MQTT di esempio e librerie client MQTT che supportano queste applicazioni.

  Nota: Per ottenere la versione più aggiornata di questo SDK, scarica Mobile Messaging and M2M Pacchetto client. Per ulteriori informazioni, vedere “Introduzione ai client MQTT” a pagina 11.

- Il supporto MQTT è stato incluso per la prima volta in IBM WebSphere MQ Version 7.0.1. Per informazioni complete per ogni release di IBM WebSphere MQ, consultare la seguente documentazione del prodotto:
  - WebSphere MQ Telemetry Versione 7.5
  - WebSphere MQ Telemetry Versione 7.1

  Per una breve introduzione a IBM WebSphere MQe la procedura introduttiva al componente IBM WebSphere MQ Telemetry , consultare “IBM WebSphere MQ come server MQTT” a pagina 138.

IBM MessageSight

- IBM MessageSight è un server MQTT basato su appliance che può connettersi a un numero elevato di client MQTT allo stesso tempo e fornire le prestazioni e la scalabilità necessarie per soddisfare la crescente quantità di sensori e dispositivi mobili. Supporta il protocollo MQTT version 3.1 e MQTT su WebSocket protocol

.

- Le funzioni e i vantaggi principali di IBM MessageSight come server MQTT sono i seguenti:
  - Prestazioni elevate, affidabilità e messaggistica scalabile.
  - Progettato specificamente per scenari machine - to - machine (M2M) e Internet of Things, supportando comunità di grandi dimensioni per endpoint connessi contemporaneamente.
  - Facilità di installazione e di utilizzo. Può essere attivo e in esecuzione in meno di 30 minuti.
  - Supporto per le applicazioni mobili native che includono Android e iOS.
  - Integrazione con IBM WebSphere MQ come broker di pubblicazione / sottoscrizione.
- Per una rapida introduzione a IBM MessageSight, vedi l'introduzione di MessageSight su YouTube e l'annuncio MessageSight. Per informazioni tecniche dettagliate, consultare la documentazione del prodotto MessageSight.

IBM WebSphere MQ Telemetry daemon for devices

- Questo è noto anche come IBM WebSphere MQ Telemetry advanced client for C. Si tratta di un piccolo server MQTT di ingombro che generalmente viene eseguito in ubicazioni satellitari o in dispositivi vicini al bordo della rete; ad esempio in set-top box, in unità di telemetria remote o in terminali di punto vendita.
- Un uso tipico è quello di concentrare molte connessioni client MQTT , che vengono quindi connesse a IBM WebSphere MQ su Internet in una singola connessione MQTT . Ad esempio, è possibile installare un numero elevato di sensori in un edificio, collegarli a IBM WebSphere MQ Telemetry daemon for devicese connettere il daemon a IBM WebSphere MQ.
- Il IBM WebSphere MQ Telemetry daemon for devices è incluso con IBM WebSphere MQ. È richiesta una licenza separata per collegarla a IBM WebSphere MQ. Consultare IBM United States Software Announcement 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) è una versione di IBM WebSphere MQ Telemetry daemon for devices. La differenza principale è nell'uso. RSMB è un server di test di piccole dimensioni, disponibile presso IBM alphaWorkse destinato ad essere utilizzato durante la valutazione o la sperimentazione di soluzioni basate su MQTT. RSMB supporta MQTT su diverse piatteforme Linux , su Windows XP, su Apple Mac OS X Leoparde su Unslung (Linksys NSLU12)

Server MQTT precedenti da IBM

WebSphere Message Broker ***(ora noto come IBM Integration Bus)***

- WebSphere Message Broker Versione 6 ha fornito il proprio server MQTT . Il supporto è stato sostituito in WebSphere Message Broker Versione 7 dal componente di telemetria di IBM WebSphere MQ.

Altri server MQTT

MQTT.org conserva un elenco di server e broker MQTT nella pagina Software , inclusi i server open source.

Attività correlate

“Introduzione ai client MQTT” a pagina 11
Puoi iniziare a sviluppare un'applicazione mobile o M2M (machine-to-machine) creando ed eseguendo un'applicazione client MQTT di esempio che utilizza una libreria client MQTT . Le applicazioni di esempio e librerie client associate sono disponibili in Mobile Messaging and M2M Pacchetto client da IBM. Ci sono versioni delle app e delle librerie client scritte in Java, in JavaScripte in C. Puoi eseguire queste applicazioni sulla maggior parte delle piattaforme e dei dispositivi, inclusi i dispositivi e i prodotti Android da Apple.

IBM WebSphere MQ come server MQTT

Un'introduzione all'uso del server MQTT incluso in IBM WebSphere MQ.

Per iniziare, attenersi alla procedura descritta nei seguenti articoli:

- “InstallazioneIBM WebSphere MQ” a pagina 138
- “Configurazione del servizio MQTT dalla riga di comando” a pagina 140
- “Configurazione del servizio MQTT con IBM WebSphere MQ Explorer” a pagina 143

Nota: Puoi iniziare rapidamente utilizzando l'esempio della CLI (command - line interface). Tuttavia, se la tua configurazione è significativamente differente rispetto all'esempio, hai bisogno di più conoscenza e capacità per utilizzare in modo efficace la CLI (command - line interface). Utilizzare l'interfaccia IBM WebSphere MQ Explorer per iniziare ed eseguire facilmente le attività di configurazione standard.

Per le informazioni concettuali chiave sul componente IBM WebSphere MQ Telemetry , consultare i seguenti articoli nella documentazione del prodotto IBM WebSphere MQ :

- Connessione delle periferiche di telemetria a un gestore code
- Servizio di telemetria (MQXR)
- Canali di telemetria

Informazioni correlate

Configurazione di un gestore code per la telemetria su AIX e Linux
Configurazione di un gestore code per la telemetria su Windows
Configurare l'accodamento distribuito per inviare messaggi ai client MQTT
Amministrazione di WebSphere MQ Telemetry

InstallazioneIBM WebSphere MQ

Seguire queste istruzioni per ottenere e installare IBM WebSphere MQ e configurare IBM WebSphere MQ Telemetry su Windows o Linux.

Prima di iniziare

Per i sistemi operativi supportati dal servizio MQTT in esecuzione su IBM WebSphere MQ, vedi Requisiti di sistema di IBM WebSphere MQ Telemetry.

Ottenere una copia del materiale di installazione di IBM WebSphere MQ e una licenza in uno dei seguenti modi:

1. Chiedere all'amministratore IBM WebSphere MQ il materiale di installazione e la conferma che è possibile accettare l'accordo di licenza.
2. Ottenere una copia di valutazione di 90 giorni di IBM WebSphere MQ. Consultare: Valuta: IBM WebSphere MQ.
3. Comprare IBM WebSphere MQ. Consultare: Pagina del prodottoIBM WebSphere MQ.

Informazioni su questa attività

Installare IBM WebSphere MQ come `root` su Linuxe come amministratore su Windows. Durante l'installazione, selezionare le opzioni aggiuntive `Servizio di telemetria` e `Client di telemetria` per installare il componente IBM WebSphere MQ Telemetry . Creare un ID utente per gestire IBM WebSphere MQe verificare che l'ID utente guest sia definito. L'ID utente guest viene utilizzato nella configurazione del servizio MQTT di esempio per autorizzare MQTT l'accesso a IBM WebSphere MQ.

Dopo aver installato IBM WebSphere MQ, avviare il servizio MQTT effettuando le operazioni riportate in “Configurazione del servizio MQTT dalla riga di comando” a pagina 140 o “Configurazione del servizio MQTT con IBM WebSphere MQ Explorer” a pagina 143.

Procedura

1. Accedere come `root` su Linuxo come amministratore su Windows.
2. Installa IBM WebSphere MQ.

   Seguire le istruzioni in Installazione del server WebSphere MQ su Linux o Installazione del server WebSphere MQ su Windows. Selezionare `Servizio di telemetria` e `Client di telemetria` per installare il componente IBM WebSphere MQ Telemetry .

   In Linux, prendere nota delle istruzioni nella sezione "Operazioni successive" per rendere l'installazione primaria. Anche se questa installazione è l'unica installazione di IBM WebSphere MQ sulla stazione di lavoro, renderla primaria. Consultare Installazione singola di WebSphere MQ Versione 7.1, o successiva, configurata come installazione primaria.

   Per seguire esattamente le istruzioni di configurazione di esempio, è necessario rendere l'installazione primaria.

   più installazioni: Se si desidera lavorare con un'installazione non primaria, eseguire il comando setmqenv . Imposta l'ambiente IBM WebSphere MQ in una finestra comandi sulla stazione di lavoro. Vedere Più installazioni.

   Supponendo di aver accettato l'ubicazione di installazione predefinita offerta dal programma di installazione, IBM WebSphere MQ viene installato nelle seguenti directory:

   Linux a 64 bit

   ```
   /opt/mqm
   ```

   Windows a 32 bit

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows a 64 bit

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   La directory di installazione viene visualizzata come *MQ_INSTALLATION_PATH*

3. Opzionale: Aggiungere l'utente con cui si intende gestire IBM WebSphere MQ al gruppo `mqm` su questa stazione di lavoro.

   Questo passo è facoltativo in Windows poiché è possibile amministrare IBM WebSphere MQ come amministratore Windows . Consultare Autorità per amministrare WebSphere MQ su sistemi UNIX e Windows.

   Se la workstation Windows è un membro di un dominio, consultare Dominio Windows 2000 con autorizzazioni di sicurezza non predefinite o dominio Windows 2003 e Windows Server 2008 con autorizzazioni di sicurezza predefinite.

   Su Linux, il programma di installazione crea un utente `mqm`, come membro del gruppo `mqm`. Fornire a questo utente una parola d'ordine oppure creare un altro utente con `mqm` come gruppo principale.
4. Opzionale: Collegarsi con l'utente che ha reso membro del gruppo `mqm` .

   Questo passo è facoltativo in Windows poiché è possibile amministrare IBM WebSphere MQ come amministratore Windows .
5. Verificare che l'ID utente guest sia stato definito sulla workstation.

   L'ID utente guest è `"guest"` su Windows e `"nobody"` su Linux. L'ID utente guest non richiede autorizzazioni o diritti del sistema operativo.

Risultati

IBM WebSphere MQ è stato installato sulla workstation come installazione IBM WebSphere MQ primaria e il gruppo è stato creato `mqm`. L'installazione concede l'autorizzazione per amministrare IBM WebSphere MQ ai membri del gruppo `mqm` . I membri del gruppo di amministratori su Windows hanno anche l'autorità per gestire IBM WebSphere MQ.

Operazioni successive

1. Configura il servizio MQTT dalla riga di comando o da IBM WebSphere MQ Explorer; vedi “Configurazione del servizio MQTT con IBM WebSphere MQ Explorer” a pagina 143 o “Configurazione del servizio MQTT dalla riga di comando” a pagina 140.
2. Verificare i client Android, iOS, WebSockets, Javae "C" MQTT .
3. Una volta terminata la verifica, rimuovere il gestore code e il servizio MQTT eseguendo il comando *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` on Windows e *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` on Linux.

Informazioni correlate

Installazione di WebSphere MQ Telemetry

Installazione del server WebSphere MQ su Linux

Installazione del server WebSphere MQ in Windows

Configurazione del servizio MQTT dalla riga di comando

Seguire queste istruzioni per configurare IBM WebSphere MQ utilizzando la riga comandi per eseguire le applicazioni IBM WebSphere MQ Telemetry di esempio. I passi ti mostrano come eseguire un script per creare un servizio MQTT su un nuovo gestore code denominato `MQXR_SAMPLE_QM`.

Prima di iniziare

Devi disporre dell'accesso di amministratore a un gestore code IBM WebSphere MQ per configurare il servizio MQTT . Esistono diversi modi per ottenere l'accesso a un gestore code:

1. Ottenere una copia di IBM WebSphere MQ e creare un gestore code sulla propria workstation Linux o Windows . Seguire le istruzioni in “InstallazioneIBM WebSphere MQ” a pagina 138 per ottenere e installare IBM WebSphere MQ. È necessario selezionare anche `Servizio di telemetria` e `Client di telemetria` durante l'installazione. È anche possibile modificare un'installazione esistente per aggiungere queste opzioni.

2. Contattare un amministratore di IBM WebSphere MQ e richiedere l'accesso di gestione a un gestore code su un server su cui è installato IBM WebSphere MQ Telemetry come opzione. V 7.5.0.1 Oltre al nome del gestore code, sono necessarie almeno due porte TCP/IP per MQTT e per MQTT su WebSockets. Se si intende connettere client sicuri, sono necessarie almeno due porte in più.

Per eseguire le operazioni nell'attività esattamente come sono descritte, è necessario essere in grado di creare un gestore code denominato MQXR_SAMPLE_QMe la porta TCP/IP 1883 deve essere inutilizzata.

Informazioni su questa attività

In questa attività si esegue uno script che crea un gestore code e quindi configura il servizio MQTT per ascoltare le connessioni client MQTT V3.1 sulla porta 1883. La configurazione fornisce a chiunque l'autorizzazione a pubblicare e sottoscrivere qualsiasi argomento. La configurazione di sicurezza e controllo dell'accesso è minima ed è destinata solo a un gestore code che si trova su una rete sicura con accesso limitato. Per eseguire IBM WebSphere MQ e MQTT in un ambiente non sicuro, è necessario configurare la sicurezza. Per configurare la sicurezza per IBM WebSphere MQ e MQTT, consultare i link correlati alla fine di questa attività.

Procedura

1. Accedere con un ID utente che disponga dell'autorità amministrativa per IBM WebSphere MQ.

   Per definire un ID utente con autorità amministrativa per IBM WebSphere MQ, fare riferimento al passo 3 in “InstallazioneIBM WebSphere MQ” a pagina 138.
2. Aprire una finestra comandi ed eseguire lo script di comandi di esempio per creare e avviare il gestore code di esempio denominato MQXR_SAMPLE_QM e il servizio MQTT .

   Il percorso dello script di esempio è %MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat su Windows e *MQ_INSTALLATION_PATH*/mqxr/samples/SampleMQM.sh su Linux.

   Immettere il seguente comando per creare e configurare il gestore code:

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Risultati

L'esempio crea un canale MQTT denominato PlainText con queste proprietà su 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
```

Le proprietà del canale su Linux sono le stesse di Windows, tranne com.ibm.mq.MQXR.UserName=nobody.

MQTT V3.1 i client che si collegano alla porta 1883 accedono IBM WebSphere MQ con l'ID utente impostato nella variabile *com.ibm.mq.MQXR.UserName*. Lo script di esempio autorizza l'ID utente con i seguenti comandi 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
```

Il primo comando fornisce all'utente l'autorizzazione a pubblicare e sottoscrivere gli argomenti che ereditano le loro autorizzazioni dall'argomento di base. Il secondo comando fornisce all'utente l'autorità di inserire i messaggi nella coda di trasmissione `SYSTEM.MQTT.TRANSMIT.QUEUE` . Il servizio MQTT invia messaggi su `SYSTEM.MQTT.TRANSMIT.QUEUE` come pubblicazioni ai sottoscrittori MQTT .

Lo script avvia il servizio di MQTT sulla coda che gestisce l'ascolto delle connessioni sulla porta 1883.

Operazioni successive

Attenersi alla seguente procedura per verificare la connessione eseguendo l'applicazione MQTT V3.1 Java di esempio.

L'origine per l'applicazione Java di esempio si trova nel file `MQTTV3Sample.java` .

Per eseguire l'esempio sono richieste due finestre di comando. Eseguire l'esempio come sottoscrittore in una finestra e come publisher nell'altra.

- Windows Per avviare l'utente, eseguire il comando

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  Per pubblicare, eseguire il comando:

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux Per avviare l'utente, eseguire il comando

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  Per pubblicare, eseguire il comando:

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

Il publisher e il sottoscrittore scrivono l'output nelle relative finestre dei comandi:

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 27. Output del publisher

```
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. Output dal sottoscrittore

Il server è ora pronto per il test della tua applicazione MQTT V3.1 .

Attività correlate

Configurazione del servizio MQTT con WebSphere MQ Explorer
Seguire queste istruzioni per configurare IBM WebSphere MQ utilizzando IBM WebSphere MQ Explorer per eseguire i client IBM WebSphere MQ Telemetry di esempio. La procedura mostra come creare un servizio di MQTT eseguendo la procedura guidata di configurazione `Define sample` .

Informazioni correlate

WebSphere MQ Telemetry
Sviluppo di applicazioni per WebSphere MQ Telemetry
Amministrazione di WebSphere MQ Telemetry

Configurazione del servizio MQTT con IBM WebSphere MQ Explorer

Seguire queste istruzioni per configurare IBM WebSphere MQ utilizzando IBM WebSphere MQ Explorer per eseguire i client IBM WebSphere MQ Telemetry di esempio. La procedura mostra come creare un servizio di MQTT eseguendo la procedura guidata di configurazione `Define sample` .

Prima di iniziare

Devi disporre dell'accesso di amministratore a un gestore code IBM WebSphere MQ per configurare il servizio MQTT . Esistono diversi modi per ottenere l'accesso a un gestore code:

1. Ottenere una copia di IBM WebSphere MQ e creare un gestore code sulla propria workstation Linux o Windows . Seguire le istruzioni in “InstallazioneIBM WebSphere MQ” a pagina 138 per ottenere e installare IBM WebSphere MQ. È necessario selezionare anche `Servizio di telemetria` e `Client di telemetria` durante l'installazione. È anche possibile modificare un'installazione esistente per aggiungere queste opzioni.
2. Contattare un amministratore di IBM WebSphere MQ e richiedere l'accesso di gestione a un gestore code su un server su cui è installato IBM WebSphere MQ Telemetry come opzione. V 7.5.0.1 Oltre al nome del gestore code, sono necessarie almeno due porte TCP/IP per MQTT e per MQTT su WebSockets. Se si intende connettere client sicuri, sono necessarie almeno due porte in più.

Per eseguire le operazioni nell'attività esattamente come sono descritte, è necessario essere in grado di creare un gestore code denominato `MQXR_SAMPLE_QM`e la porta TCP/IP 1883 deve essere inutilizzata.

Informazioni su questa attività

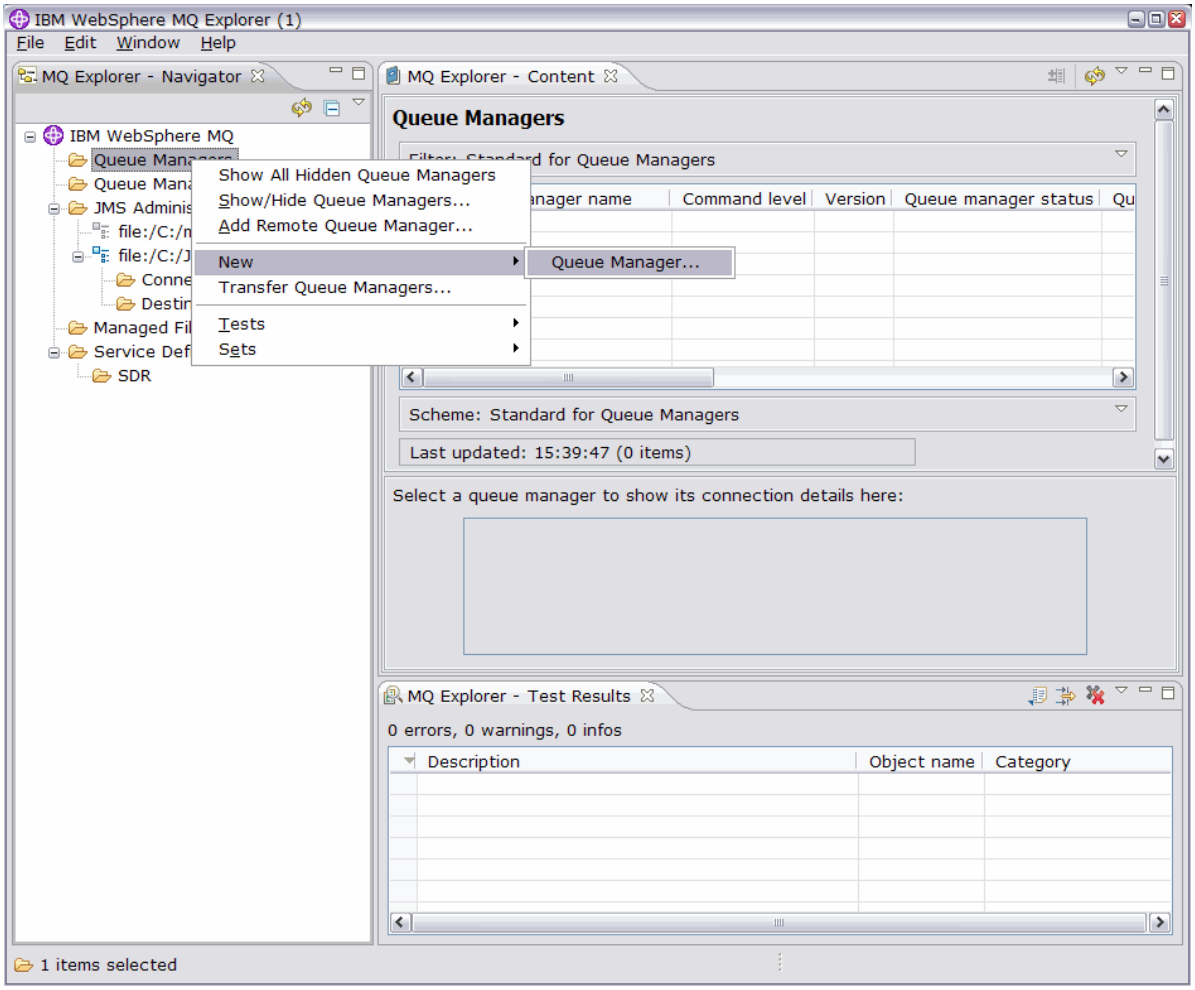
In questa attività, si esegua la procedura guidata di configurazione di IBM WebSphere MQ Explorer `Define sample` per creare un servizio MQTT in ascolto per le connessioni client MQTT V3.1 sulla porta 1883. La configurazione fornisce a chiunque l'autorizzazione a pubblicare e sottoscrivere qualsiasi argomento. La configurazione di sicurezza e controllo dell'accesso è minima ed è destinata solo a un gestore code che si trova su una rete sicura con accesso limitato. Per eseguire IBM WebSphere MQ e MQTT in un ambiente non sicuro, è necessario configurare la sicurezza. Per configurare la sicurezza per IBM WebSphere MQ e MQTT, consultare i link correlati alla fine di questa attività.

Procedura

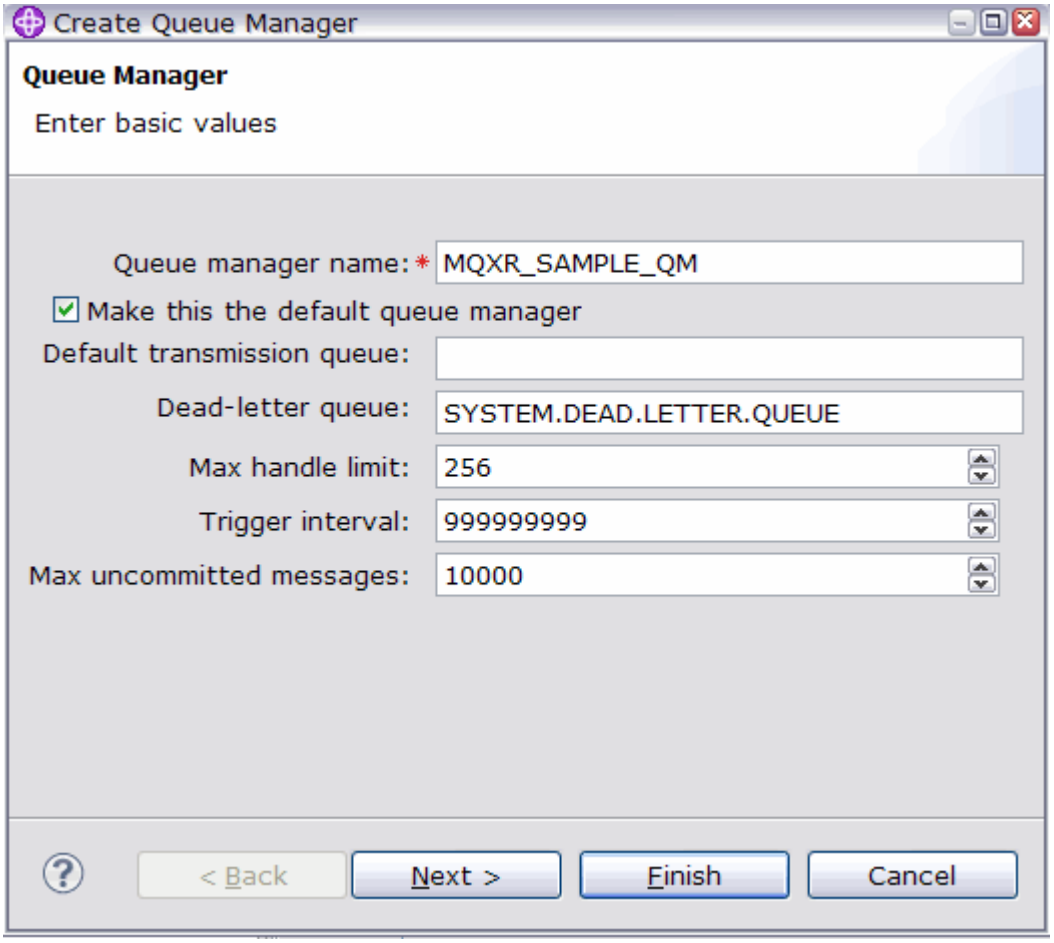
1. Accedere con un ID utente che disponga dell'autorità amministrativa per IBM WebSphere MQ.

   Per definire un ID utente con autorità amministrativa per IBM WebSphere MQ, fare riferimento al passo 3 in “InstallazioneIBM WebSphere MQ” a pagina 138.
2. Aprire una finestra di comandi ed eseguire il IBM WebSphere MQ Explorer comando **strmqcfg** per avviare IBM WebSphere MQ Explorer.
3. Creare un gestore code

   a) Avviare la procedura guidata **Nuovo gestore code**

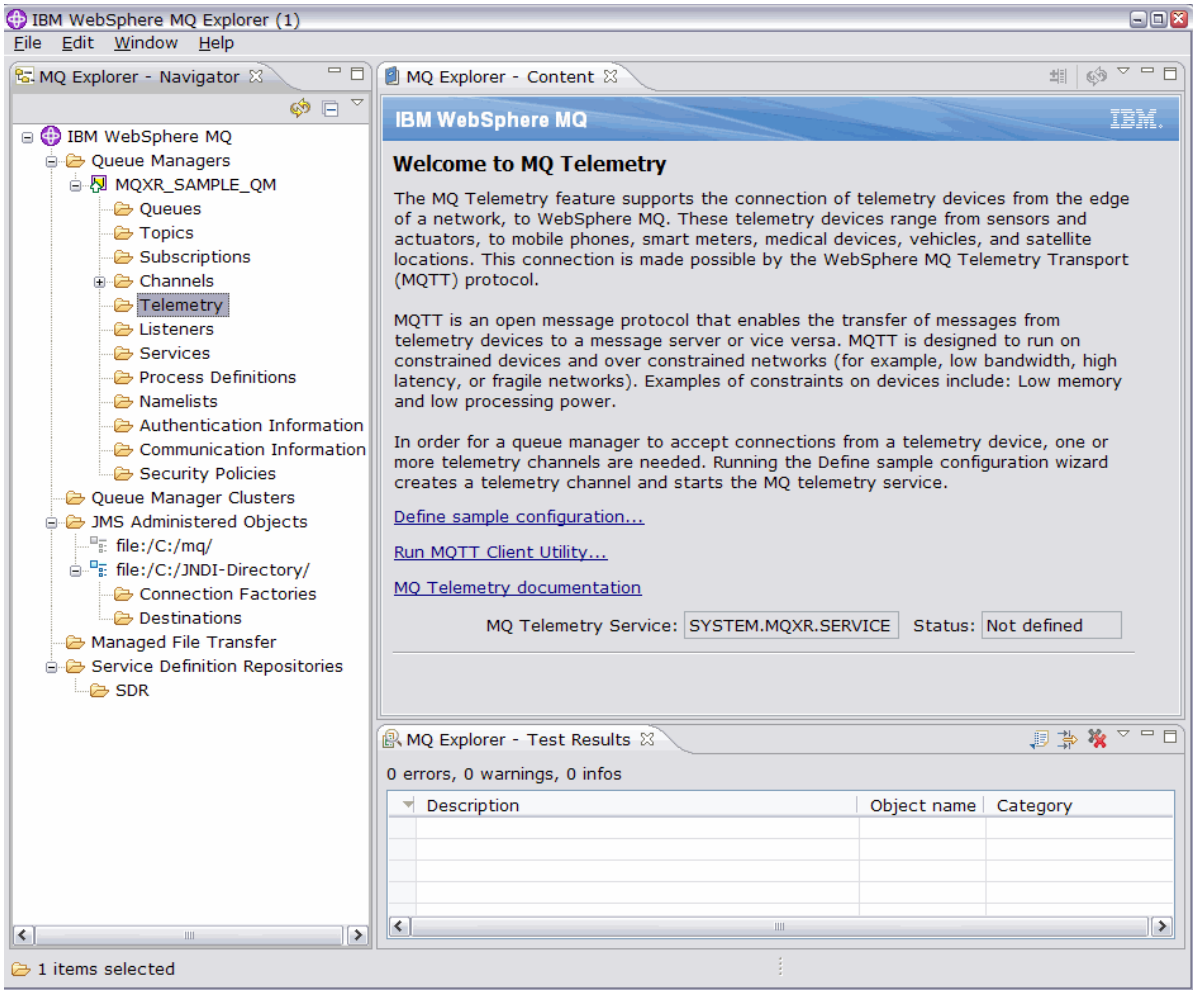


b) Immettere un **Nome gestore code**e il nome della **Dead - letter queue**. Per comodità, renderlo il gestore code predefinito. Fare clic su **Fine**.



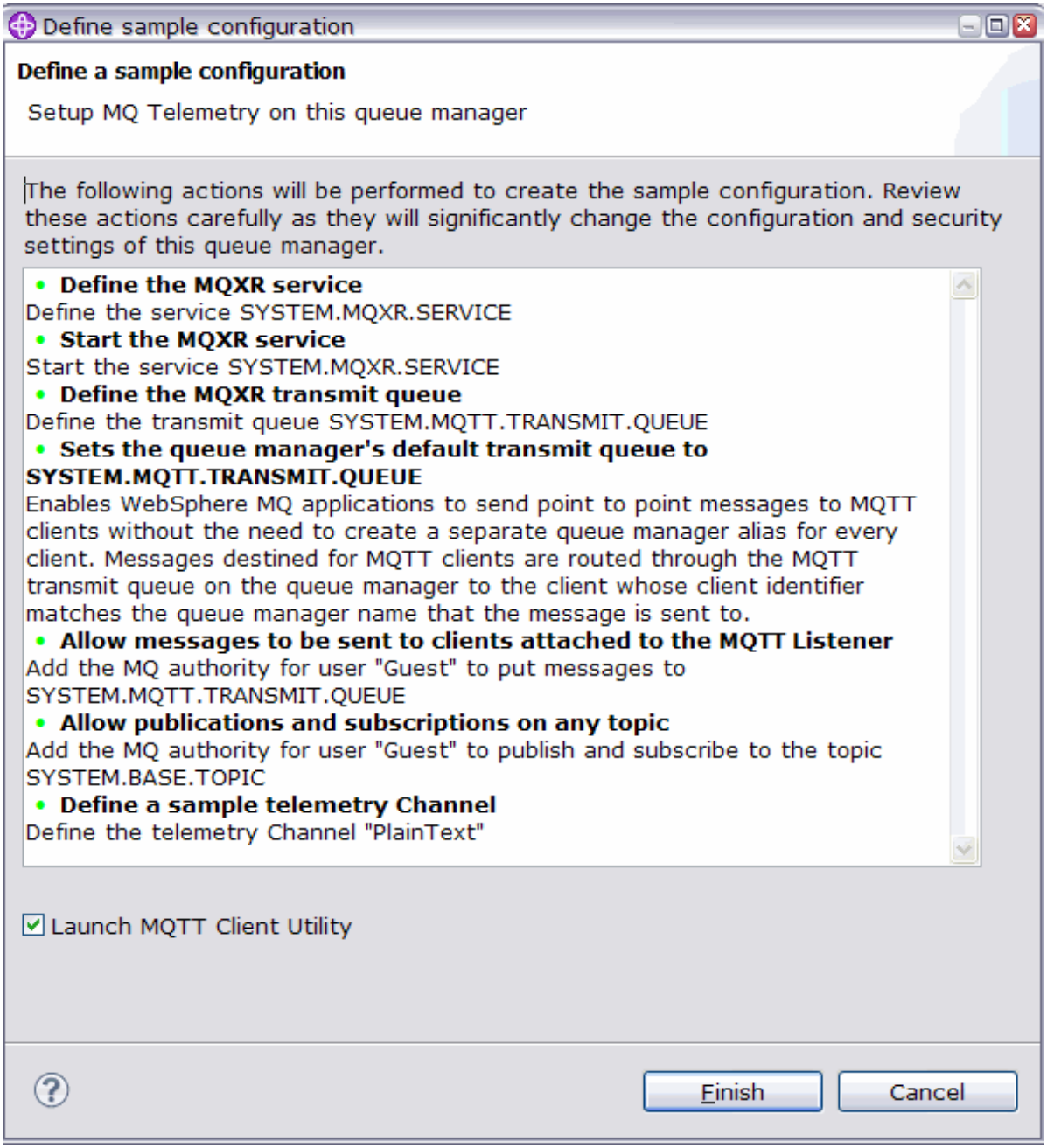
IBM WebSphere MQ Explorer crea il gestore code e lo avvia.

4. Eseguire la procedura guidata **Definisci configurazione di esempio** di Telemetria.
   a) Aprire la cartella Telemetria per il gestore code.



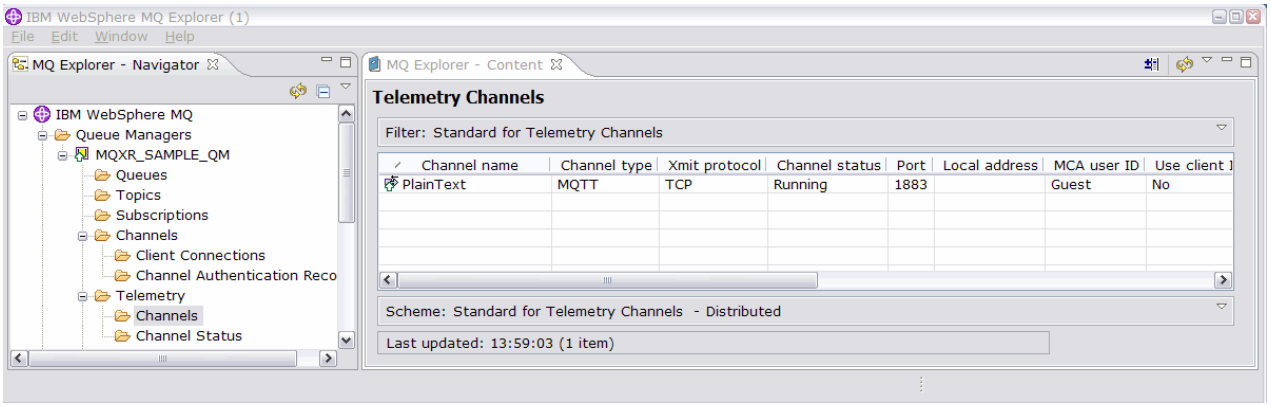
b) Fare clic su **Definisci configurazione di esempio** per avviare la procedura guidata.

c) Cliccare su **Fine** per creare il servizio di telemetria ed eseguire il programma di utilità del client MQTT



Risultati

Aprire la cartella Canali di telemetria per elencare i canali di esempio.



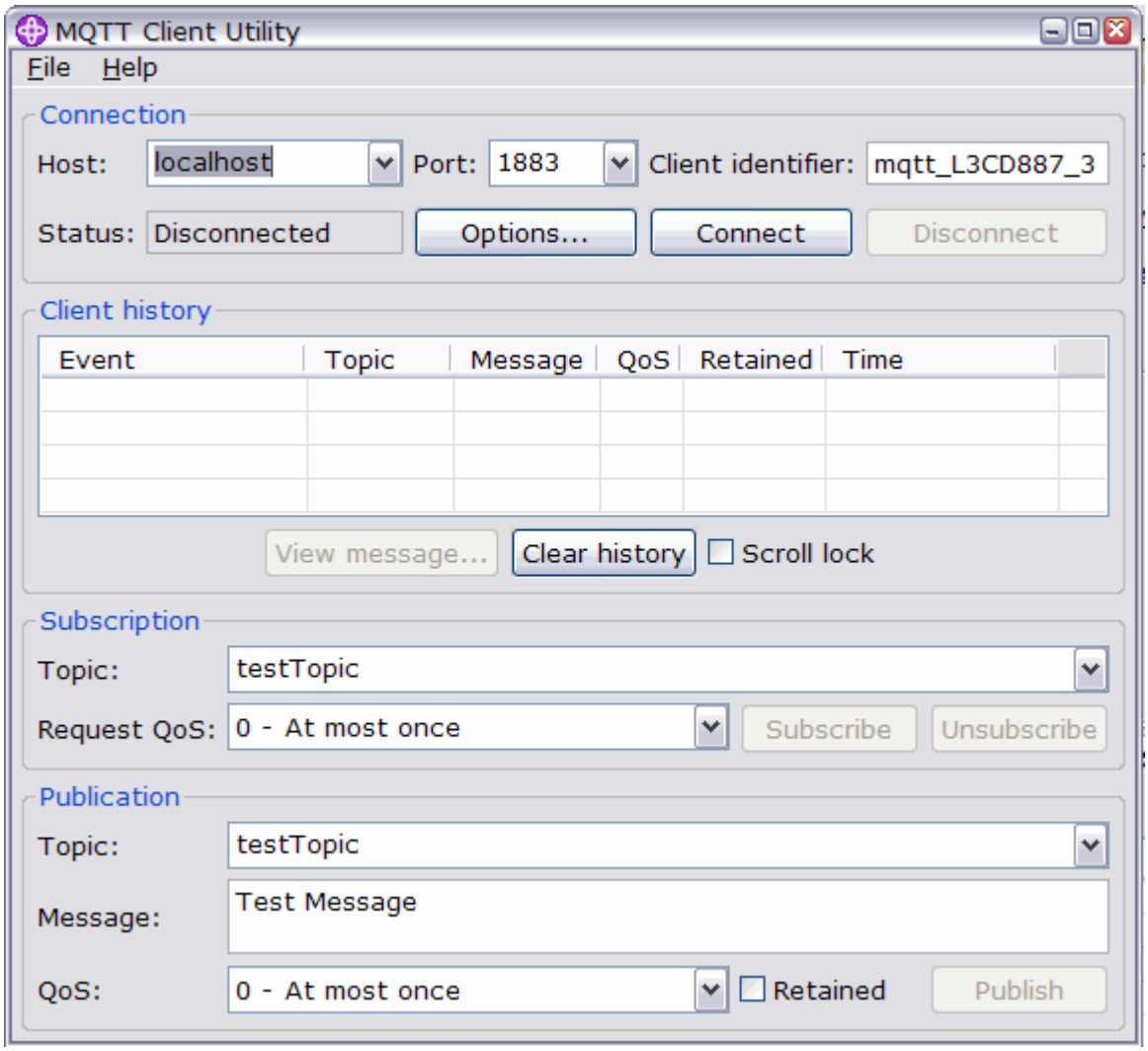
È possibile modificare le proprietà di questo canale e aggiungere ed eliminare canali in questa finestra.

Operazioni successive

Verificare la connessione eseguendo il programma di utilità MQTT Client.

1. Per avviare il programma di utilità del client, aprire la cartella **Telemetria** e fare clic su **Esegui programma di utilità del client MQTT** due volte.

   Vengono aperte due finestre **Programma di utilità del client MQTT** , identiche ma per identificativi client differenti.



2. Fare clic su **Connetti** in entrambe le finestre.
3. Fare clic su **Sottoscrivi** in entrambe le finestre.
4. Fare clic su **Pubblica** in una delle finestre. I risultati sono riportati in Figura 29 a pagina 149

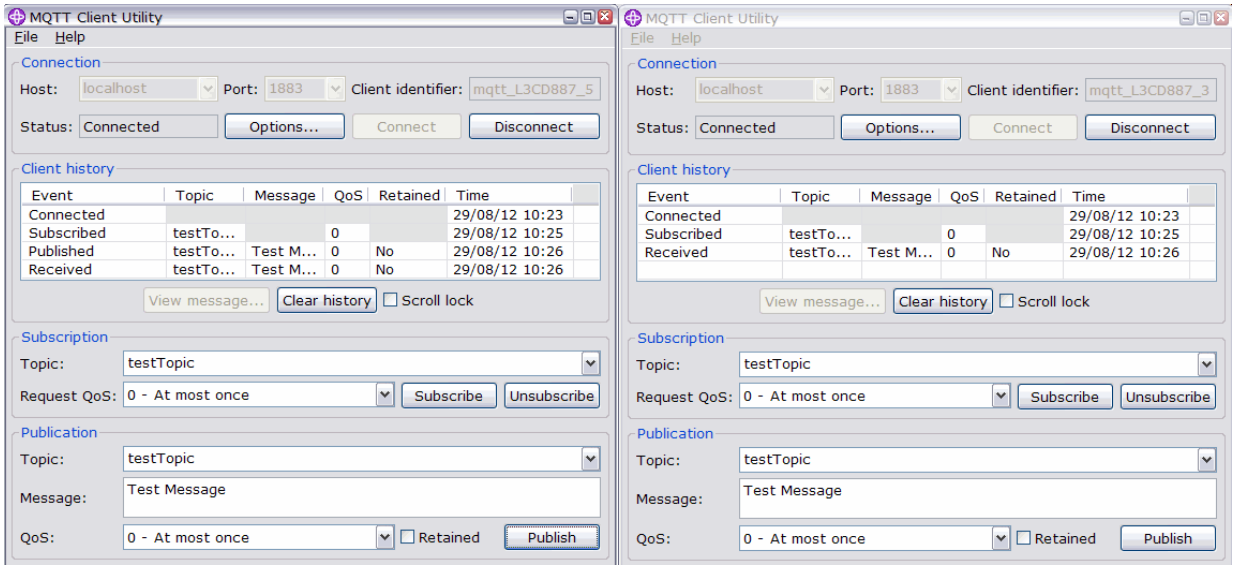


Figura 29. Risultati

5. Fare clic su **Disconnetti** in entrambe le finestre.

Il server è ora pronto per il test della tua applicazione MQTT V3.1 .

Attività correlate

Configurazione del servizio MQTT dalla riga comandi

Seguire queste istruzioni per configurare IBM WebSphere MQ utilizzando la riga comandi per eseguire le applicazioni IBM WebSphere MQ Telemetry di esempio. I passi ti mostrano come eseguire un script per creare un servizio MQTT su un nuovo gestore code denominato `MQXR_SAMPLE_QM`.

Amministrazione di WebSphere MQ Telemetry

Informazioni correlate

WebSphere MQ Telemetry

Amministrazione di WebSphere MQ Telemetry con WebSphere MQ Explorer

Sviluppo di applicazioni per WebSphere MQ Telemetry

Sicurezza

Sicurezza WebSphere MQ Telemetry

IBM WebSphere MQ Daemon di telemetria per i concetti delle periferiche

Il daemon IBM WebSphere MQ Telemetry per dispositivi è un'app client MQTT V3 avanzata. Utilizzarlo per memorizzare e inoltrare messaggi da altri client MQTT. Si connette a IBM WebSphere MQ come un client MQTT, ma è anche possibile connettervi altri client MQTT.

Il daemon è un broker di pubblicazione / sottoscrizione. I client MQTT V3 si collegano ad esso per pubblicare e sottoscrivere argomenti, utilizzando stringhe di argomenti per la pubblicazione e filtri argomenti per la sottoscrizione. La stringa di argomenti è gerarchica, con livelli di argomenti divisi per /. I filtri argomento sono stringhe di argomento che possono includere caratteri jolly + di livello singolo e un carattere jolly # multilivello come ultima parte della stringa di argomento.

Nota: I caratteri jolly nel daemon seguono le regole più restrittive di WebSphere Message Broker, v6. IBM WebSphere MQ è diverso. Supporta più caratteri jolly multilivello; i caratteri jolly possono indicare qualsiasi numero di livelli della gerarchia, in qualsiasi punto della stringa di argomenti.

Più client MQTT v3 si collegano al daemon utilizzando una porta listener. La porta listener predefinita è modificabile. È possibile definire più porte listener e assegnare diversi spazi dei nomi ad esse, consultare “Daemon WebSphere MQ Telemetry per le porte listener dei dispositivi” a pagina 157. Il daemon è esso stesso un client MQTT v3 . Configurare una connessione bridge daemon per collegare il daemon alla porta listener di un altro daemon o a un servizio WebSphere MQ Telemetry (MQXR).

È possibile configurare più bridge per il daemon WebSphere MQ Telemetry per i dispositivi. Utilizzare i bridge per collegare insieme una rete di daemon che possono scambiare pubblicazioni.

Ogni bridge può pubblicare e sottoscrivere argomenti sul proprio daemon locale. Può anche pubblicare e sottoscrivere argomenti in un altro daemon, in un broker di pubblicazione / sottoscrizione WebSphere MQ o in qualsiasi altro broker MQTT v3 a cui è connesso. Utilizzando un filtraggio argomenti, è possibile selezionare le pubblicazioni da propagare da un broker all'altro. È possibile propagare le pubblicazioni in entrambe le direzioni. È possibile propagare le pubblicazioni dal daemon locale a ciascuno dei relativi broker remoti collegati o da uno qualsiasi dei broker collegati al daemon locale; consultare "IBM WebSphere MQ Daemon di telemetria per bridge di dispositivi" a pagina 150.

IBM WebSphere MQ Daemon di telemetria per bridge di dispositivi

Un daemon IBM WebSphere MQ Telemetry per il bridge dei dispositivi collega due broker di pubblicazione / sottoscrizione utilizzando il protocollo MQTT v3 . Il bridge propaga le pubblicazioni da un broker all'altro, in entrambe le direzioni. Da un lato è presente un daemon WebSphere MQ Telemetry per la connessione bridge dei dispositivi e dall'altro potrebbe essere un gestore code o un altro daemon. Un gestore code è connesso alla connessione bridge utilizzando un canale di telemetria. Un daemon è connesso alla connessione bridge utilizzando un listener daemon.

Il daemon IBM WebSphere MQ Telemetry per le periferiche supporta una o più connessioni simultanee ad altri broker. Le connessioni dal daemon sono chiamate bridge e sono definite da voci di connessione nel file di configurazione del daemon. Le connessioni a IBM WebSphere MQ vengono effettuate utilizzando i canali di telemetria IBM WebSphere MQ , come mostrato nella seguente figura:

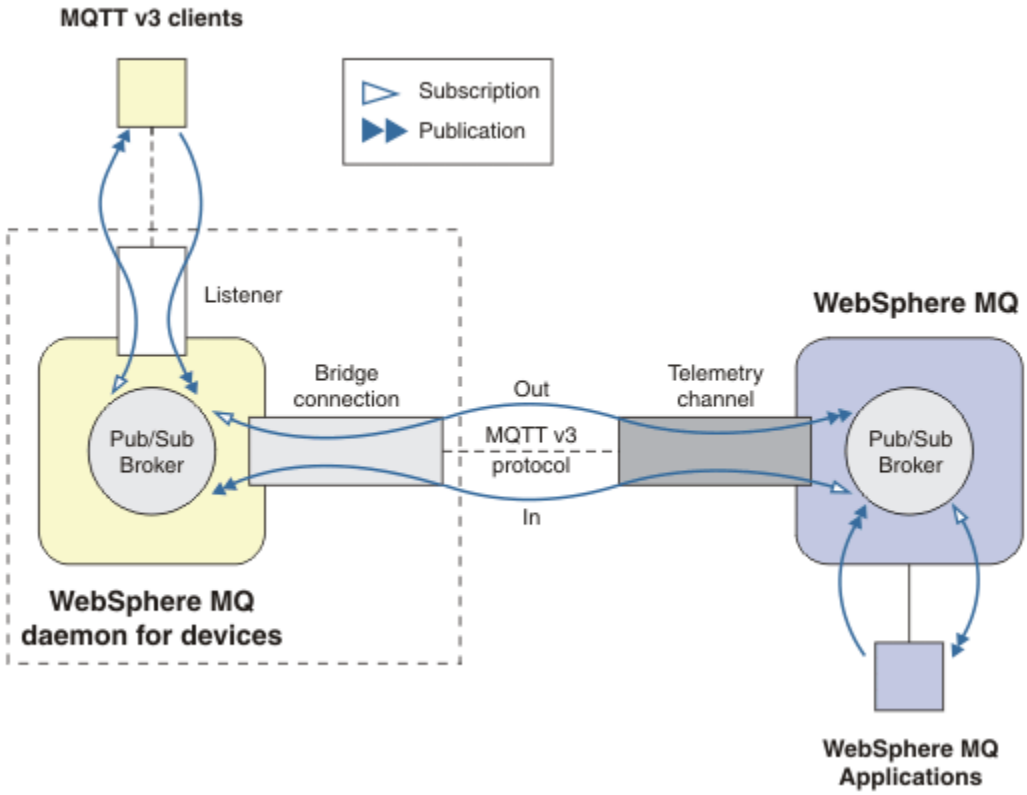


Figura 30. Connessione IBM WebSphere MQ Telemetry daemon for devices a IBM WebSphere MQ

Un bridge collega il daemon a un altro broker come client MQTT v3 . I parametri bridge riflettono gli attributi di un client MQTT v3 .

Un bridge è più di una connessione. Agisce come agente di pubblicazione e sottoscrizione situato tra due broker di pubblicazione / sottoscrizione. Il broker locale è il daemon IBM WebSphere MQ Telemetry per i

dispositivi e il broker remoto è qualsiasi broker di pubblicazione / sottoscrizione che supporta il protocollo MQTT v3 . Di solito, il broker remoto è un altro deamon o IBM WebSphere MQ.

Il lavoro del bridge è quello di propagare le pubblicazioni tra i due broker. Il ponte è bidirezionale. Propaga le pubblicazioni in entrambe le direzioni. Figura 30 a pagina 150 illustra il modo in cui il bridge collega il daemon IBM WebSphere MQ Telemetry per i dispositivi a IBM WebSphere MQ. “Impostazioni dell'argomento di esempio per il ponte” a pagina 151 utilizza esempi per illustrare come utilizzare il parametro dell'argomento per configurare il bridge.

Le frecce In e Out in Figura 30 a pagina 150 indicano la bidirezionalità del bridge. Ad un'estremità della freccia, viene creata una sottoscrizione. Le pubblicazioni che corrispondono alla sottoscrizione vengono pubblicate sul broker all'estremità opposta della freccia. La freccia viene etichettata in base al flusso di pubblicazioni. Le pubblicazioni vengono portate In al daemon e Out dal daemon. L'importanza delle etichette è che vengono utilizzate nella sintassi del comando. Tenere presente che In e Out fanno riferimento al punto in cui vengono inviate le pubblicazioni e non al punto in cui viene inviata la sottoscrizione.

Altri client, applicazioni o broker potrebbero essere connessi a IBM WebSphere MQ o al daemon WebSphere MQ Telemetry per dispositivi. Pubblicano e sottoscrivono argomenti nel broker a cui sono connessi. Se il broker è IBM WebSphere MQ, gli argomenti potrebbero essere in cluster o distribuiti e non esplicitamente definiti nel gestore code locale.

Usi dei ponti

Collegare i daemon utilizzando le connessioni bridge e i listener. Connettere i daemon e i gestori code utilizzando le connessioni bridge e i canali di telemetria. Quando si connettono più broker insieme, è possibile creare dei loop. Attenzione: le pubblicazioni potrebbero circolare all'infinito intorno a un loop di broker, non rilevati.

Alcuni dei motivi per utilizzare i daemon collegati a IBM WebSphere MQ sono i seguenti:

Ridurre il numero di connessioni client MQTT a WebSphere MQ
Utilizzando una gerarchia di daemon, è possibile connettere molti client a WebSphere MQ; un numero di client superiore al numero di connessioni di un singolo gestore code alla volta.

Memorizza e inoltra i messaggi tra client MQTT e WebSphere MQ
È possibile utilizzare l'archiviazione e l'inoltro per evitare di mantenere connessioni continue tra client e IBM WebSphere MQ, se i client non dispongono di una propria memoria. È possibile utilizzare più tipi di connessione tra il client MQTT e WebSphere MQ; consultare Scenari e concetti di telemetria per il monitoraggio e il controllo.

Filtrare le pubblicazioni scambiate tra client MQTT e WebSphere MQ
In genere, le pubblicazioni si dividono in messaggi elaborati localmente e messaggi che coinvolgono altre applicazioni. Le pubblicazioni locali possono includere i flussi di controllo tra sensori e attuatori e le pubblicazioni remote includono richieste di letture, stato e comandi di configurazione.

Modificare gli spazi argomenti delle pubblicazioni
Evitare che le stringhe di argomenti dai client collegati a porte listener differenti entrino in conflitto tra loro. L'esempio utilizza il daemon per etichettare le letture contatore provenienti da edifici differenti; consultare Separazione degli spazi argomento di gruppi di client diversi.

Impostazioni dell'argomento di esempio per il ponte

Pubblica tutto nel broker remoto - utilizzando i valori predefiniti

La direzione predefinita viene denominata oute il bridge pubblica gli argomenti sul broker remoto. Il parametro topic controlla quali argomenti vengono propagati utilizzando filtri argomento.

Il bridge utilizza il parametro topic in Figura 31 a pagina 152 per sottoscrivere tutti gli elementi pubblicati sul daemon locale dai client MQTT o da altri broker. Il bridge pubblica gli argomenti sul broker remoto connesso dal bridge.

```
connection Daemon1
topic #
```

Figura 31. Pubblica tutto sul broker remoto

Pubblica tutto nel broker remoto - esplicito

L'impostazione topic nel seguente frammento di codice fornisce lo stesso risultato dell'utilizzo dei valori predefiniti. L'unica differenza è che il parametro **direction** è esplicito. Utilizzare la direzione `out` per sottoscrivere il broker locale, il daemon e pubblicare sul broker remoto. Le pubblicazioni create sul daemon locale a cui il bridge ha sottoscritto, vengono pubblicate sul broker remoto.

```
connection Daemon1
topic # out
```

Figura 32. Pubblica tutto nel broker remoto - esplicito

Pubblica tutto nel broker locale

Invece di utilizzare la direzione `out`, è possibile impostare la direzione opposta, `in`. Il seguente frammento di codice configura il bridge per sottoscrivere tutti gli elementi pubblicati sul broker remoto connesso dal bridge. Il bridge pubblica gli argomenti nel broker locale, il daemon.

```
connection Daemon1
topic # in
```

Figura 33. Pubblica tutto nel broker locale

Pubblicare tutto dall'argomento di esportazione nel Broker locale all'argomento di importazione nel Broker remoto

Utilizzare due parametri argomento aggiuntivi, **local_prefix** e **remote_prefix**, per modificare il filtro argomento, # negli esempi precedenti. Un parametro viene utilizzato per la modifica del filtro argomento utilizzato nella sottoscrizione e l'altro parametro viene utilizzato per modificare l'argomento in cui viene pubblicata la pubblicazione. L'effetto è quello di sostituire l'inizio della stringa di argomenti utilizzata in un broker con un'altra stringa di argomenti sull'altro broker.

In base alla direzione del comando dell'argomento, il significato di **local_prefix** e di **remote_prefix** viene invertire. Se la direzione è `out`, il valore predefinito, **local_prefix** viene utilizzato come parte della sottoscrizione argomento e **remote_prefix** sostituisce la parte **local_prefix** della stringa argomento nella pubblicazione remota. Se la direzione è `in`, **remote_prefix** diventa parte della sottoscrizione remota e **local_prefix** sostituisce la parte **remote_prefix** della stringa di argomenti.

La prima parte di una stringa di argomenti viene spesso considerata come una definizione di uno spazio argomenti. Utilizzare i parametri aggiuntivi per modificare lo spazio argomento in cui viene pubblicato un argomento. È possibile eseguire questa operazione per evitare che l'argomento venga propagato in conflitto con un altro argomento sul broker di destinazione o per rimuovere una stringa di argomento del punto di montaggio.

Ad esempio, nel frammento di codice riportato di seguito, tutte le pubblicazioni nella stringa di argomenti `export/#` sul daemon vengono ripubblicate in `import/#` sul broker remoto.

```
topic # out export/ import/
```

Figura 34. Pubblicare tutto dall'argomento di esportazione nel Broker locale all'argomento di importazione nel Broker remoto

Pubblicare tutto nell'argomento di importazione nel broker locale dall'argomento di esportazione nel broker remoto

Il seguente frammento di codice mostra la configurazione invertita; il bridge sottoscrive tutto ciò che è pubblicato con la stringa di argomenti `export/#` nel broker remoto e lo pubblica in `import/#` nel broker locale.

```
connection Daemon1
topic # in import/ export/
```

Figura 35. Pubblicare tutto nell'argomento di importazione nel broker locale dall'argomento di esportazione nel broker remoto

Pubblicare tutto dal punto di montaggio 1884/ al broker remoto con le stringhe di argomenti originali

Nel seguente frammento di codice, il bridge sottoscrive tutto ciò che viene pubblicato dai client connessi al punto di montaggio 1884/ sul daemon locale. Il bridge pubblica tutti gli elementi pubblicati sul punto di montaggio sul broker remoto. La stringa del punto di montaggio 1884/ viene eliminata dagli argomenti pubblicati sul broker remoto. *local_prefix* è uguale alla stringa del punto di montaggio 1884/e *remote_prefix* è una stringa vuota.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Figura 36. Pubblicare tutto dal punto di montaggio 1884/ al broker remoto con le stringhe di argomenti originali.

Separazione degli spazi argomento di client differenti connessi a daemon differenti

Si supponga che un'applicazione sia scritta per contatori elettrici per pubblicare letture contatore per un edificio. Le letture vengono pubblicate utilizzando i client MQTT per un daemon ospitato nello stesso edificio. L'argomento selezionato per le pubblicazioni è `power`. La stessa applicazione viene distribuita a un numero di edifici in un complesso. Per il monitoraggio del sito e l'archiviazione dei dati, le letture da tutti gli edifici vengono aggregate utilizzando connessioni bridge. Le connessioni collegano i daemon di creazione a WebSphere MQ in un'ubicazione centrale.

Un'applicazione client identica viene utilizzata in tutti gli edifici. Questa app viene pubblicata nell'argomento `power`. Tuttavia, i dati devono essere differenziati mediante la creazione. Questa operazione viene eseguita dal daemon per ogni edificio, che aggiunge il numero di edificio come prefisso al nome dell'argomento. Il ponte dal primo edificio nel complesso utilizza il prefisso `meters/building01/`, dal secondo il prefisso è `meters/building02/`. Le letture degli altri edifici seguono lo stesso schema. WebSphere MQ riceve quindi le letture con argomenti come `meters/building01/power`.

Il file di configurazione per ogni demone ha un'istruzione di argomento che segue il pattern nel seguente frammento di codice:

```
connection Daemon1
topic power out "" meters/building01/
```

Figura 37. Separare gli spazi argomenti dei client connessi a diversi daemon

Nel frammento di codice precedente, la stringa vuota è un segnaposto per il parametro local_prefix non utilizzato.

Nota: Questo esempio è un po 'artificiale, e inteso solo come un'illustrazione. In pratica, lo spazio argomento su cui l'applicazione pubblica è probabilmente configurabile.

Separare gli spazi argomenti dei client connessi allo stesso daemon

Supponiamo che un singolo daemon venga utilizzato per collegare tutti i misuratori di potenza. Supponendo che nell'applicazione sia possibile configurare la connessione a porte differenti, è possibile distinguere gli edifici collegando i contatori da edifici differenti a porte listener differenti, come nel seguente frammenti di codice. Di nuovo, l'esempio è inventato; illustra come potrebbero essere utilizzati i punti di montaggio.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Figura 38. Separare gli spazi argomenti dei client connessi allo stesso daemon

Riassocia diversi argomenti per le pubblicazioni che fluiscono in entrambe le direzioni

Nella configurazione contenuta nel seguente frammento di codice, il bridge sottoscrive il singolo argomento b sul broker remoto e inoltra le pubblicazioni relative a b al daemon locale, modificando l'argomento in a. Il bridge inoltre sottoscrive il singolo argomento x nel broker locale e inoltra le pubblicazioni relative a x al broker remoto, modificando l'argomento in y.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Figura 39. Riassocia diversi argomenti per le pubblicazioni che fluiscono in entrambe le direzioni

Un punto importante su questo esempio è che diversi argomenti sono sottoscritti e pubblicati in entrambi i broker. Gli spazi argomenti in entrambi i broker sono disgiunti.

Riassocia gli stessi argomenti per le pubblicazioni che scorrono in entrambe le direzioni (loop)

A differenza dell'esempio precedente, la configurazione in Figura 40 a pagina 155, in genere, risulta in un loop. Nell'istruzione dell'argomento `topic "" in a b`, il bridge sottoscrive b in remoto e pubblica in a localmente. Nell'altra istruzione dell'argomento, il bridge sottoscrive a localmente e pubblica in b in remoto. La stessa configurazione può essere scritta come mostrato in Figura 41 a pagina 155.

Il risultato generale è che se un client pubblica in b in remoto, la pubblicazione viene trasferita al daemon locale come pubblicazione sull'argomento a. Tuttavia, quando viene pubblicato dal bridge al daemon locale sull'argomento a, la pubblicazione corrisponde alla sottoscrizione effettuata dal bridge all'argomento locale a. La sottoscrizione è `topic "" out a b`. Di conseguenza, la pubblicazione viene trasferita nuovamente al broker remoto come pubblicazione sull'argomento b. Il bridge è ora sottoscritto all'argomento remoto be il ciclo inizia nuovamente.

Alcuni broker implementano il rilevamento del loop per evitare che si verifichi il loop. Ma il meccanismo di rilevamento del loop deve funzionare quando diversi tipi di broker sono collegati tra loro. Il rilevamento dei loop non funziona se WebSphere MQ è collegato al daemon WebSphere MQ Telemetry per i dispositivi. Funziona se due daemon IBM WebSphere MQ Telemetry per dispositivi sono collegati tra loro. Per impostazione predefinita, il rilevamento loop è attivato; consultare try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Figura 40. !Riassocia gli stessi argomenti per le pubblicazioni che fluiscono in entrambe le direzioni

```
connection Daemon1
topic "" both a b
```

Figura 41. !Riassociare gli stessi argomenti per le pubblicazioni che fluiscono in entrambe le direzioni, utilizzando `both`.

La configurazione in Figura 39 a pagina 154 è uguale a Figura 40 a pagina 155.

Disponibilità delle connessioni bridge IBM WebSphere MQ Telemetry daemon for devices

Configurare più indirizzi di connessione bridge IBM WebSphere MQ Telemetry daemon for devices per connettersi al primo broker remoto disponibile. Se il broker è un gestore code a più istanze, fornire entrambi gli indirizzi TCP/IP. Configurare una connessione primaria per connettersi, o riconnettersi, al server primario, quando è disponibile.

Il parametro del bridge di connessione, indirizzi, è un elenco di indirizzi socket TCP/IP. Il bridge tenta di connettersi a ciascun indirizzo a turno, fino a quando non effettua una connessione corretta. I parametri di connessione round_robin e start_type controllano il modo in cui gli indirizzi vengono utilizzati una volta stabilita una corretta connessione.

Se start_type è auto, manualo lazy, se la connessione ha esito negativo, il bridge tenta di riconnettersi. Usa ogni indirizzo a turno, con un ritardo di circa 20 secondi tra ogni tentativo di connessione. Se `tipo_avvio` è una volta, se la connessione ha esito negativo, il bridge non tenta di riconnettersi automaticamente.

Se round_robin è true, i tentativi di connessione bridge iniziano dal primo indirizzo nell'elenco e tentano di volta in volta ogni indirizzo nell'elenco. Inizia di nuovo al primo indirizzo, quando l'elenco è esaurito. Se c'è solo un indirizzo nell'elenco, lo tenta di nuovo ogni 20 secondi.

Se round_robin è false, viene data la preferenza al primo indirizzo dell'elenco, denominato server primario. Se il primo tentativo di connessione al server primario non riesce, il bridge continua a provare a riconnettersi al server primario in background. Allo stesso tempo, il ponte tenta di connettersi utilizzando gli altri indirizzi nell'elenco. Quando i tentativi di connessione in background al server principale hanno esito positivo, il bridge si disconnette dalla connessione corrente e passa alla connessione del server principale.

Se una connessione viene disconnessa volontariamente, ad esempio immettendo un comando **`connection_stop`** , se la connessione viene riavviata, tenta di utilizzare nuovamente lo stesso indirizzo. Se la connessione è stata disconnessa a causa di un errore di connessione o se il broker remoto ha rilasciato la connessione, il bridge attende 20 secondi. Tenta quindi di collegarsi all'indirizzo successivo nell'elenco, o allo stesso indirizzo, se c'è un solo indirizzo nell'elenco.

Connessione a un gestore code a più istanze

In una configurazione del gestore code a più istanze, il gestore code viene eseguito su due server differenti con indirizzi IP diversi. In genere, i canali di telemetria vengono configurati senza un indirizzo IP specifico. Sono configurati solo con un numero di porta. Quando il canale di telemetria viene avviato, per impostazione predefinita seleziona il primo indirizzo di rete disponibile sul server locale.

Configurare il parametro indirizzi della connessione bridge con i due indirizzi IP utilizzati dal gestore code. Impostare round_robin su true.

Se l'istanza del gestore code attiva ha esito negativo, il gestore code passa all'istanza in standby. Il daemon rileva che la connessione all'istanza attiva è stata interrotta e tenta di riconnettersi all'istanza standby. Utilizza l'altro indirizzo IP nell'elenco di indirizzi configurati per la connessione bridge.

Il gestore code a cui si connette il bridge è ancora lo stesso gestore code. Il gestore code ripristina il proprio stato. Se cleansession è impostato su false, la sessione di connessione bridge viene ripristinata allo stesso stato di prima del failover. La connessione riprende dopo un ritardo. I messaggi con "almeno una volta" o "al massimo una volta" di QoS (quality of service) non vengono persi e le sottoscrizioni continuano a funzionare.

Il tempo di riconnessione dipende dal numero di canali e client che si riavviano all'avvio dell'istanza in standby e dal numero di messaggi in corso. La connessione bridge potrebbe tentare di riconnettersi a entrambi gli indirizzi IP un numero di volte prima che la connessione venga ristabilita.

Non configurare un canale di telemetria del gestore code a più istanze con un indirizzo IP specifico. L'indirizzo IP è valido solo su un server.

Se si utilizza una soluzione di alta disponibilità alternativa, che gestisce l'indirizzo IP, potrebbe essere corretto configurare un canale di telemetria con un indirizzo IP specifico.

pulisci sessione

Una connessione bridge è una sessione client MQTT v3 . È possibile controllare se una connessione avvia una nuova sessione o se ripristina una sessione esistente. Se ripristina una sessione esistente, la connessione bridge conserva le sottoscrizioni e le pubblicazioni conservate della sessione precedente.

Non impostare cleansession su false se address elenca più indirizzi IP e gli indirizzi IP si connettono a canali di telemetria ospitati da gestori code differenti o a daemon di telemetria differenti. Lo stato della sessione non viene trasferito tra gestori code o daemon. Il tentativo di riavviare una sessione esistente su un altro gestore code o daemon comporta l'avvio di una nuova sessione. I messaggi in dubbio vengono persi e le sottoscrizioni potrebbero non comportarsi come previsto.

notifiche

Un'applicazione può tenere traccia se la connessione bridge è in esecuzione utilizzando le notifiche. Una notifica è una pubblicazione con il valore `1`, connesso o `0`, disconnesso. Viene pubblicato in *topicString* definito dal parametro `notification_topic` . Il valore predefinito di *topicString* Š $SYS/`broker`/ `connection`/ *clientIdentifier*/`state`. La *topicString* predefinita contiene il prefisso $SYS. Sottoscrivi gli argomenti che iniziano con $SYS definendo un filtro argomenti che inizia con $SYS. Il filtro argomenti #, sottoscrive tutto, non sottoscrive gli argomenti che iniziano con $SYS sul daemon. Considerare $SYS come la definizione di uno spazio argomenti di sistema speciale distinto dallo spazio argomenti dell'applicazione.

Notifiche abilita IBM WebSphere MQ Telemetry daemon for devices a notificare ai client MQTT quando un bridge è connesso o disconnesso.

intervallo_keepaliv

Il parametro di connessione bridge keepalive_interval imposta l'intervallo tra il bridge che invia un ping TCP/IP al server remoto. L'intervallo predefinito è di 60 secondi. Il ping impedisce la chiusura della sessione TCP/IP da parte del server remoto o di un firewall, che rileva un periodo di inattività sulla connessione.

clientID

Una connessione bridge è una sessione client MQTT v3 e ha un `clientIdentifier` impostato dal parametro di connessione bridge clientid. Se si intende ripristinare una sessione precedente impostando il parametro cleansession su false, il `clientIdentifier` utilizzato in ciascuna sessione deve essere lo stesso. Il valore predefinito di clientid è *hostname.connectionName*, che rimane lo stesso.

Installazione, verifica, configurazione e controllo del daemon WebSphere MQ Telemetry per i dispositivi

L'installazione, la configurazione e il controllo del daemon sono basati su file.

Installare il daemon copiando il SDK (Software Development Kit) sul dispositivo su cui si sta per eseguire il daemon.

Come esempio, eseguire il programma di utilità del client MQTT e collegarsi al daemon WebSphere MQ Telemetry per i dispositivi come broker di pubblicazione / sottoscrizione; consultare Utilizzare il daemon WebSphere MQ Telemetry per i dispositivi come broker di pubblicazione / sottoscrizione.

Configurare il daemon creando un file di configurazione; consultare WebSphere MQ Telemetry daemon for devices configuration file.

Controllare un daemon in esecuzione creando comandi nel file, `amqtdd.upd`. Ogni 5 secondi il daemon legge il file, esegue i comandi ed elimina il file; consultare il daemon WebSphere MQ Telemetry per il file di comando dei dispositivi.

Daemon WebSphere MQ Telemetry per le porte listener dei dispositivi

Connettere i client MQTT V3 al daemon WebSphere MQ Telemetry per i dispositivi che utilizzano le porte listener. È possibile qualificare una porta listener con un punto di montaggio e un numero massimo di connessioni.

Una porta del listener deve corrispondere al numero di porta specificato sul metodo `connect(serverURI)` del client MQTT di un client che si connette a questa porta. Il valore predefinito è `1883`sia sul client che sul daemon.

È possibile modificare la porta predefinita per il daemon impostando la definizione globale `port` nel file di configurazione del daemon. È possibile impostare porte specifiche aggiungendo una definizione `listener` al file di configurazione del daemon.

Per ogni porta listener, diversa da quella predefinita, è possibile specificare un punto di montaggio per isolare i client. I client connessi a una porta con un punto di montaggio sono isolati dagli altri client; consultare “Daemon WebSphere MQ Telemetry per i punti di montaggio dei dispositivi” a pagina 158.

È possibile limitare il numero di client che possono connettersi a qualsiasi porta. Impostare la definizione globale `max_connections` per limitare le connessioni alla porta predefinita o qualificare ciascuna porta listener con `max_connections`.

Esempio

Un esempio di un file di configurazione che modifica la porta predefinita da `1883` a `1880`e limita la connessione alla porta `1880` a `10000`. Le connessioni alla porta `1884` si limitano a `1000`. I client collegati alla porta `1884` sono isolati dai client collegati ad altre porte.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Daemon WebSphere MQ Telemetry per i punti di montaggio dei dispositivi

È possibile associare un punto di montaggio a una porta listener utilizzata dai client MQTT per connettersi a un daemon WebSphere MQ Telemetry per dispositivi. Un punto di montaggio isola le pubblicazioni e le sottoscrizioni scambiate dai clienti MQTT utilizzando una porta listener dai client MQTT connessi a una porta listener differente.

I client collegati ad una porta listener con un punto di montaggio non possono mai scambiare direttamente argomenti con i client collegati ad altre porte listener. I client collegati a una porta listener senza un punto di montaggio possono pubblicare o sottoscrivere argomenti di qualsiasi client. I client non sanno se sono collegati tramite un punto di montaggio o meno; non fa alcuna differenza per le stringhe di argomenti create dai client.

Un punto di montaggio è una stringa di testo che ha come prefisso la stringa di argomenti di pubblicazioni e sottoscrizioni. Ha come prefisso tutte le stringhe di argomento create dai client collegati alla porta del listener con un punto di montaggio. La stringa di testo viene eliminata da tutte le stringhe di argomenti inviate a client collegati alla porta listener.

Se una porta listener non ha un punto di montaggio, le stringhe di argomenti delle pubblicazioni e delle sottoscrizioni create e ricevute dai client collegati alla porta non vengono modificate.

Creare stringhe di punto di montaggio con un `/`finale. In questo modo il punto di montaggio è l'argomento principale della struttura ad albero degli argomenti per il punto di montaggio.

Esempio

Un file di configurazione contiene le seguenti porte listener:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Un client, collegato alla porta 1883, crea una sottoscrizione a `MyTopic`. Il daemon registra la sottoscrizione come `1883/MyTopic`. Un altro client collegato alla porta 1883 pubblica un messaggio sull'argomento, `MyTopic`. Il daemon modifica la stringa argomento in `1883/MyTopic` e ricerca le sottoscrizioni corrispondenti. Il sottoscrittore sulla porta 1883 riceve la pubblicazione con la stringa di argomenti originale `MyTopic`. Il daemon ha rimosso il prefisso del punto di montaggio dalla stringa argomento.

Un altro client, collegato alla porta 1884, pubblica anche l'argomento `MyTopic`. Questa volta il daemon registra l'argomento come `1884/MyTopic`. Il sottoscrittore (subscriber) sulla porta 1883 non riceve la pubblicazione perché il punto di montaggio differente risulta in una sottoscrizione con una stringa di argomenti differente.

Un client, collegato alla porta 1885, pubblica sull'argomento, `1883/MyTopic`. Il daemon non modifica la stringa argomento. Il sottoscrittore sulla porta 1883 riceve la pubblicazione in `MyTopic`.

Daemon WebSphere MQ Telemetry per la qualità del servizio dei dispositivi, le sottoscrizioni durevoli e le pubblicazioni conservate

Le impostazioni QoS (Quality of Service) si applicano solo a un daemon in esecuzione. Se un daemon si arresta, in modo controllato o a causa di un errore, lo stato dei messaggi in corso viene perso. La consegna di un messaggio almeno una volta, o al massimo una volta, non può essere garantita se il daemon si arresta. Il daemon WebSphere MQ Telemetry per dispositivi supporta la persistenza limitata. Impostare il parametro di configurazione **`retained_persistence`** per salvare le pubblicazioni e le sottoscrizioni conservate quando il daemon è arrestato.

A differenza di WebSphere MQ, il daemon WebSphere MQ Telemetry per dispositivi non registra i dati persistenti. Lo stato della sessione, lo stato del messaggio e le pubblicazioni conservate non vengono salvate in modo transazionale. Per impostazione predefinita, il daemon elimina tutti i dati quando si arresta. È possibile impostare un'opzione per controllare periodicamente le sottoscrizioni e le

pubblicazioni conservate. Lo stato del messaggio viene sempre perso quando il daemon viene arrestato. Tutte le pubblicazioni non conservate vengono perse.

Impostare l'opzione di configurazione del daemon, `Retained_persistenza` su `true`, per salvare periodicamente le pubblicazioni conservate in un file. Quando il daemon viene riavviato, le pubblicazioni conservate che sono state salvate automaticamente per l'ultima volta vengono ripristinate. Per impostazione predefinita, i messaggi conservati creati dai client non vengono ripristinati al riavvio del daemon.

Impostare l'opzione di configurazione daemon, `Retained_persistence` su `true`, per salvare le sottoscrizioni create periodicamente in una sessione persistente in un file. Se `Retained_persistence` è impostato su `true`, le sottoscrizioni che i client creano in una sessione con `CleanSession` impostato su `false`, una "sessione persistente", vengono ripristinate. Il daemon ripristina le sottoscrizioni al riavvio, che iniziano a ricevere le pubblicazioni. Il client riceve le pubblicazioni quando viene riavviato con `CleanSession` in `false`. Per impostazione predefinita, lo stato della sessione client non viene salvato quando un daemon viene arrestato e quindi le sottoscrizioni non vengono ripristinate, anche se il client imposta `CleanSession` su `false`.

`Retained_persistence` è un meccanismo di salvataggio automatico. Potrebbe non salvare le pubblicazioni o le sottoscrizioni conservate più recenti. È possibile modificare la frequenza con cui vengono salvate le pubblicazioni e le sottoscrizioni conservate. Impostare l'intervallo tra i salvataggi o il numero di modifiche tra salvataggi, utilizzando le opzioni di configurazione `autosave_on_changes` e `autosave_interval`.

Configurazione di esempio per l'impostazione della persistenza

```
# 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
```

Daemon WebSphere MQ Telemetry per la sicurezza dei dispositivi

Il daemon WebSphere MQ Telemetry per i dispositivi può autenticare i client che si collegano ad esso, utilizzare credenziali per connettersi ad altri broker e controllare l'accesso agli argomenti. La sicurezza fornita dal daemon è limitata dall'utilizzo del client C WebSphere MQ Telemetry , che non fornisce il supporto SSL. Di conseguenza, le connessioni al e dal daemon non vengono codificate e non possono essere autenticate utilizzando i certificati.

Per impostazione predefinita, non è attivata alcuna sicurezza.

Autenticazione dei client

I clienti MQTT possono impostare un nome utente e una password utilizzando i metodi `MqttConnectOptions.setUserName` e `MqttConnectOptions.setPassword`.

Autenticare un cliente che si connette al daemon controllando il nome utente e la password forniti da un cliente rispetto alle voci nel file delle password. Per abilitare l'autenticazione, creare un file di password e impostare il parametro `password_file` nel file di configurazione daemon; consultare password_file.

Impostare il parametro `allow_anonymous` nel file di configurazione daemon per consentire ai client che si collegano senza nomi utente o password di connettersi a un daemon che sta controllando l'autenticazione; consultare allow_anonymous. Se un client non fornisce un nome utente o una password, viene sempre controllato rispetto al file di password, se il parametro `password_file` è impostato.

Impostare il parametro `clientid_prefixes` nel file di configurazione daemon per limitare le connessioni a client specifici. I client devono avere `clientIdentifiers` che iniziano con uno dei prefissi elencati nel parametro `clientid_prefixes` ; consultare clientid_prefixes.

Sicurezza della connessione bridge

Ogni daemon WebSphere MQ Telemetry per la connessione bridge dei dispositivi è un client MQTT V3 . È possibile impostare il nome utente e la password per ogni connessione bridge come parametro di connessione bridge nel file di configurazione daemon; consultare username e password. Un bridge può quindi autenticarsi su un broker.

Controllo accessi degli argomenti

Se i client sono in fase di autenticazione, il daemon può anche fornire l'accesso di controllo agli argomenti per ciascun utente. Il daemon concede il controllo dell'accesso in base alla corrispondenza dell'argomento su cui un client sta effettuando la pubblicazione o la sottoscrizione con una stringa di argomenti di accesso nel file di controllo dell'accesso; consultare acl_file.

L'elenco di controllo accessi è diviso in due parti. La prima parte controlla l'accesso per tutti i client, inclusi quelli anonimi. La seconda parte contiene una sezione per qualsiasi utente nel file di password. Elenca il controllo accessi specifico per ciascun utente.

Esempio

I parametri di sicurezza sono riportati nel seguente esempio.

```
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. File di configurazione daemon

```
Fred:Fredpassword
Barney:Barneypassword
```

Figura 43. File di password, 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. File di controllo accessi, acl.txt

Risoluzione dei problemi dei client MQTT

Ricercare un'attività di risoluzione dei problemi per risolvere un problema con l'esecuzione dei client MQTT .

Attività correlate

“Traccia e debug del client Java MQTT (Paho)” a pagina 169

Il programma di registrazione predefinito utilizza la funzione di registrazione Java standard nota come `java.util.logging` (JSR47). È possibile configurarlo utilizzando un file di configurazione o in modo programmatico.

“Traccia del client MQTT JavaScript” a pagina 171

È possibile utilizzare il client JavaScript per raccogliere la traccia modificando l'applicazione Web client per richiamare i metodi sull'oggetto client connesso.

“Traccia del servizio di telemetria (MQXR)” a pagina 165
Seguire queste informazioni per avviare una traccia del servizio di telemetria, impostare i parametri che controllano la traccia e trovare l'emissione della traccia.

“Traccia del client Java MQTT v3” a pagina 166
Seguire queste istruzioni per creare una traccia client Java MQTT e controllarne l'output.

“Traccia del client MQTT per C” a pagina 168
Imposta la variabile di ambiente `MQTT_C_CLIENT_TRACE` per tracciare un'applicazione C del client MQTT

“Risoluzione del problema: il client MQTT non si connette” a pagina 178
Risolvere il problema di un programma client MQTT che non riesce a collegarsi al servizio di telemetria (MQXR).

“Risoluzione del problema: connessione client MQTT interrotta” a pagina 180
Scopri cosa sta causando la generazione di eccezioni `ConnectionLost` non previste da parte di un client dopo la connessione e l'esecuzione per un periodo di tempo breve o lungo.

“Risoluzione del problema: messaggi persi in un'applicazione MQTT” a pagina 181
Risolvere il problema di perdita di un messaggio. Il messaggio non è persistente, è stato inviato nel posto sbagliato o non è mai stato inviato? Un programma client codificato in modo errato potrebbe perdere i messaggi.

“Risoluzione del problema: il servizio MQXR (Telemetry) non viene avviato” a pagina 183
Risolvere il problema del mancato avvio del servizio di telemetria (MQXR). Controllare l'installazione di WebSphere MQ Telemetry e non ci sono file mancanti, spostati o con autorizzazioni errate. Controllare i percorsi utilizzati dal servizio di telemetria (MQXR) per individuare i programmi di servizio di telemetria (MQXR).

“Risoluzione del problema: modulo di login di JAAS non richiamato dal servizio di telemetria” a pagina 184
Verificare se il modulo di login JAAS non viene richiamato dal servizio di telemetria (MQXR) e configurare JAAS per correggere il problema.

“Risoluzione del problema: avvio o esecuzione del daemon” a pagina 187
Consultare il daemon IBM WebSphere MQ Telemetry per il log della console dei dispositivi, attivare la traccia o utilizzare la tabella dei sintomi in questo argomento per risolvere i problemi relativi al daemon.

“Risoluzione del problema: i client MQTT non si collegano al daemon” a pagina 188
I client non si collegano al daemon oppure il daemon non si collega ad altri daemon o a un canale di telemetria WebSphere MQ .

Riferimenti correlati

“Ubicazione dei log di telemetria, dei log degli errori e dei file di configurazione” a pagina 162
Individuare i log, i log degli errori e i file di configurazione utilizzati da IBM WebSphere MQ Telemetry.

“Codici di errore del client Java MQTT v3” a pagina 164
Ricercare le cause dei codici di errore in un'eccezione o in un throwable del client Java MQTT v3 .

“Requisiti di sistema per l'utilizzo delle suite di cifratura SHA-2 con client MQTT” a pagina 172
Per Java 6 da IBM, SR13 in poi, è possibile utilizzare le suite di cifratura SHA-2 per proteggere i canali MQTT e le applicazioni client. Tuttavia, le suite di cifratura SHA-2 non sono abilitate per impostazione predefinita fino a Java 7 da IBM, SR4 in poi, quindi nelle versioni precedenti è necessario specificare la suite richiesta. Se si sta eseguendo un client MQTT con il proprio JRE, è necessario assicurarsi che supporti le suite di cifratura SHA-2 . Perché le tue applicazioni client utilizzino le suite di cifratura SHA-2 , il client deve impostare anche il contesto SSL su un valore che supporta TLS (Transport Layer Security) versione 1.2.

“Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL” a pagina 173
Ci sono differenze nella capacità tra browser diversi, su piattaforme diverse. La comprensione di tali differenze ti aiuta a configurare le tue app, le autorità di certificazione (CA) e i certificati client per la connessione utilizzando MQTT messaging client per JavaScript su SSL e WebSockets.

Ubicazione dei log di telemetria, dei log degli errori e dei file di configurazione

Individuare i log, i log degli errori e i file di configurazione utilizzati da IBM WebSphere MQ Telemetry.

Nota: Gli esempi sono codificati per Windows. Modificare la sintassi per eseguire gli esempi su Linux

Log lato server

La procedura guidata di installazione per IBM WebSphere MQ Telemetry scrive i messaggi nel log di installazione:

```
WMQ program directory\mqxr
```

Il servizio di telemetria (MQXR) scrive i messaggi nel log degli errori del gestore code WebSphere MQ e i file FDC nella directory degli errori IBM WebSphere MQ :

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

Scrive anche un log per il servizio di telemetria (MQXR). Il file di log visualizza le proprietà con cui è stato avviato il servizio e gli errori rilevati che fungono da proxy per un client MQTT. Ad esempio, l'annullamento della sottoscrizione da una sottoscrizione che il client non ha creato. Il percorso del log è:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

La configurazione di esempio di telemetria IBM WebSphere MQ creata da IBM WebSphere MQ Explorer avvia il servizio di telemetria utilizzando il comando **runMQXRService**. **runMQXRService** è in *WMQ Telemetry install directory*\bin. Scrive in:

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Modificare **runMQXRService** per visualizzare i percorsi configurati per il servizio di telemetria (MQXR) o per ripetere l'inizializzazione prima di avviare il servizio di telemetria (MQXR).

File di configurazione lato server

Canali di telemetria e servizio di telemetria (MQXR)

Limitazione: Il formato, l'ubicazione, il contenuto e l'interpretazione del file di configurazione del canale di telemetria potrebbero cambiare nelle release future. È necessario utilizzare Esplora risorse di IBM WebSphere MQ per configurare i canali di telemetria.

IBM WebSphere MQ Explorer salva le configurazioni di telemetria nel file `mqxr_win.properties` in Windows e il file `mqxr_unix.properties` in Linux. I file delle proprietà vengono salvati nella directory di configurazione della telemetria:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Figura 45. Directory di configurazione della telemetria su Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figura 46. Directory di configurazione della telemetria su Linux

JVM

Impostare le proprietà Java passate come argomenti al servizio di telemetria (MQXR) nel file `java.properties`. Le proprietà nel file vengono trasmesse direttamente alla JVM che esegue il servizio di telemetria (MQXR). Vengono passate come proprietà JVM aggiuntive sulla riga comandi

Java. Le proprietà impostate sulla riga comandi hanno la precedenza sulle proprietà aggiunte alla riga comandi dal file `java.properties` .

Individuare il file `java.properties` nella stessa cartella delle configurazioni di telemetria, consultare Figura 45 a pagina 162 e Figura 46 a pagina 162.

Modificare `java.properties` specificando ciascuna proprietà come riga separata. Formattare ciascuna proprietà esattamente come si farebbe per passare la proprietà alla JVM come argomento; ad esempio:

```
-Xmx1024m
-Xms1024m
```

JAAS

Il file di configurazione JAAS è descritto in Configurazione JAAS del canale di telemetria , che comprende il file di configurazione JAAS di esempio, JAAS.config, fornito con IBM WebSphere MQ Telemetry.

Se si configura JAAS, si scriverà quasi certamente una classe per autenticare gli utenti per sostituire le procedure standard di autenticazione JAAS .

Per includere la classe `Login` nel percorso classe utilizzato dal percorso classe del servizio di telemetria (MQXR), fornire un file di configurazione WebSphere MQ `service.env` .

Impostare il percorso classe per JAAS `LoginModule` in `service.env`. Non è possibile utilizzare la variabile `%classpath%` in `service.env`. Il percorso classe in `service.env` viene aggiunto al percorso classe già impostato nella definizione del servizio di telemetria (MQXR).

Visualizzare i percorsi classe utilizzati dal servizio di telemetria (MQXR) aggiungendo `echo set classpath` a `runMQXRService.bat`. L'output viene inviato a `mqxr.stdout`.

L'ubicazione predefinita per il file `service.env` è:

```
WMQ data directory\service.env
```

Sovrascrivere queste impostazioni con un file di `service.env` per ogni gestore code in:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

Figura 47 a pagina 163 visualizza un `service.env` file di esempio per utilizzare il `LoginModule.classd`i esempio.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Nota: `service.env` non deve contenere alcuna variabile. Sostituire il valore effettivo di *`WMQ Install Directory`*.

Figura 47. Esempio `service.env` per Windows

Trace

Un tecnico dell'assistenza IBM potrebbe richiedere di configurare la traccia; consultare “Traccia del servizio di telemetria (MQXR)” a pagina 165. i parametri per la traccia configurata sono memorizzati in due file:

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

File di log lato client

La classe di persistenza file predefinita nel client Java SE MQTT fornito con IBM WebSphere MQ Telemetry crea una cartella con il nome: *clientIdentifier*`-tcp`*hostNameport* o *clientIdentifier*`-ssl`*hostNameport* nella directory di lavoro del client. Il nome della cartella indica il `nome host` e la `porta` utilizzati nel tentativo di connessione. La cartella contiene i messaggi che sono stati memorizzati dalla classe di persistenza. I messaggi vengono eliminati quando sono stati consegnati correttamente.

La cartella viene eliminata quando un client, con una sessione pulita, termina.

Se la traccia del client è attivata, il log non formattato viene, per impostazione predefinita, memorizzato nella directory di lavoro del client. Il file di traccia è denominato `mqtt-n.trc`

File di configurazione lato client

Impostare le proprietà di traccia e SSL per il client Java MQTT utilizzando i file delle proprietà Java o impostare le proprietà in modo programmatico. Passare le proprietà al client MQTT Java utilizzando lo switch JVM `-D` : ad esempio,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Consultare “Traccia del client Java MQTT v3” a pagina 166.Per i collegamenti alla documentazione API client per le librerie client MQTT, vedere Riferimento di programmazione client MQTT.

Codici di errore del client Java MQTT v3

Ricercare le cause dei codici di errore in un'eccezione o in un throwable del client Java MQTT v3 .

Tabella 5. Codici di errore del client Java MQTT v3

Codice di errore	Valore	Causa
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	Il client è già connesso.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	Il client è già disconnesso.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Generata quando è stato effettuato un tentativo di richiamare `MqttClient.disconnect` dall'interno di un metodo su `MqttCallback`.
REASON_CODE_CLIENT_DISCONNECTING	32102	Il client è attualmente in fase di disconnessione e non può accettare alcun nuovo lavoro.
REASON_CODE_CLIENT_EXCEPTION	0	Il client ha rilevato una eccezione.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	Il client non è connesso al server.
REASON_CODE_CLIENT_TIMEOUT	32000	Timeout del client durante l'attesa di una risposta dal server.
REASON_CODE_FAILED_AUTHENTICATION	4	L'autenticazione con il server non è riuscita a causa di un nome utente o di una password non validi.
REASON_CODE_INVALID_CLIENT_ID	2	Il server ha rifiutato l'ID client fornito.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	La versione del protocollo richiesta non è supportata dal server.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Errore interno, causato da nessun nuovo ID messaggio disponibile.
REASON_CODE_NOT_AUTHORIZED	5	Non si è autorizzati ad eseguire l'operazione richiesta.

Tabella 5. Codici di errore del client Java MQTT v3 (Continua)		
Codice di errore	**Valore**	**Causa**
REASON_CODE_SERVER_CONNECT_ERROR	32103	Impossibile connettersi al server.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	L'URI del server e SocketFactory forniti non corrispondono.
REASON_CODE_SSL_CONFIG_ERROR	32106	Errore di configurazione SSL.
REASON_CODE_UNEXPECTED_ERROR	6	Si è verificato un errore non previsto.

Traccia del servizio di telemetria (MQXR)

Seguire queste informazioni per avviare una traccia del servizio di telemetria, impostare i parametri che controllano la traccia e trovare l'emissione della traccia.

Prima di iniziare

La traccia è una funzione di supporto. Seguire queste istruzioni se un tecnico del servizio IBM richiede di tracciare il servizio di telemetria (MQXR). La documentazione del prodotto non documenta il formato del file di traccia o come utilizzarlo per il debug di un client.

Informazioni su questa attività

È possibile utilizzare i comandi IBM WebSphere MQ **strmqtrc** e **endmqtrc** per avviare e arrestare la traccia IBM WebSphere MQ . **strmqtrc** cattura la traccia per il servizio di telemetria (MQXR). Quando si usa **strmqtrc** , si verifica un ritardo fino a un paio di secondi prima dell'avvio della traccia del servizio di telemetria. Per ulteriori informazioni sulla traccia di IBM WebSphere MQ , consultare Utilizzo della traccia. In alternativa, è possibile tracciare il servizio di telemetria (MQXR) utilizzando la seguente procedura:

Procedura

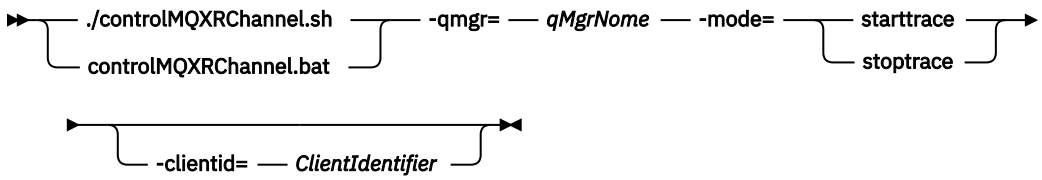
1. Impostare le opzioni di traccia per controllare la quantità di dettaglio e la dimensione della traccia. Le opzioni si applicano a una traccia avviata con il comando **strmqtrc** o **controlMQXRChannel** .

   Impostare le opzioni di traccia nei file seguenti:

   ```
   mqxrtrace.properties
   trace.config
   ```

   I file si trovano nella directory:

   - Su Windows, *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr.
   - Su Linux, var/mqm/qmgrs/ *qMgrName*/mqxr.
2. Aprire una finestra comandi nella seguente directory:
   - Su sistemi Windows , *WebSphere MQ installation directory*\mqxr\bin .
   - Su sistemi Linux /opt/mqm/mqxr/bin.
3. Immettere il comando riportato di seguito per avviare una traccia SYSTEM.MQXR.SERVICE :



Parametri obbligatori

qmgr=*qmgrName*
Impostare *qmgrName* sul nome del gestore code

mode=starttrace| stoptrace
Impostare starttrace per iniziare la traccia o su stoptrace per terminare la traccia

Parametri facoltativi

clientid=*ClientIdentifier*
Impostare *ClientIdentifier* su `ClientIdentifier` di un client. clientid filtra la traccia su un singolo client. Eseguire il comando di traccia più volte per tracciare più client.

Ad esempio:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

Risultati

Per visualizzare l'output di traccia, andare alla seguente directory:

- Su Windows , *WebSphere MQ data directory*\trace.
- Su Linux, /var/mqm/trace .

I file di traccia sono denominati mqxr_PPPPP.trc, dove PPPPP è l'ID processo.

Riferimenti correlati

strmqtrc

Traccia del client Java MQTT v3

Seguire queste istruzioni per creare una traccia client Java MQTT e controllarne l'output.

Prima di iniziare

Questo argomento è applicabile solo a IBM WebSphere MQ versione 7.5.0.0. Per informazioni sulla traccia del client Java per versioni successive, consultare “Traccia e debug del client Java MQTT (Paho)” a pagina 169.

La traccia è una funzione di supporto. Segui queste istruzioni se un tecnico del servizio IBM ti chiede di tenere traccia del tuo client MQTT Java. La documentazione del prodotto non documenta il formato del file di traccia o come utilizzarlo per il debug di un client.

La traccia funziona solo per il client Java WebSphere MQ Telemetry .

Informazioni su questa attività

Nota: Gli esempi sono codificati per Windows. Modificare la sintassi per eseguire gli esempi su Linux[2].

Procedura

1. Creare un file delle proprietà Java contenente la configurazione della traccia.

   Nel file delle proprietà, specificare le seguenti proprietà facoltative. Se una chiave della proprietà viene specificata più di una volta, l'ultima ricorrenza imposta la proprietà.

   a) `com.ibm.micro.client.mqttv3.trace.outputName`

      La directory in cui scrivere il file di traccia. Il valore predefinito è la directory di lavoro del client. Il file di traccia è denominato mqtt-*n*.trc.

[2] Java utilizza il delimitatore di percorso corretto. È possibile codificare il delimitatore in un file delle proprietà come '/' o '\\'; '\' è il carattere escape

```
java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
```

b) `com.ibm.micro.client.mqttv3.trace.count`

Il numero di file di traccia da scrivere. Il valore predefinito è un file di dimensione illimitata.

```
java com.ibm.micro.client.mqttv3.trace.count=5
```

c) `com.ibm.micro.client.mqttv3.trace.limit`

La dimensione massima del file da scrivere, il valore predefinito è 500000. Il limite si applica solo se è richiesto più di un file di traccia.

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

Attivare o disattivare la traccia, per client. Se *`clientIdentifier`*=*, la traccia è attivata o disattivata per tutti i client. Per impostazione predefinita, la traccia è disattivata per tutti i client.

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. Passare il file delle proprietà di traccia alla JVM utilizzando una proprietà di sistema.

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. Eseguire il client.
4. Convertire il file di traccia dalla codifica binaria in testo o `.html`. Utilizzare il seguente comando:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

dove gli argomenti sono:

-?
visualizza la guida

-i traceFile
Obbligatorio. Passa al file di input (ad esempio, `mqtt-0.trc`).

-o outputFile
Obbligatorio. Definisce il file di output (ad esempio, `mqtt-0.trc.html` o `mqtt-0.trc.txt`).

-h
Output come HTML. L'estensione dei file di output deve essere `.html`. Se non viene specificato, l'output è testo semplice.

-d time
Rientra una riga con * se la differenza di tempo in millisecondi è maggiore o uguale all'ora (> =). Non applicabile per l'output HTML.

Il seguente esempio emetterà il file di traccia in formato HTML

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

Il secondo esempio emetterà il file di traccia come testo semplice, con qualsiasi data / ora consecutiva con millisecondi con una differenza di 50 o superiore rientrata con un asterisco (*).

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

L'esempio finale emetterà il file di traccia come testo semplice:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Traccia del client MQTT per C

Imposta la variabile di ambiente MQTT_C_CLIENT_TRACE per tracciare un'applicazione C del client MQTT

Prima di iniziare

Il client MQTT per la traccia C è disponibile sia per il client pre - creato Windows che per il client Linux MQTT per le librerie C e per le librerie iOS create dall'utente.

Informazioni su questa attività

Impostare la variabile di ambiente MQTT_C_CLIENT_TRACE su un percorso per un file che deve contenere l'output di traccia. L'output di traccia viene scritto nel file.

Procedura

Impostare MQTT_C_CLIENT_TRACE=mqttcclient.log prima di eseguire l'applicazione C client MQTT .

a) Ad esempio, modificare lo script di esempio in “Introduzione al client MQTT per C” a pagina 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) Eseguire lo script dalla directory %sdkroot%/sdk/client/c/samples .

Risultati

I file di output di traccia iniziano con le seguenti righe:

```
============================================================
                    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
```

Attività correlate

“Introduzione al client MQTT per C” a pagina 26
Introduzione e esecuzione con il client MQTT di esempio per C su qualsiasi piattaforma su cui è possibile compilare l'origine C. Verificare che sia possibile eseguire il client MQTT di esempio per C con IBM MessageSight o IBM WebSphere MQ come server MQTT.

Traccia e debug del client Java MQTT (Paho)

Il programma di registrazione predefinito utilizza la funzione di registrazione Java standard nota come `java.util.logging` (JSR47). È possibile configurarlo utilizzando un file di configurazione o in modo programmatico.

Informazioni su questa attività

Nota: Il client Paho Java è applicabile solo alle versioni di IBM WebSphere MQ 7.5.0.1 e successive. Per informazioni sulla traccia del client Java in IBM WebSphere MQ versione 7.5.0.0, consultare “Traccia del client Java MQTT v3” a pagina 166.

Nota: La traccia è una funzione di supporto. Attieniti a queste istruzioni se un tecnico del servizio IBM ti chiede di tenere traccia del tuo client Java MQTT . La documentazione del prodotto non documenta il formato del file di traccia o come utilizzarlo per il debug di un client. La traccia funziona solo per il client IBM WebSphere MQ Telemetry Java.

Il metodo più semplice per utilizzare un file di configurazione è specificarne il nome nella proprietà `java.util.logging.config.file`.

Un file delle propriet ... di lavoro `jsr47min.properties` viene fornito nel pacchetto `org.eclipse.paho.client.mqttv3.logging`

La funzione di registrazione JSR47 può essere utilizzata in diversi modi:

- Per raccogliere messaggi da una serie selezionata di pacchetti
- Per raccogliere i messaggi da un livello di log e al di sotto di tale livello
- Per scegliere più destinazioni per i messaggi di log
- Fornendo un programma di registrazione integrato che scrive in un file e controlla la dimensione e il numero di file utilizzati
- Fornendo un programma di registrazione integrato che scrive in memoria e abilita la scrittura dei messaggi in memoria in base a un trigger
- Se anche l'applicazione che utilizza la libreria del client MQTT viene strumentata utilizzando JSR47, i messaggi dell'applicazione e della libreria del client vengono mescolati

Viene fornita una classe di utilità per raccogliere le informazioni di debug. Questa classe include i messaggi di log e di traccia descritti in precedenza, ma può raccogliere informazioni quali le proprietà del sistema Java e il valore di variabili dall'interno del client Paho.

La funzione di debug viene fornita nella classe pubblica `Debug`, che fa parte del pacchetto `org.eclipse.paho.client.mqttv3.util`. Un'istanza di debug può essere ottenuta utilizzando il metodo getDebug() su entrambi gli oggetti client MQTT asincroni e sincroni.

Ad esempio:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Il metodo dumpClientDebug () esegue il dump della quantità massima di informazioni di debug. La funzione di registrazione deve essere abilitata per catturare le informazioni di debug complete, che vengono scritte su di essa. Per acquisire le informazioni di debug complete, richiamare un metodo dump quando il problema si verifica, ad esempio dopo che si è verificata una particolare eccezione.

Procedura

1. Creare un file di configurazione o utilizzare il file jsr47min.properties fornito.

   Se si utilizza il file delle proprietà fornito, verificare che il trigger push sia impostato sul livello di errore corretto. Per impostazione predefinita, questo valore è impostato su un errore di livello Grave, ma potrebbe essere necessario scrivere continuamente la traccia nel file piuttosto che conservarla in memoria fino a quando non si verifica un errore. Per effettuare questa operazione, modificare:

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   a

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. Inoltrare il file di configurazione della traccia alla JVM utilizzando una proprietà di sistema.

   Se si sta utilizzando il file jsr4min.properties :

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Eseguire il client.

Risultati

Quando si verifica un'eccezione o un problema, la classe di debug Paho scrive la traccia in memoria nella destinazione del file configurato.

La traccia non viene scritta automaticamente nel file come viene generata, ciò si verifica solo quando viene raggiunto il trigger di inserimento o quando la classe di debug fa sì che la traccia venga scritta. Quest' ultimo potrebbe richiedere delle modifiche al codice dell'applicazione.

Ogni riga viene scritta nel gestore file man mano che viene creata. È possibile controllare il formato in cui vengono scritti i messaggi configurando un FileHandler. Un gestore file personalizzato viene fornito con Paho che scrive più di SimpleHandler e meno di XMLHandler fornito con JRE. I record di traccia che utilizzano il programma di formattazione log Paho sono nel formato seguente:

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Esempio

Viene fornito un file delle proprietà di lavoro `jsr47min.properties` . Questo file contiene una configurazione consigliata per la raccolta della traccia che consente di risolvere i problemi relativi al client Paho MQTT. Configura la traccia per essere continuamente raccolta in memoria con un impatto minimo sulle prestazioni. Quando si verifica il trigger di inserimento o viene effettuata una richiesta specifica di inserimento, la traccia in memoria viene inviata al gestore di destinazione configurato.
Il trigger di push predefinito è un messaggio di livello Grave, che è una connessione interrotta. Per impostazione predefinita, la traccia raccolta in memoria viene scritta nel file specificato a questo punto. Per impostazione predefinita, questo file è lo standard `java.util.logging.FileHandler`. È possibile utilizzare la classe Debug Paho per inviare la traccia di memoria alla sua destinazione.

I dettagli completi di JSR47 sono disponibili in Javadoc per pacchetto 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
```

Operazioni successive

Per raccogliere la traccia in modo programmatico, viene fornita una classe di utilità che consente di raccogliere le informazioni di debug. Questa classe include i messaggi di log e di traccia descritti in precedenza, ma può raccogliere informazioni quali le proprietà del sistema Java e il valore di variabili dall'interno del client Paho.

La funzione di debug viene fornita nella classe pubblica Debug, che fa parte del pacchetto `org.eclipse.paho.client.mqttv3.util`. Un'istanza di debug può essere ottenuta utilizzando il metodo getDebug() su entrambi gli oggetti client MQTT asincroni e sincroni.

Ad esempio:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Il metodo dumpClientDebug () esegue il dump della quantità massima di informazioni di debug. La funzione di registrazione deve essere abilitata per catturare le informazioni di debug complete, che vengono scritte su di essa. Per acquisire le informazioni di debug complete, richiamare un metodo dump quando il problema si verifica, ad esempio dopo che si è verificata una particolare eccezione.

Traccia del client MQTT JavaScript

È possibile utilizzare il client JavaScript per raccogliere la traccia modificando l'applicazione Web client per richiamare i metodi sull'oggetto client connesso.

Informazioni su questa attività

Per raccogliere la traccia, è possibile utilizzare i seguenti metodi:

- `client.startTrace()` avvia la traccia per il client.
- `client.stopTrace()` arresta la traccia per il client.
- `client.getTraceLog()` restituisce il buffer di traccia corrente.

È possibile emettere il buffer di traccia da inviare al supporto software IBM . Esistono diversi modi per farlo. L'esempio mostra la traccia in fase di avvio, quindi l'output inviato alla console e a un indirizzo email specificato e infine la traccia in fase di arresto.

```
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();
 };
```

Output di esempio:

```
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 Requisiti di sistema per l'utilizzo delle suite di cifratura SHA-2 con client MQTT

Per Java 6 da IBM, SR13 in poi, è possibile utilizzare le suite di cifratura SHA-2 per proteggere i canali MQTT e le applicazioni client. Tuttavia, le suite di cifratura SHA-2 non sono abilitate per impostazione predefinita fino a Java 7 da IBM, SR4 in poi, quindi nelle versioni precedenti è necessario specificare la suite richiesta. Se si sta eseguendo un client MQTT con il proprio JRE, è necessario assicurarsi che supporti le suite di cifratura SHA-2 . Perché le tue applicazioni client utilizzino le suite di cifratura SHA-2 , il client deve impostare anche il contesto SSL su un valore che supporta TLS (Transport Layer Security) versione 1.2.

Per Java 7 da IBM, SR4 in poi, le suite di cifratura SHA-2 sono abilitate per impostazione predefinita. Per Java 6 da IBM, SR13 e release di servizio successivi, se si definisce un canale MQTT senza specificare una suite di cifratura, il canale non accetterà le connessioni da un client che utilizza una suite di cifratura SHA-2 . Per utilizzare le suite di cifratura SHA-2 , è necessario specificare la suite richiesta nella definizione del canale. In questo modo il server MQTT abilita la suite prima di effettuare le connessioni. Significa anche che solo le applicazioni client che utilizzano la suite specificata possono connettersi a questo canale.

Esiste una limitazione simile per il Client MQTT per Java. Se il codice client è in esecuzione su un JRE Java 1.6 da IBM, le suite di cifratura SHA-2 richieste devono essere abilitate esplicitamente. Per utilizzare

queste suite, il client deve anche impostare il contesto SSL su un valore che supporta la Versione 1.2 del protocollo TLS (Transport Layer Security). Ad esempio:

```
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);
```

A giugno 2013, Internet Explorer 10 è l'unico browser che funziona con MQTT messaging client per JavaScript e supporta anche il protocollo TLS 1.2 , quindi è l'unico browser che è possibile utilizzare se si desidera creare connessioni SHA-2 con il client JavaScript .

Per un elenco delle suite di cifratura attualmente supportate, consultare i link correlati.

Concetti correlati

“Configurazione client MQTT per l'autenticazione client mediante SSL” a pagina 104
Per autenticare il client MQTT utilizzando SSL, il client si connette a un canale di telemetria utilizzando SSL. Deve specificare una porta TCP che corrisponde a un canale di telemetria configurato per autenticare i client SSL.

“Configurazione del client MQTT per l'autenticazione di canale mediante SSL” a pagina 107
Per autenticare il canale di telemetria utilizzando SSL, il client deve connettersi al canale di telemetria utilizzando SSL. Deve specificare una porta che corrisponde a un canale di telemetria configurato per SSL. La configurazione deve includere un keystore protetto da passphrase che contenga il certificato digitale firmato privatamente del server.

V 7.5.0.1 Restrizioni nel supporto del browser per le app web di messaggistica mobile su SSL

Ci sono differenze nella capacità tra browser diversi, su piattaforme diverse. La comprensione di tali differenze ti aiuta a configurare le tue app, le autorità di certificazione (CA) e i certificati client per la connessione utilizzando MQTT messaging client per JavaScript su SSL e WebSockets.

La messaggistica mobile che utilizza JavaScript su SSL è abbastanza nuova, quindi non sorprende che diverse combinazioni di browser e piattaforme abbiano implementato la funzionalità in modi leggermente diversi e in diverse estensioni. La seguente tabella fornisce una panoramica di ciò che attualmente funziona e non funziona per ogni combinazione di browser (Firefox, Chrome, Internet Explorere Safari) e piattaforma (Windows, Linux, Mac, iOSe Android).

Tabella 6. Supporto SSL per piattaforma e browser. Per ogni combinazione di browser e piattaforma, la tabella specifica se le connessioni SSL anonime e non anonime sono supportate e la misura in cui il browser funziona con tutte le CA (Certificate Authority) e i certificati del client.

Browser	**Supporto SSL (S/N)**	**SSL funziona con qualsiasi CA (Y/N)**	**Ulteriori informazioni**
Desktop Firefox .	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - Sì	Aggiungere il certificato CA e client al browser. Firefox utilizza la propria memorizzazione certificato. Per importare un certificato CA, fare clic su **Strumenti** > **Opzioni** > **Avanzate** > **Codifica** > **Visualizza certificati** > **Autorità** > **Importa** Per importare un certificato client, fare clic su **Strumenti** > **Opzioni** > **Avanzate** > **Crittografia** > **Visualizza certificati** > **Certificati** > **Importa** Per attivare una connessione sicura, specificare `https://` nell'URL. Firefox ti dà la possibilità di selezionare un certificato automaticamente o chiedendoti ogni volta. Firefox ti offre anche l'opzione di utilizzare SSL 3.0 o TLS 1.0; assicurati che entrambi siano selezionati.

Tabella 6. Supporto SSL per piattaforma e browser. Per ogni combinazione di browser e piattaforma, la tabella specifica se le connessioni SSL anonime e non anonime sono supportate e la misura in cui il browser funziona con tutte le CA (Certificate Authority) e i certificati del client. *(Continua)*

Browser	**Supporto SSL (S/N)**	**SSL funziona con qualsiasi CA (Y/N)**	**Ulteriori informazioni**
Desktop Chrome .	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - Sì	Utilizzare il browser per aggiungere il certificato CA e client all'archivio certificati del sistema operativo, condiviso con altri software. Per importare un certificato CA, fare clic su **Impostazioni** > **Mostra impostazioni avanzate** > **Gestisci certificati** > **Autorità di certificazione root attendibili** > **Importa** Per importare un certificato client, fare clic su **Impostazioni** > **Mostra impostazioni avanzate** > **Gestisci certificati** > **Personale** > **Importa** Per attivare una connessione sicura, specificare `https://` nell'URL. Chrome richiede diverse opzioni; selezionare quella corretta, a seconda se si sta configurando una connessione anonima o non anonima.

Tabella 6. Supporto SSL per piattaforma e browser. Per ogni combinazione di browser e piattaforma, la tabella specifica se le connessioni SSL anonime e non anonime sono supportate e la misura in cui il browser funziona con tutte le CA (Certificate Authority) e i certificati del client. *(Continua)*

Browser	**Supporto SSL (S/N)**	**SSL funziona con qualsiasi CA (Y/N)**	**Ulteriori informazioni**
Internet Explorer.	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - Sì	Quando si effettua una connessione SSL non anonima, viene richiesto di scegliere il certificato client corretto. Internet Explorer utilizza l'archivio certificati Windows , condiviso con altri software. Per importare un certificato CA, fare clic su **Strumenti** > **Opzioni Internet** > **Contenuto** > **Certificati** > **Autorità di certificazione root attendibili** > **Importa** Per importare un certificato client, fare clic su **Strumenti** > **Opzioni Internet** > **Contenuto** > **Certificati** > **Personale** > **Importa**
Desktop Safari .	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - Sì	Utilizzare il browser per aggiungere il certificato CA e client all'archivio certificati del sistema operativo, condiviso con altri software.
Firefox su Android	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - No	Non anonimo: I certificati del client non funzionano, poiché non puoi soddisfare il requisito di aggiungere la tua CA all'elenco in Firefox. Per importare un certificato client, fare clic su **Impostazioni** > **Sicurezza** > **Memoria credenziali**. Se il certificato è firmato da una CA attendibile nell'elenco, è possibile stabilire una connessione sicura.

Tabella 6. Supporto SSL per piattaforma e browser. Per ogni combinazione di browser e piattaforma, la tabella specifica se le connessioni SSL anonime e non anonime sono supportate e la misura in cui il browser funziona con tutte le CA (Certificate Authority) e i certificati del client. *(Continua)*

Browser	**Supporto SSL (S/N)**	**SSL funziona con qualsiasi CA (Y/N)**	**Ulteriori informazioni**
Chrome su Android	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - No	Non anonimo: I certificati del client non funzionano, poiché non puoi soddisfare il requisito di aggiungere la tua CA all'elenco in Chrome. **Nota:** Google prevede di supportarlo nella Versione 27 di Chrome. Questo è stato un difetto aperto dalla versione 18. Per importare un certificato client, fare clic su **Impostazioni** > **Sicurezza** > **Memoria credenziali**. Se il certificato è firmato da una CA attendibile nell'elenco, è possibile stabilire una connessione sicura.
Safari su iOS	SSL anonimo - Sì SSL non anonimo - Sì	SSL anonimo - Sì SSL non anonimo - No	Non anonimo: La periferica non considera attendibile il certificato client, anche quando il certificato CA è installato contemporaneamente. Safari utilizza l'archivio certificati della periferica. Per importare in questo negozio, fare clic su **Impostazioni** > **Generale** > **Profilo**e fornire il certificato CA o client da una pagina Web oppure inviarlo via email.

Tabella 6. Supporto SSL per piattaforma e browser. Per ogni combinazione di browser e piattaforma, la tabella specifica se le connessioni SSL anonime e non anonime sono supportate e la misura in cui il browser funziona con tutte le CA (Certificate Authority) e i certificati del client. *(Continua)*

Browser	Supporto SSL (S/N)	SSL funziona con qualsiasi CA (Y/N)	Ulteriori informazioni
Chrome su iOS	SSL anonimo - Sì SSL non anonimo - No	Anonimo SSL - No SSL non anonimo - No	Anonimo: Solo le applicazioni Apple possono accedere all'archivio root di sistema iOS . Pertanto Chrome deve utilizzare il proprio elenco di CA, a cui non è possibile aggiungere. Non anonimo: I certificati client non funzionano, perché non è possibile soddisfare il requisito di aggiungere la propria CA all'elenco.

Attività correlate

“Connessione di MQTT messaging client per JavaScript su SSL e WebSockets” a pagina 77
Connetti la tua applicazione web in modo sicuro a IBM WebSphere MQ utilizzando le pagine HTML di esempio MQTT messaging client per JavaScript con SSL e WebSocket protocol.

Informazioni correlate

Mozilla: (SSL) Firefox utilizza lo storage Android CA o il proprio?

Chromium: issue 134418 - Implementa supporto certificato client

Impossibile aprire il sito https con certificato non attendibile su ie10

Risoluzione del problema: il client MQTT non si connette

Risolvere il problema di un programma client MQTT che non riesce a collegarsi al servizio di telemetria (MQXR).

Prima di iniziare

Il problema è relativo al server, al client o alla connessione? È stato scritto il proprio client di gestione del protocollo MQTT v3 o un'app client MQTT utilizzando i client C o Java WebSphere MQTT?

Eseguire l'applicazione di verifica fornita con WebSphere MQ Telemetry sul server e controllare che il canale di telemetria e il servizio di telemetria (MQXR) siano in esecuzione correttamente. Quindi, trasferire l'applicazione di verifica al client ed eseguire l'applicazione di verifica.

Informazioni su questa attività

Esistono diversi motivi per cui un client MQTT potrebbe non connettersi, o si potrebbe concludere che non si è connesso, al server di telemetria.

Procedura

1. Considerare quali inferenze possono essere tratte dal codice motivo che il servizio di telemetria (MQXR) ha restituito a `MqttClient.Connect`. Che tipo di errore di connessione è?

Opzione	Descrizione
REASON_CODE_INVALID_PROTOCOL_VERSION	Assicurarsi che l'indirizzo del socket corrisponda ad un canale di telemetria e che non sia stato utilizzato lo stesso indirizzo del socket per un altro broker.
REASON_CODE_INVALID_CLIENT_ID	Verificare che l'identificativo client non sia più lungo di 23 byte e contenga solo caratteri dell'intervallo: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	Controllare che l'identificativo client non sia lo stesso del nome gestore code.
REASON_CODE_SERVER_CONNECT_ERROR	Verificare che il servizio MQXR (telemetria) e il gestore code siano in esecuzione normalmente. Utilizzare **netstat** per controllare che l'indirizzo del socket non sia assegnato ad un'altra applicazione.

Se è stata scritta una libreria client MQTT invece di utilizzare una delle librerie fornite da IBM WebSphere MQ Telemetry, esaminare il codice di ritorno `CONNACK` .

Da questi tre errori è possibile dedurre che il client si è collegato al servizio di telemetria (MQXR), ma il servizio ha rilevato un errore.

2. Considerare quali inferenze possono essere tratte dai codici motivo che il client produce quando il servizio di telemetria (MQXR) non risponde:

Opzione	Descrizione
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	Cercare un file FDC sul server; consultare “Log lato server” a pagina 162. Quando il servizio di telemetria (MQXR) rileva il timeout del client, scrive un file FDC (first - failure data capture). Scrive un file FDC ogni volta che la connessione viene inaspettatamente interrotta.

Il servizio di telemetria (MQXR) potrebbe non aver risposto al client e il timeout al client scade. Il client WebSphere MQ Telemetry Java si blocca solo se l'applicazione ha impostato un timeout indefinito. Il client genera una di queste eccezioni dopo che il timeout impostato per `MqttClient.Connect` scade con un problema di collegamento non diagnosticato.

A meno che non si trovi un file FDC correlato all'errore di connessione, non è possibile dedurre che il client abbia tentato di connettersi al server:

a) Confermare che il client ha inviato una richiesta di connessione.

   Controllare la richiesta TCPIP con uno strumento come **tcpmon**, disponibile all'indirizzo https://java.net/projects/tcpmon

b) L'indirizzo del socket remoto utilizzato dal client corrisponde all'indirizzo del socket definito per il canale di telemetria?

   La classe di persistenza file predefinita nel client Java SE MQTT fornito con IBM WebSphere MQ Telemetry crea una cartella con il nome: *clientIdentifier*`-tcp`*hostNameport* o *clientIdentifier*`-ssl`*hostNameport* nella directory di lavoro del client. Il nome della cartella indica il `nome host` e la `porta` utilizzati nel tentativo di connessione; consultare “File di log lato client” a pagina 163.

c) È possibile eseguire il ping dell'indirizzo del server remoto?

d) **netstat** sul server mostra che il canale di telemetria è in esecuzione anche sulla porta a cui si connette il client?

3. Verificare se il servizio di telemetria (MQXR) ha rilevato un problema nella richiesta del client.

   Il servizio di telemetria (MQXR) scrive gli errori rilevati in `mqxr.log`e il gestore code scrive gli errori in `AMQERR01.LOG`; consultare
4. Tentare di isolare il problema eseguendo un altro client.
   - Eseguire l'applicazione di esempio MQTT utilizzando lo stesso canale di telemetria.
   - Eseguire il client della GUI **wmqttSample** per verificare la connessione. Scarica **wmqttSample** scaricando SupportPac IA92.

     Nota: Le versioni precedenti di IA92 non includono la libreria client Java MQTT v3 .

   Eseguire i programmi di esempio sulla piattaforma del server per eliminare le incertezze relative alla connessione di rete, quindi eseguire gli esempi sulla piattaforma client.
5. Altre cose da controllare:
   a) Decine di migliaia di client MQTT stanno tentando di connettersi contemporaneamente?

      I canali di telemetria hanno una coda per memorizzare nel buffer un backlog delle connessioni in entrata. Le connessioni vengono elaborate più di 10.000 al secondo. La dimensione del buffer di backlog è configurabile utilizzando la procedura guidata del canale di telemetria in IBM WebSphere MQ Explorer. La dimensione predefinita è 4096. Verificare che il backlog non sia stato impostato su un valore basso.

   b) Il servizio di telemetria (MQXR) e gestore code sono ancora in esecuzione?

   c) Il client è stato connesso a un gestore code ad alta disponibilità che ha cambiato il proprio indirizzo TCPIP?

   d) Un firewall sta filtrando selettivamente i pacchetti di dati in uscita o restituiti?

Risoluzione del problema: connessione client MQTT interrotta

Scopri cosa sta causando la generazione di eccezioni `ConnectionLost` non previste da parte di un client dopo la connessione e l'esecuzione per un periodo di tempo breve o lungo.

Prima di iniziare

Il client MQTT si è connesso correttamente. Il client potrebbe essere attivo per un lungo periodo di tempo. Se i client vengono avviati solo con un breve intervallo tra loro, il tempo tra la connessione riuscita e la connessione eliminata potrebbe essere breve.

Non è difficile distinguere una connessione eliminata da una connessione che è stata creata correttamente e successivamente eliminata. Una connessione eliminata viene definita dal client MQTT che richiama il metodo `MqttCallback.ConnectionLost` . Il metodo viene richiamato solo dopo che la connessione è stata stabilita correttamente. Il sintomo è diverso dal fatto che `MqttClient.Connect` ha generato un'eccezione dopo aver ricevuto un riconoscimento negativo o un timeout.

Se l'applicazione client MQTT non utilizza le librerie del client MQTT fornite da IBM WebSphere MQ, il sintomo dipende dal client. Nel protocollo MQTT v3 , il sintomo è la mancanza di una risposta tempestiva a una richiesta al server o l'errore della connessione TCP/IP.

Informazioni su questa attività

Il client MQTT richiama `MqttCallback.ConnectionLost` con un'eccezione generabile in risposta a eventuali problemi sul lato server riscontrati dopo aver ricevuto un riconoscimento di connessione positivo. Quando un client MQTT ritorna da `MqttTopic.publish` e `MqttClient.subscribe` , la richiesta viene trasferita a un thread del client MQTT responsabile dell'invio e della ricezione di messaggi. Gli errori lato server vengono riportati in maniera asincrona inoltrando un'eccezione generabile al metodo callback `ConnectionLost` .

Il servizio di telemetria (MQXR) scrive sempre un file di acquisizione dati del primo errore se interrompe la connessione.

Procedura

1. È stato avviato un altro client che ha utilizzato lo stesso `ClientIdentifier`?

   Se un secondo client viene avviato o lo stesso client viene riavviato, utilizzando lo stesso `ClientIdentifier`, la prima connessione al primo client viene eliminata.
2. Il client ha eseguito l'accesso a un argomento per cui non è autorizzato alla pubblicazione o alla sottoscrizione?

   Qualsiasi azione intrapresa dal servizio di telemetria per conto di un client che restituisce `MQCC_FAIL` comporta l'eliminazione della connessione client da parte del servizio.

   Il codice motivo non viene restituito al client.

   - Ricercare i messaggi di log nei file `mqxr.log` e `AMQERR01.LOG` per il gestore code a cui è connesso il client; fare riferimento a "Log lato server" a pagina 162.
3. La connessione TCP/IP è stata interrotta?

   Un firewall potrebbe avere un'impostazione di timeout basso per contrassegnare una connessione TCPIP come inattiva ed eliminare la connessione.

   - Ridurre il tempo di connessione TCPIP inattivo utilizzando `MqttConnectOptions.setKeepAliveInterval`.

Risoluzione del problema: messaggi persi in un'applicazione MQTT

Risolvere il problema di perdita di un messaggio. Il messaggio non è persistente, è stato inviato nel posto sbagliato o non è mai stato inviato? Un programma client codificato in modo errato potrebbe perdere i messaggi.

Prima di iniziare

Quanto sei certo che il tuo messaggio sia andato perduto? Si può dedurre che un messaggio è stato perso perché il messaggio non è stato ricevuto? Se il messaggio è una pubblicazione, quale messaggio viene perso: il messaggio inviato dal publisher o il messaggio inviato al sottoscrittore? Oppure la sottoscrizione è stata persa e il broker non sta inviando le pubblicazioni per tale sottoscrizione al sottoscrittore?

Se la soluzione implica la pubblicazione / sottoscrizione distribuita, utilizzando i cluster o le gerarchie di pubblicazione / sottoscrizione, esistono numerosi problemi di configurazione che potrebbero causare la perdita di un messaggio.

Se hai inviato un messaggio con la qualità del servizio "Almeno una volta" o "Al massimo una volta", è probabile che il messaggio che pensi sia stato perso non sia stato recapitato nel modo previsto. È improbabile che il messaggio sia stato eliminato erroneamente dal sistema. Potrebbe non essere stato possibile creare la pubblicazione o la sottoscrizione prevista.

Il passo più importante che si esegue nella determinazione dei problemi dei messaggi persi è quello di confermare che il messaggio è perso. Ricreare lo scenario e perdere ulteriori messaggi. Utilizzare la QoS (quality of service) "Almeno una volta" o "Al massimo una volta" per eliminare tutti i casi in cui il sistema elimina i messaggi.

Informazioni su questa attività

Ci sono quattro gambe per diagnosticare un messaggio perso.

1. Messaggi "Fuoco e dimenticati" che funzionano come - progettato. I messaggi "Fuoco e dimenticati" sono a volte scartati dal sistema.
2. Configurazione: l'impostazione della pubblicazione / sottoscrizione con le autorizzazioni corrette in un ambiente distribuito non è semplice.
3. Errori di programmazione del client: la responsabilità della consegna dei messaggi non è esclusivamente responsabilità del codice scritto da IBM.
4. Se hai esaurito tutte queste possibilità, potresti decidere di coinvolgere il servizio IBM .

Procedura

1. Se il messaggio perso aveva la qualità del servizio "Fuoco e dimentica", impostare la qualità del servizio "Almeno una volta" o "Al massimo una volta". Tentare nuovamente di perdere il messaggio.
   - I messaggi inviati con la qualità del servizio "Fuoco e dimentica" vengono gettati via da IBM WebSphere MQ in una serie di circostanze:
     – Perdita di comunicazioni e canale arrestato.
     – Gestore code arrestato.
     – Numero eccessivo di messaggi.
   - La consegna dei messaggi "Fire and forget" (Fuoco e dimentica) dipende dall'affidabilità del TCP/IP. TCP/IP continua a inviare nuovamente i pacchetti di dati fino a quando non viene confermata la consegna. Se la sessione TCP/IP è interrotta, i messaggi con QoS (quality of service) "Fire and forget" vengono persi. La sessione potrebbe essere interrotta dalla chiusura del client o del server, da un problema di comunicazione o da un firewall che scollega la sessione.
2. Verificare che il client stia riavviando la precedente sessione, in modo da inviare di nuovo i messaggi non recapitati con QoS (quality of service) "Almeno una volta" o "Al massimo una volta".
   a) Se l'applicazione client sta utilizzando il client Java SE MQTT, verificare che `MqttClient.CleanSession` sia impostato su `false`
   b) Se si utilizzano librerie client differenti, verificare che una sessione sia stata riavviata correttamente.
3. Verificare che l'applicazione client stia riavviando la stessa sessione e che non stia avviando una sessione differente per errore.

   Per avviare nuovamente la stessa sessione, `cleanSession = false`, `Mqttclient.clientIdentifier` e `MqttClient.serverURI` devono essere uguali alla sessione precedente.
4. Se una sessione viene chiusa prematuramente, verificare che il messaggio sia disponibile nell'archivio di persistenza sul client per inviarlo di nuovo.
   a) Se l'applicazione client utilizza il client Java SE MQTT, verificare che il messaggio venga salvato nella cartella di persistenza; consultare “File di log lato client” a pagina 163
   b) Se si utilizzano librerie client differenti o se è stato implementato il proprio meccanismo di persistenza, verificare che funzioni correttamente.
5. Verificare che nessuno abbia eliminato il messaggio prima della consegna.

   I messaggi non recapitati in attesa di consegna ai clienti MQTT vengono memorizzati in `SYSTEM.MQTT.TRANSMIT.QUEUE`. I messaggi in attesa di consegna al server di telemetria vengono memorizzati dal meccanismo di persistenza del client; consultare Persistenza del messaggio nei client MQTT.
6. Verificare che il cliente disponga di una sottoscrizione per la pubblicazione che prevede di ricevere.

   Elencare le sottoscrizioni utilizzando WebSphere MQ Explorer oppure utilizzando i comandi **runmqsc** o PCF. Tutte le sottoscrizioni dei client MQTT vengono denominate. Viene fornito un nome del formato: *`ClientIdentifier`*:*`Topic name`*
7. Verificare che il publisher disponga dell'autorizzazione alla pubblicazione e che il sottoscrittore sottoscriva l'argomento della pubblicazione.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   In un sistema di pubblicazione / sottoscrizione in cluster, il sottoscrittore deve essere autorizzato all'argomento sul gestore code a cui è connesso il sottoscrittore. Non è necessario che il sottoscrittore sia autorizzato a sottoscrivere l'argomento sul gestore code in cui è pubblicata la pubblicazione. I canali tra gestori code devono essere correttamente autorizzati a trasmettere la sottoscrizione proxy e inoltrare la pubblicazione.

Creare la stessa sottoscrizione e pubblicarla utilizzando Esplora risorse di IBM WebSphere MQ . Simula la pubblicazione e la sottoscrizione del client applicativo utilizzando il programma di utilità client. Avviare il programma di utilità da IBM WebSphere MQ Explorer e modificarne l'ID utente in modo che corrisponda a quello adottato dalla propria app client.

8. Verificare che il sottoscrittore disponga dell'autorizzazione per inserire la pubblicazione in SYSTEM.MQTT.TRANSMIT.QUEUE.

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

9. Verificare che l'applicazione point-to-point IBM WebSphere MQ disponga dell'autorizzazione per inserire il suo messaggio sul SYSTEM.MQTT.TRANSMIT.QUEUE.

```
dspmqaut -m qMgr -n queueName -t queue -p user ID
```

Consultare "Invio diretto di un messaggio ad un client" in Configurazione dell'accodamento distribuito per l'invio di messaggi ai client MQTT.

Risoluzione del problema: il servizio MQXR (Telemetry) non viene avviato

Risolvere il problema del mancato avvio del servizio di telemetria (MQXR). Controllare l'installazione di WebSphere MQ Telemetry e non ci sono file mancanti, spostati o con autorizzazioni errate. Controllare i percorsi utilizzati dal servizio di telemetria (MQXR) per individuare i programmi di servizio di telemetria (MQXR).

Prima di iniziare

La funzione WebSphere MQ Telemetry è installata. IBM WebSphere MQ Explorer ha una cartella Telemetria in **IBM WebSphere MQ > Gestori code> *qMgrNome* > Telemetria**. Se la cartella non esiste, l'installazione non è riuscita.

Il servizio di telemetria (MQXR) deve essere stato creato per essere avviato. Se il servizio di telemetria (MQXR) non è stato creato, eseguire **Definisci configurazione di esempio ...** nella cartella `Telemetry` .

Se il servizio di telemetria (MQXR) è stato avviato in precedenza, nella cartella `Telemetry` vengono create ulteriori cartelle **Canali** e **Stato canale** . Il servizio di telemetria, SYSTEM.MQXR.SERVICE, si trova nella cartella **Servizi** . È visibile se si fa clic sul pulsante di opzione Esplora per visualizzare gli oggetti di sistema.

Fare clic con il pulsante destro del mouse su SYSTEM.MQXR.SERVICE per avviare e arrestare il servizio, visualizzarne lo stato e visualizzare se l'ID utente dispone dell'autorizzazione per avviare il servizio.

Informazioni su questa attività

Impossibile avviare il servizio di telemetria SYSTEM.MQXR.SERVICE (MQXR). Un errore di avvio si manifesta in due modi diversi:

1. Il comando di avvio ha esito negativo immediatamente.
2. Il comando di avvio ha esito positivo ed è immediatamente seguito dall'arresto del servizio.

Procedura

1. Avvia servizio

Risultato

Il servizio si arresta immediatamente. Una finestra visualizza un messaggio di errore; ad esempio:

```
WebSphere MQ cannot process the request because the
executable specified cannot be started. (AMQ4160)
```

Motivo

I file non sono presenti nell'installazione o le autorizzazioni sui file installati sono impostate in modo non corretto.

la funzione IBM WebSphere MQ Telemetry è installata solo su una coppia di gestori code ad alta disponibilità. Se l'istanza del gestore code passa a uno standby, tenta di avviare SYSTEM.MQXR.SERVICE. Il comando per l'avvio del servizio ha esito negativo perché il servizio di telemetria (MQXR) non è installato sullo standby.

Indagine

Consultare i log degli errori; consultare "Log lato server" a pagina 162.

Azioni

Installare o disinstallare e reinstallare la funzione WebSphere MQ Telemetry.

2. Avviare il servizio; attendere 30 secondi; aggiornare Explorer e controllare lo stato del servizio.

Risultato

Il servizio viene avviato e quindi arrestato.

Motivo

SYSTEM.MQXR.SERVICE ha avviato il comando **runMQXRService** , ma il comando non è riuscito.

Indagine

Consultare i log degli errori; consultare "Log lato server" a pagina 162.

Verificare se il problema si verifica solo con il canale di esempio definito. Eseguire il backup e cancellare il contenuto della directory *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ . Eseguire la procedura guidata di configurazione di esempio e provare ad avviare il servizio.

Azioni

Ricercare i problemi di autorizzazione e percorso.

Risoluzione del problema: modulo di login di JAAS non richiamato dal servizio di telemetria

Verificare se il modulo di login JAAS non viene richiamato dal servizio di telemetria (MQXR) e configurare JAAS per correggere il problema.

Prima di iniziare

È stato modificato *WMQ installation directory*\mqxr\samples\LoginModule.java per creare la propria classe di autenticazione *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. In alternativa, sono state scritte le proprie classi di autenticazione JAAS e sono state inserite in una directory di propria scelta. Dopo alcuni test iniziali con il servizio di telemetria MQXR, si sospetta che la propria classe di autenticazione non venga richiamata dal servizio di telemetria (MQXR).

Nota: Evitare che le classi di autenticazione vengano sovrascritte dalla manutenzione applicata a WebSphere MQ. Utilizzare il proprio percorso per le classi di autenticazione, piuttosto che un percorso all'interno della struttura di directory WebSphere MQ .

Informazioni su questa attività

L'attività utilizza uno scenario per illustrare come risolvere il problema. Nello scenario, un pacchetto denominato security.jaas contiene una classe di autenticazione JAAS denominata JAASLogin.class. Viene memorizzato nel percorso C:\WMQTelemetryApps\security\jaas. Fare riferimento alla configurazione del canale di telemetria JAAS per assistenza nella configurazione di JAAS per IBM WebSphere MQ Telemetry. L'esempio, "Configurazione JAAS di esempio" a pagina 185 è una configurazione di esempio.

Procedura

1. Ricercare in `mqxr.log` un'eccezione generata da `javax.security.auth.login.LoginException`.

   Consultare “Log lato server” a pagina 162 per il percorso di `mqxr.log`e Figura 54 a pagina 187 per un esempio dell'eccezione elencata nel log.
2. Correggere la configurazione JAAS confrontandola con l'esempio utilizzato in “Configurazione JAAS di esempio” a pagina 185.
3. Sostituire la classe di login con l'esempio `JAASLoginModule`, dopo il refactoring nel package di autenticazione e distribuirlo utilizzando lo stesso percorso. Cambiare il valore di `loggedIn` tra `true` e `false`.

   Se il problema si risolve quando `loggedIn` è `true`e viene visualizzato lo stesso quando `loggedIn` è `false`, il problema si trova nella classe di login.
4. Verificare se il problema è relativo all'autorizzazione piuttosto che all'autenticazione.
   a) Modificare la definizione del canale di telemetria per eseguire il controllo dell'autorizzazione utilizzando un ID utente fisso. Selezionare un ID utente membro del gruppo `mqm` .
   b) Rieseguire l'applicazione client.

      Se il problema scompare, la soluzione risiede nell'ID utente che viene passato per l'autorizzazione. Qual è il nome utente che viene passato? Stamparlo nel file dal modulo di login. Controllare le relative autorizzazioni di accesso utilizzando IBM WebSphere MQ Explorer o **dspmqauth**.

Configurazione JAAS di esempio

Utilizzare la procedura guidata **Nuovo canale di telemetria** , in WebSphere MQ Explorer, per configurare un canale di telemetria. Il client si connette alla porta 1884 e al canale di telemetria `JAASMCAUser` . Figura 48 a pagina 185 mostra un esempio del file delle proprietà di telemetria creato dalla procedura guidata di telemetria. Non modificare direttamente questo file. Il canale esegue l'autenticazione utilizzando JAAS, utilizzando la configurazione denominata `JAASConfig`. Una volta autenticato, il client utilizza l'ID utente `Admin` per autorizzare l'accesso agli oggetti 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

Il file di configurazione JAAS ha una sezione denominata `JAASConfig` che denomina la classe Java `security.jaas.JAASLogin`, che JAAS deve utilizzare per autenticare i client.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Figura 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Quando `SYSTEM.MQTT.SERVICE` viene avviato, aggiunge il percorso in Figura 50 a pagina 186 al percorso classi.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figura 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

Figura 51 a pagina 186 mostra il percorso aggiuntivo in Figura 50 a pagina 186 aggiunto al percorso classi configurato per il servizio di telemetria (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. Output del percorso classi da runMQXRService.bat

L'output in Figura 52 a pagina 186 mostra che il servizio di telemetria (MQXR) è stato avviato con la definizione del canale mostrata nella Figura 48 a pagina 185.

```
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

Quando l'applicazione client si connette al canale JAAS , se `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` non corrisponde al nome di una stanza JAAS nel file `jaas.config` , la connessione non riesce e il client genera un'eccezione con un codice di ritorno `0`; consultare Figura 53 a pagina 187. La seconda eccezione, `Client is not connected (32104)`, è stata generata perché il client ha provato a disconnettersi quando non era connesso.

```
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. Eccezione generata durante la connessione com.ibm.mq.id.PubAsyncRestartable

`mqxr.log` contiene l'output aggiuntivo mostrato in Figura 53 a pagina 187.

L'errore viene rilevato da JAAS che genera `javax.security.auth.login.LoginException` con la causa `No LoginModules configured for JAAS`. Potrebbe essere causato, come in Figura 54 a pagina 187, da un nome di configurazione errato. Potrebbe anche essere il risultato di altri problemi che JAAS ha rilevato durante il caricamento della configurazione di JAAS .

Se JAASnon riporta alcuna eccezione, JAAS ha caricato correttamente la classe `security.jaas.JAASLogin` denominata nella 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` - errore durante il caricamento della configurazione JAAS

Risoluzione del problema: avvio o esecuzione del daemon

Consultare il daemon IBM WebSphere MQ Telemetry per il log della console dei dispositivi, attivare la traccia o utilizzare la tabella dei sintomi in questo argomento per risolvere i problemi relativi al daemon.

Procedura

1. Controllare il log della console.

   Se il daemon è in esecuzione in primo piano, i messaggi della console vengono scritti nella finestra del terminale. Se il daemon è stato avviato in background, la console è quella a cui è stato reindirizzato `stdout` .
2. Riavviare il daemon.

   Le modifiche al file di configurazione non vengono attivate fino al riavvio del daemon.
3. Consultare Tabella 7 a pagina 188:

Tabella 7. Tabella dei sintomi

Problema	Soluzione suggerita

Il seguente messaggio viene visualizzato quando si avvia il daemon su Windows:
`Il sistema non può eseguire il programma specificato`
o
`L'applicazione non è stata avviata poiché la relativa configurazione non è corretta.`
	Installare Microsoft Visual C++ 2008 Redistributable Package.

Due o più daemon o server con capacità MQTT sono interconnessi da uno o più bridge e il processore mostra un carico eccessivo.	È possibile che vi sia un loop di messaggi, con uno o più messaggi che vengono trasmessi ripetutamente da un server a un altro. Esaminare i parametri dell'argomento nei file di configurazione. Utilizzare argomenti più specifici, laddove possibile. I caratteri jolly generici in entrambe le direzioni sono la causa più comune dei loop di connessione.
Il bridge non è in grado di collegarsi ad un server MQTT remoto a cui possono connettersi altri client MQTT.	Il server remoto potrebbe essere incompatibile con i tentativi di determinare se il server remoto è anche il daemon WebSphere MQ Telemetry per i dispositivi. Provare a impostare **try_private** su `off` per disattivare l'elaborazione speciale per eliminare i loop di messaggi.
Questo messaggio viene stampato quando un bridge è configurato: `Avvertenza: la connessione non era il primo pacchetto sul socket 1888, ricevuto CONNACK.`	È probabile che sia stato configurato un bridge per il loop al daemon locale. Loopback non supportato.

Risoluzione del problema: i client MQTT non si collegano al daemon

I client non si collegano al daemon oppure il daemon non si collega ad altri daemon o a un canale di telemetria WebSphere MQ .

Informazioni su questa attività

Traccia ogni pacchetto MQTT inviato e ricevuto dal daemon.

Procedura

Impostare il parametro **trace_output** su `protocol` nel file di configurazione del daemon oppure inviare un comando al daemon utilizzando il file `amqtdd.upd` .

Consultare Trasferire i messaggi tra il daemon IBM WebSphere MQ Telemetry per i dispositivi e IBM WebSphere MQ , per un esempio di utilizzo del file `amqtdd.upd` .

Utilizzando l'impostazione del protocollo, il daemon stampa un messaggio alla console che descrive ogni pacchetto MQTT che invia e riceve.

Informazioni particolari

Queste informazioni sono state sviluppate per i prodotti ed i servizi offerti negli Stati Uniti.

IBM potrebbe non offrire i prodotti, i servizi o le funzioni descritti in questo documento in altri paesi. Consultare il rappresentante IBM locale per informazioni sui prodotti e sui servizi disponibili nel proprio paese. Ogni riferimento relativo a prodotti, programmi o servizi IBM non implica che solo quei prodotti, programmi o servizi IBM possano essere utilizzati. In sostituzione a quelli forniti da IBM possono essere usati prodotti, programmi o servizi funzionalmente equivalenti che non comportino la violazione dei diritti di proprietà intellettuale o di altri diritti dell'IBM. È comunque responsabilità dell'utente valutare e verificare la possibilità di utilizzare altri programmi e/o prodotti, fatta eccezione per quelli espressamente indicati dall'IBM.

IBM potrebbe disporre di applicazioni di brevetti o brevetti in corso relativi all'argomento descritto in questo documento. La fornitura di tale documento non concede alcuna licenza a tali brevetti. Chi desiderasse ricevere informazioni relative a licenze può rivolgersi per iscritto a:

Director of Commercial Relations
IBM Corporation
Schoenaicher Str. 220
D-7030 Boeblingen
U.S.A.

Per richieste di licenze relative ad informazioni double-byte (DBCS), contattare il Dipartimento di Proprietà Intellettuale IBM nel proprio paese o inviare richieste per iscritto a:

Intellectual Property Licensing
Legge sulla proprietà intellettuale e legale
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Il seguente paragrafo non si applica al Regno Unito o a qualunque altro paese in cui tali dichiarazioni sono incompatibili con le norme locali: INTERNATIONAL BUSINESS MACHINES CORPORATION FORNISCE LA PRESENTE PUBBLICAZIONE "NELLO STATO IN CUI SI TROVA" SENZA GARANZIE DI ALCUN TIPO, ESPRESSE O IMPLICITE, IVI INCLUSE, A TITOLO DI ESEMPIO, GARANZIE IMPLICITE DI NON VIOLAZIONE, DI COMMERCIABILITÀ E DI IDONEITÀ PER UNO SCOPO PARTICOLARE. Alcuni stati non consentono la rinuncia a garanzie esplicite o implicite in determinate transazioni; quindi la presente dichiarazione potrebbe non essere applicabile.

Questa pubblicazione potrebbe contenere imprecisioni tecniche o errori tipografici. Le informazioni incluse in questo documento vengono modificate su base periodica; tali modifiche vengono incorporate nelle nuove edizioni della pubblicazione. IBM si riserva il diritto di apportare miglioramenti o modifiche al prodotto/i e/o al programma/i descritti nella pubblicazione in qualsiasi momento e senza preavviso.

Qualsiasi riferimento a siti Web non IBM contenuto nelle presenti informazioni è fornito per consultazione e non vuole in alcun modo promuovere i suddetti siti Web. I materiali presenti in tali siti Web non sono parte dei materiali per questo prodotto IBM e l'utilizzo di tali siti Web è a proprio rischio.

Tutti i commenti e i suggerimenti inviati potranno essere utilizzati liberamente da IBM e diventeranno esclusiva della stessa.

Coloro che detengono la licenza su questo programma e desiderano avere informazioni su di esso allo scopo di consentire (i) uno scambio di informazioni tra programmi indipendenti ed altri (compreso questo) e (ii) l'uso reciproco di tali informazioni, dovrebbero rivolgersi a:

IBM Corporation
Coordinatore interoperabilità software, Dipartimento 49XA
Autostrada 3605 52 N

Rochester, MN 55901
U.S.A.

Queste informazioni possono essere rese disponibili secondo condizioni contrattuali appropriate, compreso, in alcuni casi, il pagamento di un addebito.

Il programma su licenza descritto in queste informazioni e tutto il materiale su licenza disponibile per esso sono forniti da IBM in base ai termini dell' IBM Customer Agreement, IBM International Program License Agreement o qualsiasi altro accordo equivalente tra le parti.

Tutti i dati relativi alle prestazioni contenuti in questo documento sono stati determinati in un ambiente controllato. Pertanto, i risultati ottenuti in altri ambienti operativi possono variare in modo significativo. Alcune misurazioni potrebbero essere state fatte su sistemi a livello di sviluppo e non vi è alcuna garanzia che queste misurazioni saranno le stesse sui sistemi generalmente disponibili. Inoltre, alcune misurazioni potrebbero essere state stimate mediante estrapolazione. I risultati quindi possono variare. Gli utenti di questo documento dovrebbero verificare i dati applicabili per il loro ambiente specifico.

Le informazioni relative a prodotti non IBM provengono dai fornitori di tali prodotti, dagli annunci pubblicati o da altre fonti pubblicamente disponibili. IBM non ha verificato tali prodotti e, pertanto, non può garantirne l'accuratezza delle prestazioni. Eventuali commenti relativi alle prestazioni dei prodotti non IBM devono essere indirizzati ai fornitori di tali prodotti.

Tutte le dichiarazioni riguardanti la direzione o l'intento futuro di IBM sono soggette a modifica o ritiro senza preavviso e rappresentano solo scopi e obiettivi.

Questa pubblicazione contiene esempi di dati e prospetti utilizzati quotidianamente nelle operazioni aziendali, Per illustrarle nel modo più completo possibile, gli esempi includono i nomi di individui, società, marchi e prodotti. Tutti questi nomi sono fittizi e qualsiasi somiglianza con nomi ed indirizzi adoperati da imprese realmente esistenti sono una mera coincidenza.

LICENZA SUL COPYRIGHT:

Queste informazioni contengono programmi applicativi di esempio in lingua originale, che illustrano le tecniche di programmazione su diverse piattaforme operative. È possibile copiare, modificare e distribuire questi programmi di esempio sotto qualsiasi forma senza alcun pagamento alla IBM, allo scopo di sviluppare, utilizzare, commercializzare o distribuire i programmi applicativi in conformità alle API (application programming interface) a seconda della piattaforma operativa per cui i programmi di esempio sono stati scritti. Questi esempi non sono stati testati approfonditamente tenendo conto di tutte le condizioni possibili. IBM, quindi, non può garantire o sottintendere l'affidabilità, l'utilità o il funzionamento di questi programmi.

Se si sta visualizzando queste informazioni in formato elettronico, le fotografie e le illustrazioni a colori potrebbero non apparire.

Informazioni sull'interfaccia di programmazione

Le informazioni sull'interfaccia di programmazione, se fornite, consentono di creare software applicativo da utilizzare con questo programma.

Questo manuale contiene informazioni sulle interfacce di programmazione che consentono al cliente di scrivere programmi per ottenere i servizi di IBM WebSphere MQ.

Queste informazioni, tuttavia, possono contenere diagnosi, modifica e regolazione delle informazioni. La diagnosi, la modifica e la regolazione delle informazioni vengono fornite per consentire il debug del software applicativo.

Importante: Non utilizzare queste informazioni di diagnosi, modifica e ottimizzazione come interfaccia di programmazione poiché sono soggette a modifica.

Marchi

IBM, il logo IBM , ibm.com, sono marchi di IBM Corporation, registrati in molte giurisdizioni nel mondo. Un elenco aggiornato dei marchi IBM è disponibile sul web in "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Altri nomi di prodotti e servizi potrebbero essere marchi di IBM o altre società.

Microsoft e Windows sono marchi di Microsoft Corporation negli Stati Uniti e/o in altri paesi.

UNIX è un marchio registrato di The Open Group negli Stati Uniti e/o in altri paesi.

Linux è un marchi registrato di Linus Torvalds negli Stati Uniti e/o in altri paesi.

Questo prodotto include il software sviluppato da Eclipse Project (http://www.eclipse.org/).

Java e tutti i marchi e i logo Java sono marchi registrati di Oracle e/o di società affiliate.

Numero parte:

(1P) P/N: