7.5

Mobile Messaging e M2M

Nota

Antes de usar estas informações e o produto que elas suportam, leia as informações em “Avisos” na página 193.

Esta edição se aplica à versão 7 liberação 5 do IBM® WebSphere MQ e a todas as liberações e modificações subsequentes até que seja indicado de outra forma em novas edições.

Índice

Introdução ao MQTT

Saiba como enviar mensagens entre aplicativos móveis usando o transporte de telemetria do MQ (MQTT). O protocolo é destinado para uso em redes sem fio e de baixa largura de banda. Um aplicativo móvel que usa MQTT envia e recebe mensagens chamando uma biblioteca MQTT . As mensagens são trocadas por meio de um servidor do sistema de mensagens do MQTT. O cliente e servidor MQTT manipulam as complexidades de entrega de mensagens confiáveis para o aplicativo móvel e mantêm os custos de gerenciamento de rede pequenos.

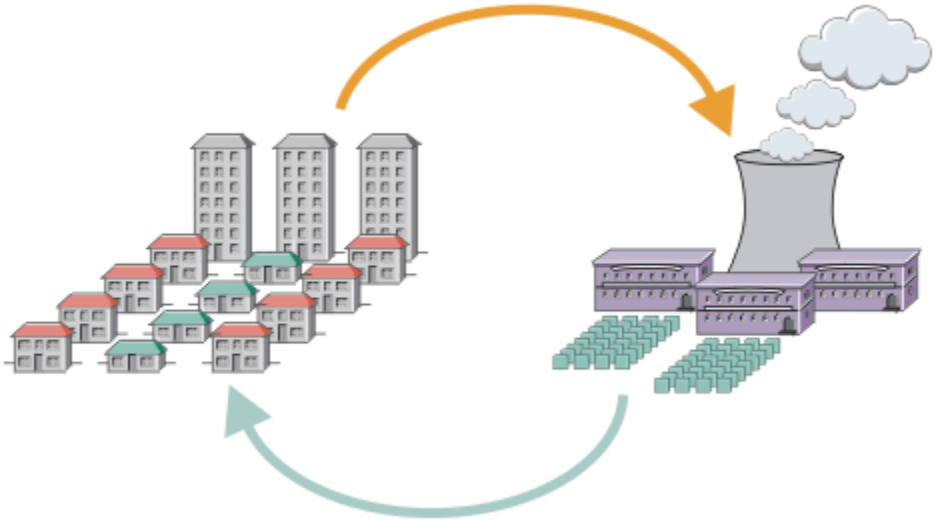
Os aplicativos MQTT são executados em dispositivos móveis, como smartphones e tablets. O MQTT também é usado para telemetria para receber dados de sensores e para controlá-los remotamente. Para dispositivos móveis e sensores, o MQTT oferece um protocolo de publicação/assinatura altamente escalável com entrega garantida. Para enviar e receber mensagens do MQTT, inclua uma biblioteca do cliente MQTT em seu aplicativo.

A biblioteca do cliente MQTT é pequena. A biblioteca age como uma caixa de correio, enviando e recebendo mensagens com outros aplicativos MQTT conectados a um servidor MQTT. Ao enviar mensagens, em vez de permanecer conectado a um servidor que está aguardando por uma resposta, os aplicativos MQTT conservam a vida útil da bateria. A biblioteca envia mensagens para outros dispositivos através de um servidor MQTT que está executando o protocolo MQTT version 3.1 . É possível enviar mensagens a um cliente específico ou usar o sistema de mensagens da publicação/assinatura para conectar muitos dispositivos.

As bibliotecas do cliente MQTT conectam aplicativos para dispositivos móveis e sensores em um servidor MQTT usando o protocolo do MQTT.

IBM MessageSight e IBM WebSphere MQ são MQTT servidores. Eles podem conectar grandes volumes de aplicativos cliente MQTT e podem conectar as redes MQTT e IBM WebSphere MQ juntas. Consulte “Introdução aos servidores MQTT” na página 138. IBM WebSphere MQ e IBM MessageSight podem formar uma ponte entre aplicativos da web externos em execução em dispositivos móveis e sensores e outros tipos de publicação/assinatura e aplicativos do sistema de mensagens em execução na empresa. A ponte facilita as "soluções inteligentes" de construção que incorporam dispositivos móveis e sensores.

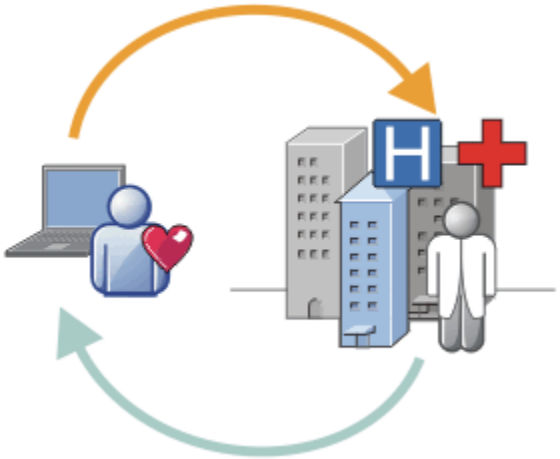
As soluções inteligentes desvendam a riqueza de informações disponíveis na Internet para os aplicativos em execução em dispositivos móveis e sensores. Dois exemplos de aplicativos inteligentes baseados em telemetria são eletricidade inteligente e serviços de saúde inteligentes.



- Uma mensagem do MQTT que contém dados de uso de energia enviados para o provedor de serviços.
- Um aplicativo de telemetria envia comandos de controle que são baseados na análise de dados de uso de energia.
- Para obter mais informações, consulte Cenário do Telemetry: monitoramento e controle de energia da Página inicial.

Figura 1. Medição de Eletricidade Inteligente

Figura 2. Monitoramento de Saúde Inteligente



- Um aplicativo de telemetria envia seus dados de saúde para o hospital e seu médico.
- O feedback ou os alertas de mensagem do MQTT podem ser enviados com base na análise dos dados de funcionamento.
- Para obter mais informações, consulte Cenário do Telemetry: monitoramento de paciente da Página inicial.

É possível construir o MQTT em dispositivos pequenos gravando seu próprio aplicativo para o MQTT protocol. Para ajudar você a fazer isso, o IBM fornece bibliotecas do cliente que suportam apps que são executados no MQTT. Consulte “Introdução aos clientes MQTT” na página 11. O IBM fornece bibliotecas do cliente para iOS apps e para Android apps V 7.5.0.1 e um JavaScript cliente do navegador para apps da web independentes da plataforma. V 7.5.0.1 As páginas do cliente do JavaScript se conectam ao IBM MessageSight e ao IBM WebSphere MQ com o protocolo MQTT sobre WebSockets. O IBM também fornece apps de amostra MQTT para C e Java on Linux® e Windows

As bibliotecas C e Java são executadas em iOS, Android, Windowse em várias plataformas UNIX and Linux É possível transmitir o código-fonte C para a biblioteca do cliente MQTT para outras plataformas. As bibliotecas do cliente MQTT para C e Java estão disponíveis com uma licença de software livre do projeto do Eclipse Paho . Consulte o Eclipse Paho. A especificação MQTT protocol está aberta e disponível em MQTT.org.

MQTT protocol

O MQTT protocol é leve no sentido que os clientes são pequenos e ele usa a largura da banda da rede com eficiência. O protocolo MQTT suporta entrega garantida e transferências fire-and-forget. No protocolo, a entrega da mensagem é desacoplada do aplicativo. A extensão do desacoplamento em um aplicativo depende da maneira como um cliente MQTT e o servidor MQTT são gravados. A entrega desacoplada libera a um aplicativo a partir de qualquer conexão do servidor e de espera por mensagens. O modelo de interação é como e-mail, mas otimizado para a programação de aplicativos.

O protocolo MQTT V3.1 é publicado; consulte MQTT V3.1 Especificação de protocolo. A especificação identifica um número de características distintivas sobre o protocolo:

- Ele é um protocolo de publicação/assinatura.

  Além de fornecer distribuição de mensagens de um para muitos, a publicação/assinatura desacopla aplicativos. Os recursos são úteis em aplicativos que possuem muitos clientes.
- Não é dependente de nenhuma maneira no conteúdo da mensagem.
- Ele é executado através de TCP/IP, que fornece conectividade de rede básica.
- Ele tem três qualidades de serviço para entrega de mensagens:

  "No máximo uma vez"
  As mensagens são entregues de acordo com os melhores esforços da rede Internet Protocol subjacente. A perda de mensagens pode ocorrer.

  Use essa qualidade de serviço com os dados do sensor de ambiente da comunicação, por exemplo. Não importa se uma leitura individual for perdida, se a próxima for publicada logo depois.

  "Pelo menos uma vez"
  As mensagens são asseguradas para chegar, mas podem ocorrer duplicatas.

  "Exatamente uma vez"
  As mensagens são asseguradas para chegar exatamente uma vez.

  Use essa qualidade de serviço com sistemas de faturamento, por exemplo. As mensagens perdidas ou duplicadas podem levar a causa ou imposição de encargos incorretos.
- É econômico na maneira como ele gerencia o fluxo de mensagens na rede. Por exemplo, o cabeçalho de comprimento fixo é apenas 2 bytes de comprimento e as trocas de protocolo são minimizadas para reduzir o tráfego na rede.
- Ele possui um recurso "Last Will and Testament" que notifica os assinantes do desconexão anormal de um cliente do servidorMQTT. Consulte “Publicação last will and testament” na página 131.

MQTT version 3.1 é suportado pelo IBM IBM WebSphere MQ e pelo IBM MessageSight MQTT está implementada através do TCP/IP. Outra versão do protocolo, MQTT-S, está disponível para as redes não TCP/IP. Consulte o Especificação do MQTT-S version 1.2.

Comunidades do MQTT

O IBM está executando o IBM Developer Sistema de Mensagens comunidade para os desenvolvedores do MQTT que estão gravando aplicativos para IBM MessageSight e IBM WebSphere MQ.

O MQTT.org é um bom lugar para acessar para conhecer e discutir as implementações e extensões para o protocolo do MQTT.

O MQTT é um projeto de software livre do Eclipse, sob o Eclipse Technology Project. A comunidade do Paho está desenvolvendo servidores e clientes de software livre. Consulte o Eclipse Paho.

Introdução ao MQTT

Saiba como enviar mensagens entre aplicativos móveis usando o transporte de telemetria do MQ (MQTT). O protocolo é destinado para uso em redes sem fio e de baixa largura de banda. Um aplicativo móvel que usa MQTT envia e recebe mensagens chamando uma biblioteca MQTT . As mensagens são trocadas por meio de um servidor do sistema de mensagens do MQTT. O cliente e servidor MQTT manipulam as complexidades de entrega de mensagens confiáveis para o aplicativo móvel e mantêm os custos de gerenciamento de rede pequenos.

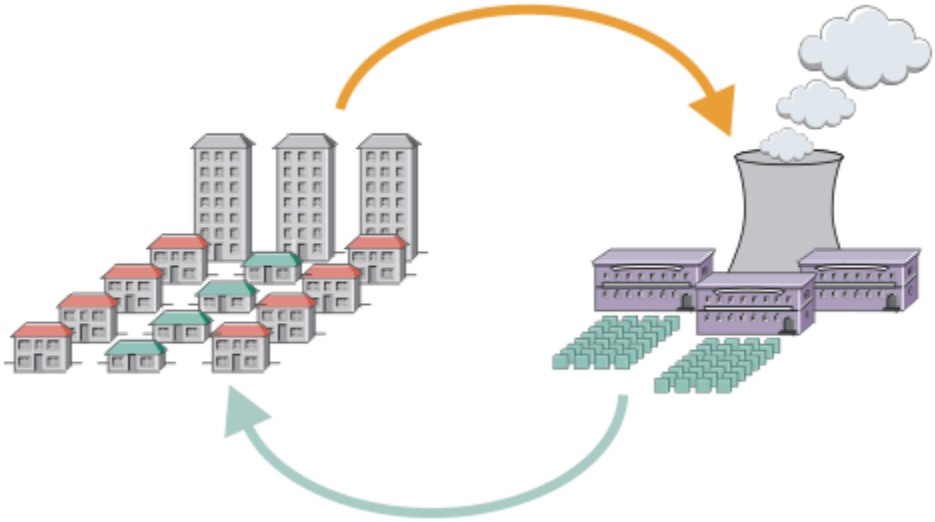
Os aplicativos MQTT são executados em dispositivos móveis, como smartphones e tablets. O MQTT também é usado para telemetria para receber dados de sensores e para controlá-los remotamente. Para dispositivos móveis e sensores, o MQTT oferece um protocolo de publicação/assinatura altamente escalável com entrega garantida. Para enviar e receber mensagens do MQTT, inclua uma biblioteca do cliente MQTT em seu aplicativo.

A biblioteca do cliente MQTT é pequena. A biblioteca age como uma caixa de correio, enviando e recebendo mensagens com outros aplicativos MQTT conectados a um servidor MQTT. Ao enviar mensagens, em vez de permanecer conectado a um servidor que está aguardando por uma resposta, os aplicativos MQTT conservam a vida útil da bateria. A biblioteca envia mensagens para outros dispositivos através de um servidor MQTT que está executando o protocolo MQTT version 3.1 . É possível enviar mensagens a um cliente específico ou usar o sistema de mensagens da publicação/assinatura para conectar muitos dispositivos.

As bibliotecas do cliente MQTT conectam aplicativos para dispositivos móveis e sensores em um servidor MQTT usando o protocolo do MQTT.

IBM MessageSight e IBM WebSphere MQ são MQTT servidores. Eles podem conectar grandes volumes de aplicativos cliente MQTT e podem conectar as redes MQTT e IBM WebSphere MQ juntas. Consulte “Introdução aos servidores MQTT” na página 138. IBM WebSphere MQ e IBM MessageSight podem formar uma ponte entre aplicativos da web externos em execução em dispositivos móveis e sensores e outros tipos de publicação/assinatura e aplicativos do sistema de mensagens em execução na empresa. A ponte facilita as "soluções inteligentes" de construção que incorporam dispositivos móveis e sensores.

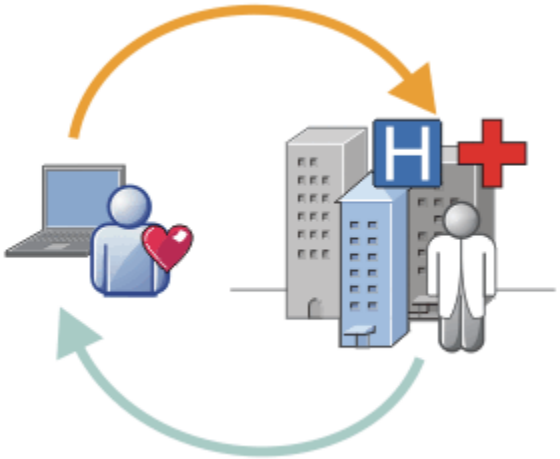
As soluções inteligentes desvendam a riqueza de informações disponíveis na Internet para os aplicativos em execução em dispositivos móveis e sensores. Dois exemplos de aplicativos inteligentes baseados em telemetria são eletricidade inteligente e serviços de saúde inteligentes.



- Uma mensagem do MQTT que contém dados de uso de energia enviados para o provedor de serviços.
- Um aplicativo de telemetria envia comandos de controle que são baseados na análise de dados de uso de energia.
- Para obter mais informações, consulte Cenário do Telemetry: monitoramento e controle de energia da Página inicial.

Figura 3. Medição de Eletricidade Inteligente

Figura 4. Monitoramento de Saúde Inteligente



- Um aplicativo de telemetria envia seus dados de saúde para o hospital e seu médico.
- O feedback ou os alertas de mensagem do MQTT podem ser enviados com base na análise dos dados de funcionamento.
- Para obter mais informações, consulte Cenário do Telemetry: monitoramento de paciente da Página inicial.

É possível construir o MQTT em dispositivos pequenos gravando seu próprio aplicativo para o MQTT protocol. Para ajudar você a fazer isso, o IBM fornece bibliotecas do cliente que suportam apps que são executados no MQTT. Consulte “Introdução aos clientes MQTT” na página 11. O IBM fornece bibliotecas do cliente para iOS apps e para Android apps V 7.5.0.1 e um JavaScript cliente do navegador para apps da web independentes da plataforma. V 7.5.0.1 As páginas do cliente do JavaScript se conectam ao IBM MessageSight e ao IBM WebSphere MQ com o protocolo MQTT sobre WebSockets. O IBM também fornece apps de amostra MQTT para C e Java on Linux e Windows

As bibliotecas C e Java são executadas em iOS, Android, Windowse em várias plataformas UNIX and Linux É possível transmitir o código-fonte C para a biblioteca do cliente MQTT para outras plataformas. As bibliotecas do cliente MQTT para C e Java estão disponíveis com uma licença de software livre do projeto do Eclipse Paho . Consulte o Eclipse Paho. A especificação MQTT protocol está aberta e disponível em MQTT.org.

MQTT protocol

O MQTT protocol é leve no sentido que os clientes são pequenos e ele usa a largura da banda da rede com eficiência. O protocolo MQTT suporta entrega garantida e transferências fire-and-forget. No protocolo, a entrega da mensagem é desacoplada do aplicativo. A extensão do desacoplamento em um aplicativo depende da maneira como um cliente MQTT e o servidor MQTT são gravados. A entrega desacoplada libera a um aplicativo a partir de qualquer conexão do servidor e de espera por mensagens. O modelo de interação é como e-mail, mas otimizado para a programação de aplicativos.

O protocolo MQTT V3.1 é publicado; consulte MQTT V3.1 Especificação de protocolo. A especificação identifica um número de características distintivas sobre o protocolo:

- Ele é um protocolo de publicação/assinatura.

  Além de fornecer distribuição de mensagens de um para muitos, a publicação/assinatura desacopla aplicativos. Os recursos são úteis em aplicativos que possuem muitos clientes.
- Não é dependente de nenhuma maneira no conteúdo da mensagem.
- Ele é executado através de TCP/IP, que fornece conectividade de rede básica.
- Ele tem três qualidades de serviço para entrega de mensagens:

  "No máximo uma vez"
  As mensagens são entregues de acordo com os melhores esforços da rede Internet Protocol subjacente. A perda de mensagens pode ocorrer.

  Use essa qualidade de serviço com os dados do sensor de ambiente da comunicação, por exemplo. Não importa se uma leitura individual for perdida, se a próxima for publicada logo depois.

  "Pelo menos uma vez"
  As mensagens são asseguradas para chegar, mas podem ocorrer duplicatas.

  "Exatamente uma vez"
  As mensagens são asseguradas para chegar exatamente uma vez.

  Use essa qualidade de serviço com sistemas de faturamento, por exemplo. As mensagens perdidas ou duplicadas podem levar a causa ou imposição de encargos incorretos.
- É econômico na maneira como ele gerencia o fluxo de mensagens na rede. Por exemplo, o cabeçalho de comprimento fixo é apenas 2 bytes de comprimento e as trocas de protocolo são minimizadas para reduzir o tráfego na rede.
- Ele possui um recurso "Last Will and Testament" que notifica os assinantes do desconexão anormal de um cliente do servidorMQTT. Consulte “Publicação last will and testament” na página 131.

MQTT version 3.1 é suportado pelo IBM IBM WebSphere MQ e pelo IBM MessageSight MQTT está implementada através do TCP/IP. Outra versão do protocolo, MQTT-S, está disponível para as redes não TCP/IP. Consulte o Especificação do MQTT-S version 1.2.

Comunidades do MQTT

O IBM está executando o IBM Developer Sistema de Mensagens comunidade para os desenvolvedores do MQTT que estão gravando aplicativos para IBM MessageSight e IBM WebSphere MQ.

O MQTT.org é um bom lugar para acessar para conhecer e discutir as implementações e extensões para o protocolo do MQTT.

O MQTT é um projeto de software livre do Eclipse, sob o Eclipse Technology Project. A comunidade do Paho está desenvolvendo servidores e clientes de software livre. Consulte o Eclipse Paho.

Introdução aos clientes MQTT

É possível iniciar o desenvolvimento de um dispositivo móvel ou aplicativo do machine-to-machine (M2M) construindo e executando um aplicativo cliente MQTT de amostra que usa uma biblioteca do cliente MQTT. Os apps de amostra e as bibliotecas do cliente associadas estão disponíveis no Mobile Messaging and M2M Pacote do Cliente da IBM. Há versões dos aplicativos e bibliotecas do cliente gravadas em Java, em JavaScripte em C. É possível executar esses apps na maioria das plataformas e dispositivos, incluindo dispositivos e produtos do Android do Apple.

Antes de começar

Para construir e executar seu aplicativo, você precisa ter alguma experiência em construir aplicativos para o dispositivo de destino ou plataforma e a linguagem de programação que está sendo usada. Um pouco de experiência geralmente é suficiente para que um aplicativo funcione adequadamente no seu dispositivo ou plataforma escolhida.

Se você usar um servidor de intensidade corporativa MQTT como IBM WebSphere MQ ou IBM MessageSight, será possível trocar informações a partir de seu aplicativo de amostra com seus aplicativos corporativos existentes.

Sobre esta tarefa

Os objetivos são os seguintes:

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.
2. Faça download do Mobile Messaging and M2M Pacote do Cliente.
3. Construir, para sua plataforma ou seu dispositivo de destino, os aplicativos de amostra do pacote do cliente.
4. Verifique se as amostras se comportar conforme esperado, conectando-as ao servidor MQTT.

Como resultado da construção e do teste dos aplicativos de amostra para o dispositivo ou a plataforma, crie um ambiente de desenvolvimento de trabalho que seja possível usar para construir seus próprios aplicativos clientes.

O Mobile Messaging and M2M Pacote do Cliente contém o MQTT SDK. Este SDK fornece os seguintes recursos:

- Aplicativos clientes MQTT de amostra gravados em Java, em JavaScript e em C.
- As bibliotecas do cliente MQTT que suportam estes aplicativos clientes e permitem que eles sejam executados na maioria das plataformas e dispositivos.

O SDK também inclui o código-fonte para o Cliente de MQTT para C. É possível adaptar esse código-fonte para construir as bibliotecas do cliente MQTT para C em outras plataformas. Para ajudar a fazer isso, consulte “Construindo o cliente MQTT para as bibliotecas C” na página 31. O código-fonte para o Cliente de MQTT para C também está disponível com uma licença de software livre a partir de Eclipse Paho.

Procedimento

Os artigos a seguir orientarão você através das etapas específicas da plataforma para construir e executar um aplicativo MQTT de amostra em um computador desktop ou em um dispositivo móvel para Android ou a partir de Apple:

- “Introdução ao cliente MQTT para o Java” na página 12
- “Introdução ao Cliente de MQTT para Java no Android” na página 18
- V 7.5.0.1 “Introdução ao Cliente de sistema de mensagens do MQTT para JavaScript” na página 24
- “Introdução ao cliente MQTT para C” na página 27
- “Introdução ao cliente MQTT para C no iOS” na página 48

Como proceder a seguir

Para desenvolver um novo aplicativo MQTT, deve-se ter ou adquirir as seguintes habilidades:

- Programação no idioma necessário para o dispositivo ou plataforma.
- Programação para o dispositivo de destino ou plataforma.
- Projetando os aplicativos de publicação/assinatura.
- Projetando programas para o modelo de programação do MQTT.
- Projetando programas para executar em seu dispositivo móvel escolhido.
- Usando SSL e JAAS para proteger os programas.

Não é necessário nenhuma habilidades de programação de rede para conectar um cliente MQTT a outro dispositivo ou aplicativo, porque o MQTT é um sistema de mensagens e um sistema de enfileiramento. As bibliotecas do cliente MQTT gerenciam as conexões de rede para seu aplicativo.

Para integrar seu cliente MQTT com aplicativos corporativos existentes, você terá duas opções. É possível compartilhar os tópicos de publicação/assinatura do MQTT com um aplicativo (por exemplo) do IBM WebSphere MQ ou JMS ou gravar seu próprio adaptador de integração como outro cliente MQTT.

As fontes de informações para consultar hoje são:

- Desenvolvendo Aplicativos para o WebSphere MQ Telemetry
- MQTT.org
- Eclipse Paho

Conceitos relacionados

“Introdução aos servidores MQTT” na página 138

Introdução ao cliente MQTT para o Java

Esteja em funcionamento com o cliente MQTT para Java aplicativos de amostra, usando IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor. Os aplicativos de amostra usam uma biblioteca do cliente a partir do kit de desenvolvimento de software (SDK) do MQTT da IBM. O aplicativo de amostra `SampleAsyncCallBack` é um modelo para gravar aplicativos MQTT para Android e outros sistemas operacionais orientados a eventos.

Antes de começar

- É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível". Consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.
- Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.

Sobre esta tarefa

O propósito da tarefa é verificar se é possível construir e executar um cliente MQTT para o aplicativo de amostra do Java, conecte-o ao IBM WebSphere MQ ou IBM MessageSight como o servidor MQTT version 3 e troque mensagens.

Siga essa tarefa para executar o aplicativo de amostra no ambiente de trabalho do Eclipse ou em uma linha de comandos. As etapas no exemplo são para o Windows. Com pequenas modificações, é possível executar o aplicativo de amostra em qualquer plataforma que suporta o JSE 1.5 ou acima.

É possível executar aplicativos no mesmo servidor que o IBM WebSphere MQ, em que o ambiente para executar aplicativos que se conectam ao IBM WebSphere MQ é configurado para você. Siga a tarefa “Configurando o serviço MQTT a partir da linha de comandos” na página 142 para instalar e configurar o IBM WebSphere MQ com a opção IBM WebSphere MQ Telemetry no Windows ou Linux. Quando o ambiente for instalado e configurado, execute o aplicativo de amostra, `MQTTV3Sample`, para verificar a instalação.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 . Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte "Introdução aos servidores MQTT" na página 138.
2. Opcional: Configurar o servidor MQTT.
   - No IBM WebSphere MQ, deve-se concluir uma ou outras tarefas a seguir para configurar um gerenciador de fila e configurar seu serviço de telemetria (MQXR):
     - "Configurando o serviço MQTT a partir da linha de comandos" na página 142
     - "Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer" na página 144
   - Em outros servidores, consulte a documentação do servidor. Nenhuma etapa de configuração é necessária para o Really Small Message Broker. Consulte Really Small Message Broker.
3. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.

   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.

   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*\SDK.
4. Instale um Java kit de desenvolvimento (JDK) Versão 6 ou posterior.

   Como você está desenvolvendo um Java app for Android, o JDK deve ser proveniente do Oracle É possível obter o JDK a partir do Java SE Downloads.
5. Compile e execute um ou mais dos cliente MQTT para os aplicativos de amostra do Java:
   - "Compile e execute os programas de amostra do Paho na linha de comandos" na página 14
   - "Compile e execute todos os aplicativos de MQTT amostra de cliente Java a partir do Eclipse" na página 16
   - "Introdução ao Cliente de MQTT para Java no Android" na página 18

   Os apps MQTT client sample Java a seguir estão incluídos no SDK:

   `MQTTV3Sample`
   A amostra também está incluída com o IBM WebSphere MQ e os links para o pacote `com.ibm.micro.client.mqttv3.jar`.

   `Sample`
   `Sample` está no pacote Paho e os links para o pacote `org.eclipse.paho.client.mqttv3`. Ele é semelhante a `MQTTV3Sample`; ele aguarda até que cada ação do MQTT seja concluída.

   `SampleAsyncWait`
   `SampleAsyncWait` está no pacote `org.eclipse.paho.client.mqttv3`. Ele usa a API do MQTT assíncrona; Ele aguarda em um encadeamento diferente até que uma ação é concluída. O encadeamento principal pode fazer outros trabalhos até que ele sincronize no encadeamento que está aguardando que a ação do MQTT seja concluída.

   `SampleAsyncCallBack`
   `SampleAsyncCallBack` está no pacote `org.eclipse.paho.client.mqttv3`. Ele chama o API MQTT assíncrono. O API assíncrono não espera o MQTT concluir o processamento de uma chamada; ele retorna para o aplicativo. O aplicativo continua com as outras tarefas e, em seguida, aguarda o próximo evento chegar para que ele o processar. MQTT posta uma notificação de eventos de volta no aplicativo quando este completa o processamento. O evento orientado à

interface MQTT é adequado ao modelo de programação de serviço e atividade do Android e outros sistemas operacionais orientados à evento.

Como um exemplo, consulte como o `mqttExerciser` de exemplo integra o MQTT no Android usando o modelo de programação de atividade e serviço.

`mqttExerciser`
`mqttExerciser` é um programa de amostra para o Android. Como é construída e executada de forma diferente, será descrito separadamente. Consulte "Introdução ao Cliente de MQTT para Java no Android" na página 18.

Resultados

Você compilou e executou os aplicativos de amostra do MQTT Java que estão conectados ao IBM WebSphere MQ ou IBM MessageSight como o servidor MQTT.

Como proceder a seguir

Analise as informações de referência Javadoc; consulte a etapa "3" na página 17 do "Compile e execute todos os aplicativos de MQTT amostra de cliente Java a partir do Eclipse" na página 16. Como alternativa, abra os arquivos html do Javadoc que estão no diretório `SDK\clients\java\doc\javadoc` no Mobile Messaging and M2M Pacote do Cliente.

Compile e execute os programas de amostra do Paho na linha de comandos

Compile e execute o aplicativo de amostra do Paho `Sample.java` na linha de comandos. A amostra está no MQTT SDK. A amostra é construída com as bibliotecas do cliente MQTT Paho no pacote `org.eclipse.paho.client.mqttv3`. Ela demonstra um publicador e assinante do MQTT. Duas outras amostras Paho no mesmo diretório podem ser construídas e executadas da mesma maneira. Elas diferem chamando a biblioteca do MQTT de forma assíncrona.

Antes de começar

Execute as etapas "1" na página 13 para "4" na página 13 na tarefa principal para configurar um servidor MQTT e faça download do Mobile Messaging and M2M Pacote do Cliente.

Sobre esta tarefa

Compile e execute `Sample.java` no subdiretório de amostras do cliente SDK `SDK\clients\java\samples`. O código de Java está no diretório `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`.

Procedimento

1. Crie um script no diretório de amostras do cliente para compilar e executar `Sample` em sua plataforma escolhida.

   O seguinte script compila e executa a amostra no 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. Compile e execute `Sample.java`

2. Execute o script.

   Resultados:

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

Figura 6. Assinante `MQTTV3Sample`

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

Figura 7. Publicador `MQTTV3Sample`

   Se você deixar a execução do aplicativo assinante, não será possível executar o mesmo assinante novamente. O identificador de cliente do novo assinante é o mesmo que o assinante antigo. Não é possível executar dois clientes MQTT com o mesmo identificador de cliente ao mesmo tempo. É possível configurar a opção `-i` para configurar o identificador do cliente para que seja possível executar assinantes diferentes ao mesmo tempo.

   Se você executar o mesmo cliente novamente, será possível aproveitar a opção `-c` para iniciar e iniciar o cliente com `cleansession` configurado como false. Com essa opção, é possível explorar o comportamento de sessões do cliente interrompida.

3. Termine o assinante, pressionando Enter ou fechando a janela.

Como proceder a seguir

Crie scripts para compilar e executar as outras amostras no subdiretório `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`. Copie o script no Figura 5 na página 15 e substitua `Sample` por `SampleAsyncWait` ou `SampleAsyncCallBack`. As outras amostras são versões assíncronas do programa síncrono `Sample`.

`SampleAsyncWait`
`SampleAsyncWait` está no pacote `org.eclipse.paho.client.mqttv3`. Ele usa a API do MQTT assíncrona; Ele aguarda em um encadeamento diferente até que uma ação é concluída. O encadeamento principal pode fazer outros trabalhos até que ele sincronize no encadeamento que está aguardando que a ação do MQTT seja concluída.

`SampleAsyncCallBack`
`SampleAsyncCallBack` está no pacote `org.eclipse.paho.client.mqttv3`. Ele chama o API MQTT assíncrono. O API assíncrono não espera o MQTT concluir o processamento de uma chamada; ele retorna para o aplicativo. O aplicativo continua com as outras tarefas e, em seguida, aguarda o próximo evento chegar para que ele o processar. MQTT posta uma notificação de eventos de volta no aplicativo quando este completa o processamento. O evento orientado à interface MQTT é adequado

ao modelo de programação de serviço e atividade do Android e outros sistemas operacionais orientados à evento.

Como um exemplo, consulte como o `mqttExerciser` de exemplo integra o MQTT no Android usando o modelo de programação de atividade e serviço.

As amostras assíncronas demonstram como reduzir a quantidade de tempo que um aplicativo MQTT é bloqueado enquanto ele está aguardando o cliente MQTT. É importante para eliminar chamadas de bloqueio no encadeamento principal para aumentar a vida útil e a resposta da bateria em um de dispositivo móvel.

As amostras demonstram dois padrões para chamar as interfaces assíncronas no cliente MQTT.

1. `SampleAsyncWait` não será bloqueado enquanto o MQTT estiver aguardando por interações de rede.
2. `SampleAsyncCallBack` não será bloqueado ao aguardar o cliente MQTT concluir quaisquer ações. O anterior será necessário quando uma página do JavaScript estiver chamando o cliente MQTT a partir de um navegador. Páginas do JavaScript não devem bloquear. As respostas para ações devem ser postadas de volta para o encadeamento do navegador principal, que chama o manipulador de eventos do MQTT gravado para processar a notificação.

Compile e execute todos os aplicativos de MQTT amostra de cliente Java a partir do Eclipse

Compile e execute os apps MQTT client sample Java que estão no Mobile Messaging and M2M Pacote do Cliente. Eles demonstram um publicador e assinante do MQTT.

Sobre esta tarefa

Compile e execute as amostras do MQTT Java, `Sample` e `MQTTV3Sample` no Eclipse. `Sample` está no subdiretório de clientes SDK *sdkroot*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` e `MQTTV3Sample.java` está em *sdkroot*`\SDK\clients\java\samples`.

Procedimento

1. Faça download do Eclipse IDE for Java Developers.
2. Crie um projeto Java chamado `MQTT Samples` em Eclipse.
   a) **Arquivo** > **Novo** > **Projeto Java** e digite `MQTT Samples`.. Clique em **Avançar**.

      Verifique se o JRE está na versão correta ou mais recente.JSE deve estar na versão 1.5 ou posterior.

   b) Na janela **Configurações de Java**, clique em **Vincular pastas de origem adicionais**.

   c) Navegue até o diretório no qual você instalou a pasta SDK do MQTT Java. Selecione a pasta *sdkroot*`\SDK\clients\java\samples` e clique em **OK** > **Avançar** > **Concluir**

   d) Na janela **Configurações de Java**, clique em **Bibliotecas** > **Incluir jars externos**

   e) Navegue até o diretório no qual você instalou a pasta SDK do MQTT Java. Localize a pasta *sdkroot*`\SDK\clients\java` e selecione os arquivos `org.eclipse.paho.client.mqttv3.jar` e `com.ibm.micro.client.mqttv3.jar` ; clique em **Abrir** > **Concluir**

      A amostra `MQTTV3Sample.java` se vincula ao `com.ibm.micro.client.mqttv3.jar` e as amostras na árvore de diretórios paho se vinculam ao `org.eclipse.paho.client.mqttv3.jar`. O `com.ibm.micro.client.mqttv3.jar` é retido para que os aplicativos MQTT existentes continuem a construir e executar sem mudança.

   O projetos `MQTT Samples` é construído com alguns avisos, mas não com erros.

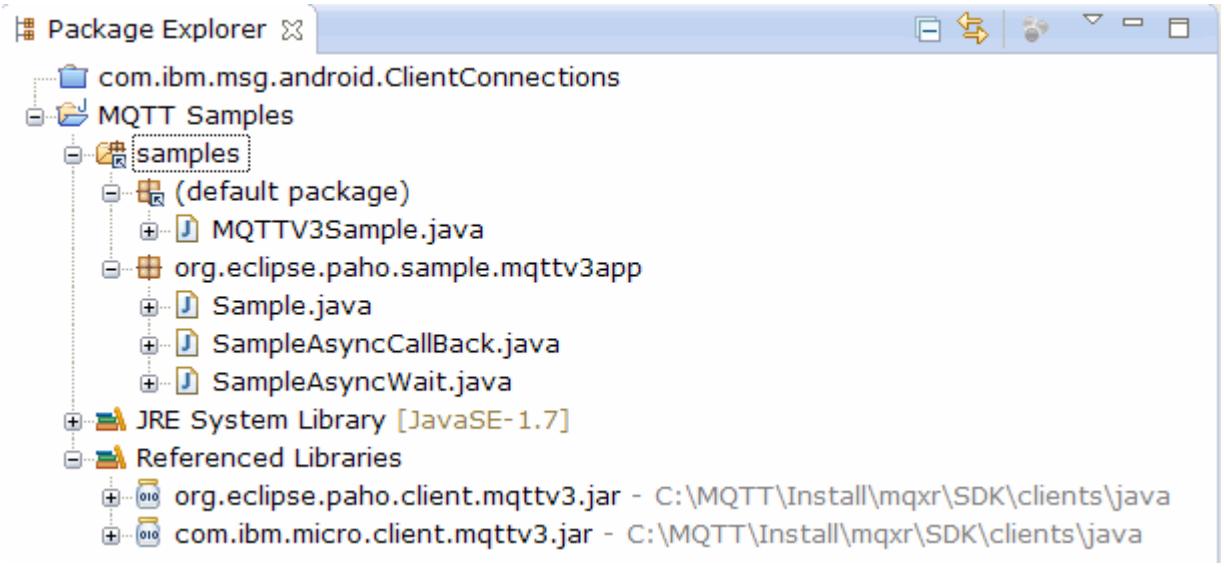


Figura 8. MQTT cliente para o projeto Java

3. Opcional: Instale o cliente MQTT Javadoc.

   Com o MQTT cliente Javadoc instalado, o editor Java descreve as classes MQTT na ajuda instantânea..

   a) Abra o **Package Explorer** > **Bibliotecas de Referência** em seu projeto Java. Clique com o botão direito em `org.eclipse.paho.client.mqttv3.jar` **> Propriedades**.

   b) No navegador Propriedades, clique em **Local do Javadoc**.

   c) Clique em **URL Javadoc** > **Procurar** na página **Local Javadoc** e localize a `SDK\clients\java\doc\javadoc` pasta **> OK**.

   d) Clique em **Validar** > **OK**

      Você recebeu um aviso para abrir um navegador para visualizar a documentação.

   e) Repita este procedimento para o arquivo `com.ibm.micro.client.mqttv3.jar`.

4. Crie uma configuração de tempo de execução do publicador e do assinante para executar o aplicativo `mqttv3app.Sample`.

   a) Clique com o botão direito na classe **Amostra** , clique em **Executar como** > **Configurações de execução**.

   b) Clique com o botão direito em **Aplicativo Java** > **Novo** e digite o nome `SampleSubscriber`.

   c) Clique na guia argumentos e digite os argumentos de programa seguidos por **Aplicar**.

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

   d) Repita a última etapa para criar uma configuração `SamplePublisher` omitindo o parâmetro `-a subscribe`.

5. Execute o assinante `mqttv3app.Sample` seguido pelo publicador.

   a) Clique em **Executar** > **Executar configurações**

   b) Clique em **SampleSubscriber** > **Executar**..

      Abra a visualização **Console**. O assinante está aguardando uma publicação.

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

   c) Clique em **SamplePublisher** > **Executar**

Abra a visualização **Console**. Consulte a publicação criada pelo publicador:

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

d) Visualizações do console do comutador para o console do assinante.

O ícone de consoles do comutador é  .

O assinante receber a publicação:

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

6. Opcional: Crie uma configuração de tempo de execução do publicador e do assinante para o `MQTTV3Sample`.
   a) Clique com o botão direito na classe **MQTTV3Sample** , clique em **Executar como** > **Executar configurações**.
   b) Clique com o botão direito em **Aplicativo Java** > **Novo** e digite o nome `MQTTV3SampleSubscriber`.
   c) Clique na guia argumentos e digite os argumentos de programa seguidos por **Aplicar**.

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

   d) Repita a última etapa para criar uma configuração `MQTTV3SamplePublisher` omitindo o parâmetro `-a subscribe`.
7. Opcional: Execute o assinante `MQTTV3Sample` seguido pelo publicador.
   a) Clique em **Executar** > **Executar configurações**
   b) Clique em **MQTTV3SampleSubscriber** > **Executar**..

Abra a visualização **Console**. O assinante está aguardando uma publicação.

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

   c) Clique em **MQTTV3SamplePublisher** > **Executar**..

Abra a visualização **Console**. É possível ver a publicação criada pelo publicador.

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

   d) Visualizações do console do comutador para o console do assinante.

O ícone de consoles do comutador é  .

O assinante receber a publicação:

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

Introdução ao Cliente de MQTT para Java no Android

É possível instalar um Aplicativo de amostra do cliente MQTT Java para Android que troca mensagens com um servidor MQTT. O aplicativo usa uma biblioteca do cliente MQTT SDK a partir da IBM. É possível construir o aplicativo por conta própria ou fazer download de um aplicativo de amostra pré-construído.

Antes de começar

- Para plataformas do cliente MQTT com suporte e referência, consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.
- Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.
- O aplicativo de amostra do cliente MQTT funciona no Ice Cream Sandwich (Android 4.0) e acima. Esta versão do Android também fornece uma resolução de exibição assertiva em tablets.

Sobre esta tarefa

O Aplicativo de amostra do cliente MQTT Java para Android é chamado de "`mqttExerciser`". Este aplicativo usa uma biblioteca do cliente a partir do MQTT SDK e troca mensagens com um servidor MQTT.

É possível construir o aplicativo de amostra você mesmo, então exporte-o a partir do Eclipse como `mqttExerciser.apk` ou use o aplicativo de amostra pré-construído disponível como um arquivo `mqttExerciser.apk` na pasta *`sdkroot`*`\SDK\clients\android\samples\apks` do Mobile Messaging and M2M Pacote do Cliente. Se escolher construir o aplicativo você mesmo, o ambiente de desenvolvimento construído será customizado para incluir o sistema de mensagens móveis em aplicativos for Android. Isso deverá ajudá-lo quando você começar a incluir o sistema de mensagens móveis em seus próprios aplicativos.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 . Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte “Introdução aos servidores MQTT” na página 138.
2. Obtenha as ferramentas certas.

   Instale um Java kit de desenvolvimento (JDK) Versão 6 ou posterior. Como você está desenvolvendo um Java app for Android, o JDK deve ser proveniente do Oracle É possível obter o JDK a partir do Java SE Downloads.

   Também é necessário um ambiente de desenvolvimento do Eclipse. Isso deve ser Eclipse 3.6.2 (Helios) ou maior. Eclipse deve ter um nível de compilador Java de pelo menos 6, para corresponder ao JDK. É possível obter tudo isso no Eclipse Foundation.

   Finalmente, você precisa do Android SDK. É possível obter isso a partir de Obter o Android SDK.
3. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.

   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.

   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*`\SDK`.
4. Opcional: Construa o aplicativo de amostra do mqttExerciser for Android.

   Configure as ferramentas do Eclipse e Android, importe e construa o projeto `mqttExerciser` a partir doMQTT SDK.

   Nota: Se você não desejar fazer isso agora, será possível usar o aplicativo de amostra pré-construído disponível como arquivo `mqttExerciser.apk` na pasta *`sdkroot`*`\SDK\clients\android\samples\apks` doMQTT SDK.

   a) Inicie o ambiente de desenvolvimento do Eclipse com o JRE a partir do JDK.

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

b) Selecione e instale um conjunto de pacotes e plataformas a partir do Android SDK.

Consulte Incluindo plataformas e pacotes para a lista de plataformas e pacotes que o Google recomenda.

Nota: A plataforma SDK deve ser o nível 16 da API do Android ou posterior. Com níveis das APIs anteriores, o projeto não pode ser compilado com sucesso.

c) Inclua o plug-in do AndroidDevelopment Tools (ADT) para Eclipse.

d) Importe o projeto do aplicativo `mqttExerciser` de amostra em Eclipse e corrija os erros.

i) Importe o projeto do aplicativo de amostra do MQTT SDK, no caminho *sdkroot*`\SDK\clients\android\samples\mqttExerciser`.

A visualização **Problemas** lista muitos erros de construção. Resolva os erros de construção em algumas das próximas etapas.

ii) Copie a biblioteca `org.eclipse.paho.client.mqttv3.jar` na pasta **libs** no projetoAndroid. Windows Por exemplo, no Windows, isso está na pasta *sdkroot*`\SDK\clients\java`. Uma janela **Operação do arquivo** é exibida. Aceite a seleção **Copiar arquivos**, em seguida, clique em **OK**.

iii) Clique com o botão direito na pasta do projeto, `com.ibm.msg.android`; clique em **Ferramentas Android ...** > **Inclua a Biblioteca de Suporte ...** Leia e aceite os termos de licença, em seguida, clique em **Instalar**.

iv) Clique com o botão direito na pasta do projeto, `com.ibm.msg.android`; clique em **Ferramentas Android ...** > **Propriedades do Projeto de Correção**

v) Se o espaço de trabalho ainda tiver cerca de 84 erros, se referindo a substituição de um método de classe super, o nível de conformidade do compilador será provavelmente configurado para 1.5 ou inferior. O Android SDK versão 16 espera que o nível de conformidade do compilador não seja maior que 1.5. Para corrigir os erros restantes, conclua as seguintes etapas:

a) Verifique e (se necessário) atualize seu Android SDK e os plug-ins correspondentes do Eclipse para Android SDK versão 17.

b) Clique com o botão direito na pasta do projeto **com.ibm.msg.android** e, em seguida, selecione **Propriedades** > **Compilador Java**. Verifique o nível de conformidade do compilador, configure-o para pelo menos 1.6, em seguida, reconstrua o espaço de trabalho.

O projeto é construído, com alguns avisos e nenhum erro.

5. Instale e inicie o Aplicativo de amostra do cliente MQTT Java em um dispositivo Android.

Consulte a página `developer.android.com` Executando seu aplicativo.

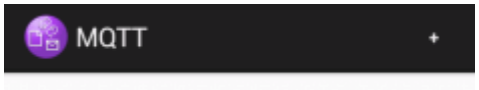
Se você construiu o aplicativo você mesmo como um projeto Eclipse, será possível iniciar o aplicativo a partir do Eclipse.

Se você tiver o arquivo de pacote de aplicativos (APK) `mqttExerciser.apk`, será possível instalá-lo fora do Eclipse usando o comando de instalação do Android Debug Bridge (ADB). Este comando usa o local do arquivo APK como um argumento. Se você estiver usando o aplicativo de amostra pré-construído, o local será *sdkroot*`\SDK\clients\android\samples\apks\mqttExerciser.apk`.

6. Use o aplicativo de amostra do mqttExerciserfor Android para se conectar, assinar e publicar em um tópico.

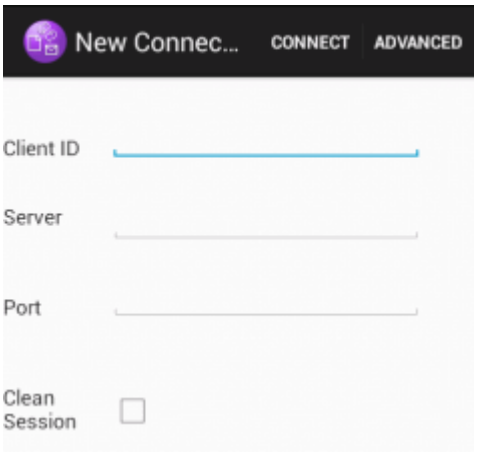
a) Abra o Aplicativo de amostra do cliente MQTT Java para Android.

Esta janela é aberta em seu dispositivo Android:



b) Conecte-se a um servidor MQTT .

i) Clique no sinal **+** para abrir uma conexão MQTT nova.



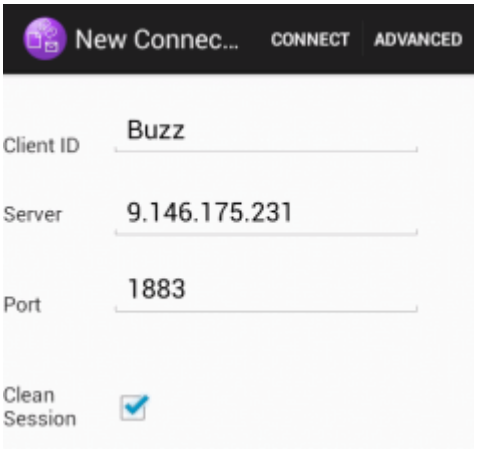
ii) Digite qualquer identificador exclusivo no campo **ID do cliente**. Seja paciente, os pressionamentos de tecla podem ser lentos.

iii) Insira no campo **Servidor** o endereço IP de seu servidor MQTT.

Este é o servidor que você escolheu na primeira etapa principal. O endereço IP não deve ser `127.0.0.1`

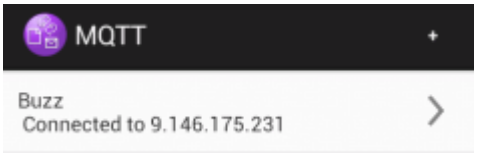
iv) Insira o número da porta da conexão do MQTT.

O número da porta padrão para uma conexão normal do MQTT é 1883.



v) Clique em **Conectar**.

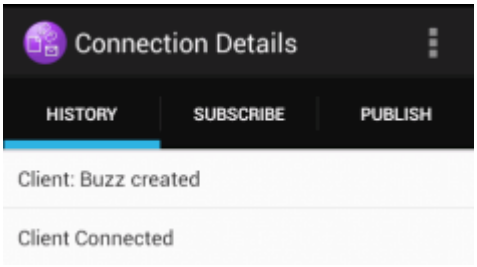
Se a conexão for bem-sucedida, você verá uma mensagem "Conectando", seguida por esta janela:



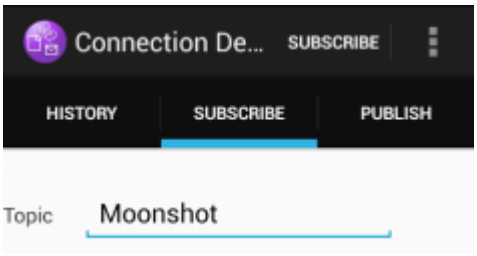
c) Assine em um tópico.

i) Clique na mensagem **Conectado**.

A janela **Detalhes da conexão** é aberta com o histórico listado:



ii) Clique na guia **Assinar** e insira uma sequência de tópicos.

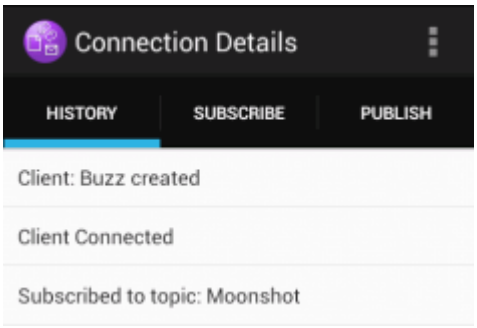


iii) Clique na ação **Assinar**.

Uma mensagem "assinada" aparece por um breve período.

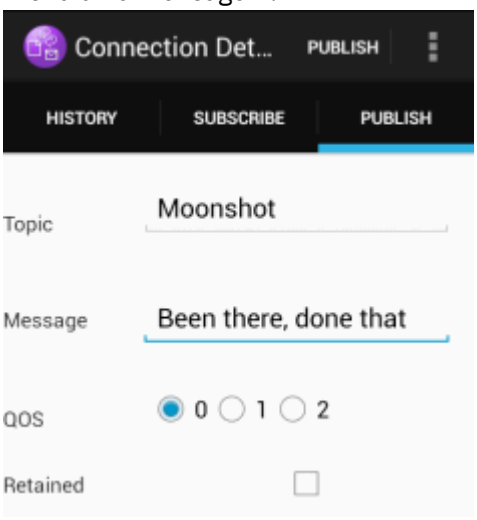
iv) Clique na guia **Histórico**.

O histórico agora inclui a assinatura:



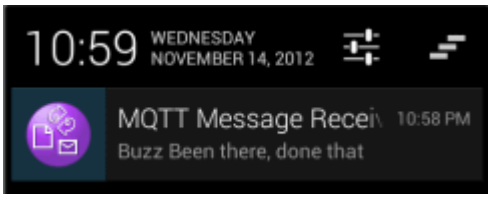
d) Publicar agora para o mesmo tópico.

i) Clique na guia **Publicar** e insira a mesma sequência de tópicos que você fez para a assinatura. Insira uma mensagem.

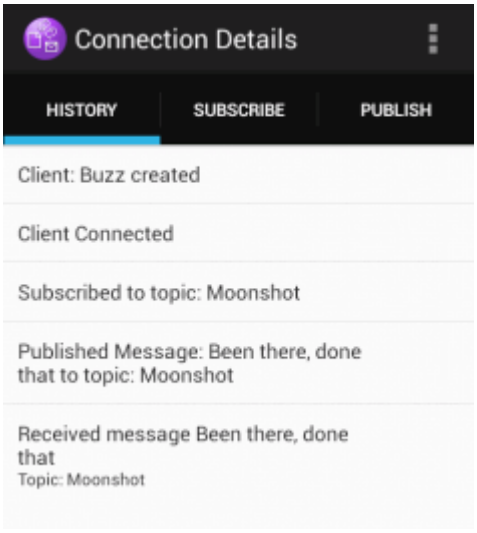


ii) Clique na ação **Publicar**.

Duas mensagens são exibidas por um breve período, "Publicado" seguido por "Assinado". A publicação é exibida na área de status (puxe a barra separadora para baixo para abrir a janela de status).



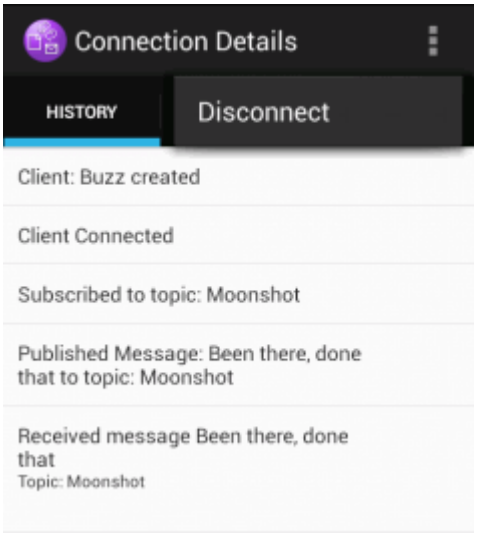
iii) Clique na guia **Histórico** para visualizar o histórico completo.



e) Desconecte a instância do cliente.

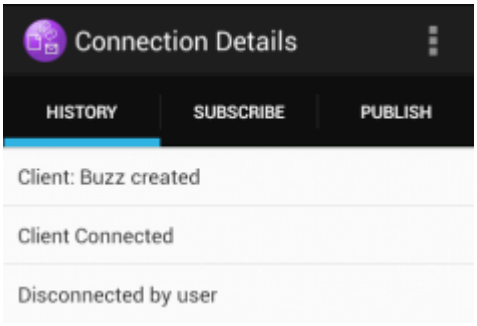
i) Clique no ícone de menu na barra de ação. 

O Aplicativo de amostra do cliente MQTT Java para Android inclui um botão **Desconectar** na janela MQTT **Detalhes da Conexão** .

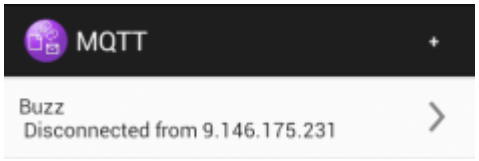


ii) Clique em **Desconectar**.

O status conectado muda para desconectado:



f) Clique em **Voltar** para retornar à lista de sessões do Aplicativo de amostra do cliente MQTT Java.



- Clique no sinal de mais para iniciar uma nova sessão do Aplicativo de amostra do cliente MQTT Java.
- Clique no cliente desconectado para reconectá-lo.
- Clique em **Voltar** para retornar para a barra de ativação.

g) Clique no botão de tarefa para listar os aplicativos em execução. Localize o Aplicativo de amostra do cliente MQTT Java. Deslize os dedos no ícone da tela para fechá-lo.

Como proceder a seguir

Se você construir o aplicativo de amostra você mesmo, você estará pronto para iniciar o desenvolvimento de seus próprios aplicativos Android que chamará as bibliotecas do MQTT para trocar mensagens. É possível modelar seus aplicativos Android nas classes em `mqttExerciser`. Para analisar a amostra, gere o Javadoc para as classes em `com.ibm.msg.android` e `com.ibm.msg.android.service` no projeto `mqttExerciser`.

Informações relacionadas

Gerenciamento de projetos a partir do Eclipse com ADT

Introdução ao Cliente de sistema de mensagens do MQTT para JavaScript

É possível iniciar com o Cliente de sistema de mensagens do MQTT para JavaScript, exibindo a página inicial de amostra do cliente com o sistema de mensagens e procurando os recursos aos quais ele se vincula. Para exibir esta página inicial, configure um servidor MQTT para aceitar conexões do Páginas JavaScript de amostra do cliente de sistema de mensagens MQTT, em seguida, digite a URL configurada no servidor em um navegador da web. O Cliente de sistema de mensagens do MQTT para JavaScript inicia automaticamente em seu dispositivo e a página inicial de amostra do cliente com o sistema de mensagens é exibida. Esta página contém links para utilitários, documentação da interface de programação, um tutorial e outras informações úteis.

Antes de começar

Para uso avançado ou em produção, você desejará reformular ou remover a página inicial de amostra do cliente com o sistema de mensagens. Observe que as interfaces do usuário que resultam do código de amostra não são garantidas para estar em conformidade com quaisquer padrões de acessibilidade ou com os requisitos de acessibilidade.

É necessário um servidor MQTT para suportar o Cliente de sistema de mensagens do MQTT para JavaScript. Este servidor deve suportar o protocolo MQTT V3.1 sobre WebSockets. IBM MessageSight, e IBM WebSphere MQ Version 7.5.0, Fix Pack 1 e versões mais recentes, suportem o MQTT protocol over WebSockets Consulte o “Introdução aos servidores MQTT” na página 138. Para instalar o IBM

WebSphere MQ para uma avaliação de 90 dias livres, consulte “Instalando o IBM WebSphere MQ” na página 140.

O WebSocket protocol foi estabelecido recentemente. Se houver um firewall entre o cliente e o servidor, verifique se ele não bloqueia o tráfego do WebSockets. Da mesma forma, se o seu navegador ainda não suportar o WebSocket protocol[1]você não poderá usar o utilitário do cliente ou os tutoriais disponíveis a partir da página inicial de amostra do cliente com o sistema de mensagens. A tabela Tabela 1 na página 25 lista os navegadores com versões mais recentes que foram testadas e mostradas para trabalhar com o cliente do sistema de mensagens.

Tabela 1. Navegadores com suporte para uso com o Cliente de sistema de mensagens do MQTT para JavaScript

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

Sobre esta tarefa

A maioria das etapas nesta tarefa são para configurar o servidor MQTT. Tudo o que é necessário para acessar o cliente do sistema de mensagens para o JavaScript é executar um navegador que suporte o WebSocket protocol.

No IBM WebSphere MQ, siga as etapas para ativar o IBM WebSphere MQ Telemetry criando os canais de amostra. Conecte-se ao canal de amostra padrão do MQTT WebSockets na porta 1883. A URL da página inicial de amostra do cliente com o sistema de mensagens será `http://`*`hostname`*`:1883` no IBM WebSphere MQ.

No IBM MessageSight, instale e configure o dispositivo, configure o hub do sistema de mensagens para aceitar conexões e crie um terminal do MQTT WebSockets.

Procedimento

1. Faça o download do Mobile Messaging and M2M Pacote do Cliente e escolha um servidor MQTT ao qual seja possível conectar o aplicativo cliente.

   Consulte “Introdução aos clientes MQTT” na página 11.
2. Configure o servidor MQTT para aceitar as conexões a partir das páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript.
   - Em IBM WebSphere MQ:
     - Se você já tiver um gerenciador de filas do IBM WebSphere MQ configurado para MQTT, altere o protocolo na definição de canal para suportar ambos MQTT e HTTP. Consulte **`ALTER CHANNEL`**.
     - Para criar um gerenciador de filas do IBM WebSphere MQ e configurar o terminal de amostra do MQTT WebSockets , conclua uma das tarefas a seguir:
       - “Configurando o serviço MQTT a partir da linha de comandos” na página 142
       - “Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer” na página 144
3. Abra um navegador da web em seu dispositivo.
4. Digite a URL da página inicial de amostra do cliente com o sistema de mensagens.
   - No IBM WebSphere MQ, isto é `http://`*`hostname`*`:1883`
   - No IBM MessageSight, isto é `http://`*`hostname`*`:`*`port`*

[1] Especificamente, se ele não suportar o padrão RFC 6455 (WebSocket).

em que *hostname* é o nome DNS ou endereço IP do soquete Ethernet configurado em seu dispositivo IBM MessageSight como o terminal para o qual o cliente está para se conectar e *port* é o número da porta TCP/IP designado para o terminal para o cliente.

A página inicial de amostra do cliente com o sistema de mensagens é exibida.

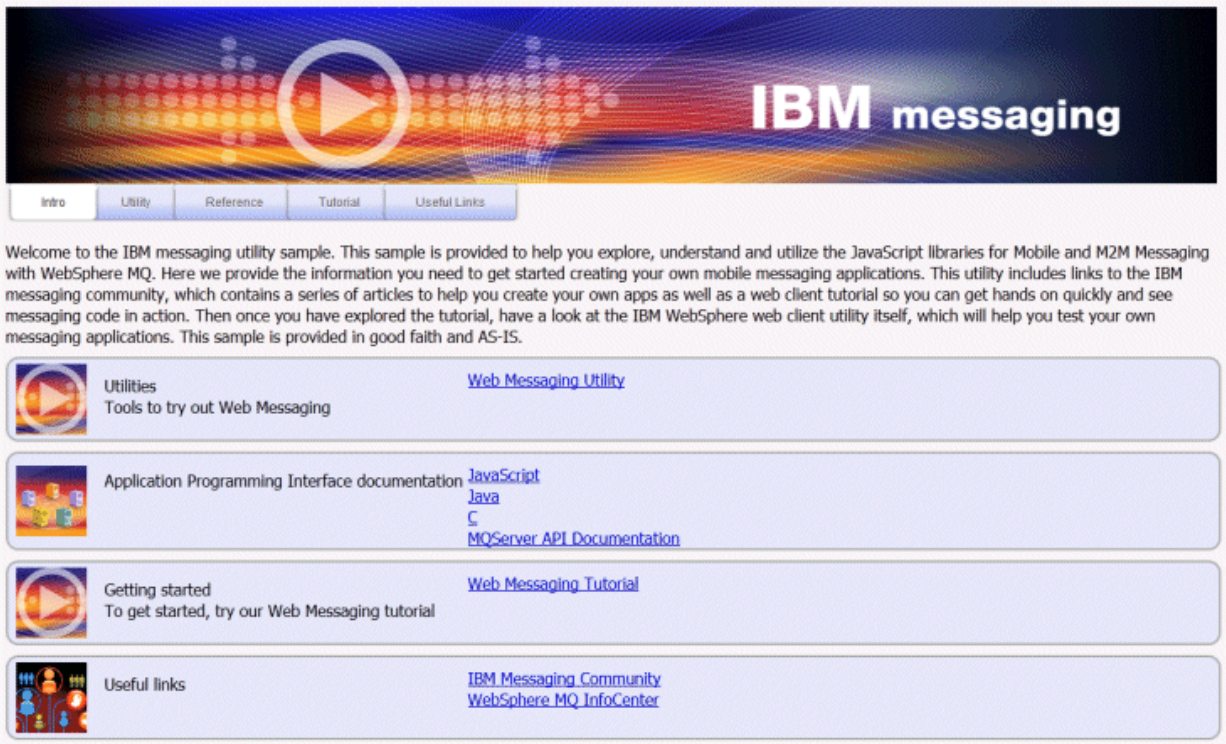


Figura 9. Página inicial de amostra doCliente de sistema de mensagens do MQTT para JavaScript

Resultados

Você configurou um canal do MQTT para WebSockets.

Na página inicial de amostra do Cliente de sistema de mensagens do MQTT para JavaScript, clique em **Utilitário do sistema de mensagens da web** para experimentar diferentes funções na API do cliente do sistema de mensagens. Por exemplo, é possível se conectar ao gerenciador de filas, assinar para as mensagens, em seguida, publicar algumas mensagens. Também é possível clicar em **Tutorial do sistema de mensagens da web** para aprender a criar uma página da web que chama o cliente do sistema de mensagens MQTT para a API do JavaScript.

Conceitos relacionados

"O Cliente de sistema de mensagens do MQTT para JavaScript e os aplicativos da web" na página 117

"Como programar os aplicativos do sistema de mensagens no JavaScript" na página 121

Tarefas relacionadas

"Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets" na página 78

Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

Introdução ao cliente MQTT para C

Funcionamento adequado com o cliente MQTT de amostra para C em qualquer plataforma na qual é possível compilar a origem C. Verifique se é possível executar o cliente MQTT de amostra para C com o IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

Antes de começar

- Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.
- Para o cliente MQTT com suporte e referência para plataformas C. Consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.

Sobre esta tarefa

Siga essa tarefa para compilar e executar o cliente MQTT de amostra para C no Windows a partir da linha de comandos ou a partir do Microsoft Visual Studio 2010. O Microsoft Visual Studio 2010 também é usado para compilar o cliente no exemplo da linha de comandos. Modifique os scripts da linha de comandos para compilar e executar a amostra em outras plataformas.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 . Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte "Introdução aos servidores MQTT" na página 138.
2. Instale um ambiente de desenvolvimento C na plataforma na qual estiver construindo.

   Os arquivos de criação nos exemplos neste tópico são voltados para as seguintes ferramentas:

   - iOS Para iOS, no Apple Mac com OS X 10.8.2 com as ferramentas de desenvolvimento iOS de Xcode.
   - Linux Para o Linux, a versão gcc 4.4.6 a partir do Red Hat® Enterprise Linux versão 6.2.

     O nível mínimo suportado da biblioteca C `glibc` é 2.12, e o kernel Linux é 2.6.32.
   - Windows Para o Microsoft Windows, Visual Studio versão 10.0.
3. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.

   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.

   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*\SDK.
4. Opcional: Siga as etapas em "Construindo o cliente MQTT para as bibliotecas C" na página 31.

   Execute essa etapa apenas se o Mobile Messaging and M2M Pacote do Cliente não incluir a biblioteca do cliente C para sua plataforma de destino.
5. Compile e execute o aplicativo C de amostra do cliente MQTT, `MQTTV3Sample.c`.

   - Na linha de comandos, siga as etapas em "Compile e execute o aplicativo C de amostra do cliente MQTT a partir da linha de comandos" na página 28.
   - A partir de um IDE, siga as etapas em "Compile e execute o aplicativo C de amostra do cliente MQTT do Microsoft Visual Studio" na página 28.

Compile e execute o aplicativo C de amostra do cliente MQTT a partir da linha de comandos

Compile e execute o aplicativo C do cliente MQTT de amostra a partir da linha de comandos. A amostra está no MQTT SDK. Ela demonstra um publicador e assinante do MQTT.

Antes de começar

Instale um ambiente de desenvolvimento C; por exemplo, o Microsoft Visual Studio 2010 conforme usado no exemplo.

Sobre esta tarefa

Compile e execute a amostra C, `MQTTV3Sample`, no subdiretório de clientes SDK, em *sdkroot*`\SDK\clients\c\samples`.

Procedimento

Crie um script no diretório de amostras do cliente para compilar e executar `Sample` em sua plataforma escolhida.

O seguinte script compila e executa a amostra em uma plataforma do Windows de 32 bits, construída com o Microsoft Visual Studio 2010. Execute o script no subdiretório *sdkroot*`\SDK\clients\c\samples`.

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

Resultados

A saída de gravação do publicador e do assinante para as janelas de comando:

```
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. A saída do publicador

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

Figura 11. A saída do assinante

Compile e execute o aplicativo C de amostra do cliente MQTT do Microsoft Visual Studio

Compile e execute o aplicativo C de amostra do cliente MQTT do Microsoft Visual Studio. A amostra está no Mobile Messaging and M2M Pacote do Cliente. Ela demonstra um publicador e assinante do MQTT.

Antes de começar

O exemplo usa o Microsoft Visual Studio 2010. É possível usar outros ambientes de desenvolvimento C no Windows e outras plataformas; por exemplo Eclipse IDE for C/C++ Developers.

Sobre esta tarefa

Compile e execute a amostra C, `MQTTV3Sample` com o Microsoft Visual Studio 2010. `MQTTV3Sample.c` está no subdiretório de clientes SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedimento

1. Inicie o Microsoft Visual Studio.
2. Criar um novo projeto a partir do código existente.
   a) Clique em **Arquivo** > **Novo** > **Projeto do Código Existente**.
   b) Selecione **Visual C++** como o tipo de projeto a ser criado.
   c) Clique em **Avançar**.
3. Especifique os parâmetros na janela **Local do projeto e arquivos de origem**.
   a) Clique em **Procurar** e localize o diretório *sdkroot*`\SDK\clients\c\samples`.
   b) Nomeie o projeto `MQTTV3Sample`.
   c) Clique em **Avançar**.
4. Selecione **Projeto de aplicativo do console** na lista **Tipo de projeto**. Clique em **Concluir**
5. Configure apenas a configuração de depuração.

   Por padrão, o Microsoft Visual Studio cria ambos uma liberação e uma configuração de depuração. No tutorial, defina a configuração de depuração. Para suprimir erros de construção, limpe a opção **Construir** para a configuração de liberação.

   a) Clique em **Projeto** > **MQTTV3Sample Propriedades** > **Gerenciador de configuração** Selecione **Liberação** como a **Configuração de solução ativa** e limpe **Construir**.
   b) Selecione **Depurar** como a **Configuração da solução ativa** > **Fechar**

   Verifique se você está modificando a configuração de depuração em todas as etapas a seguir.
6. Modifique as configurações do **C/C++** nas **Páginas de propriedade do MQTTV3Sample**.
   a) Na janela **MQTTV3Sample** , abra **Propriedades de Configuração** > **C/C++** > **Geral**
   b) Na lista de propriedades gerais, clique em **Incluir diretórios adicionais** e inclua seu caminho de diretório para *sdkroot*`\SDK\clients\c\include` e clique em **Aplicar**.
7. Modifique as configurações do **Vinculador**
   a) Abra **Propriedades de configuração** > **Vinculador** > **Geral**
   b) Na lista de propriedades gerais, clique em **Diretórios adicionais da biblioteca** e inclua seu caminho de diretório para *sdkroot*`\SDK\clients\c\windows_ia32`
   c) Na lista de propriedades do Vinculador, clique em **Linha de comandos**. Digite `mqttv3c.lib` na área de entrada de dados**Opções adicionais** e clique em **Aplicar**.
8. Remova o arquivo de origem `MQTTV3SSample.c` do projeto.
   a) Abra a pasta **MQTTV3sample** > **Arquivos de origem** na janela **Solution Explorer** .
   b) Clique com o botão direito em **MQTTV3SSample.c** > **Excluir do Projeto**
9. Construa o projeto `MQTTV3Sample`.
   a) Clique com botão direito no projeto **MQTTV3sample** no Solution Explorer e clique em **Construir**.

   A construção é concluída sem nenhum erro.
10. Inclua dois novos projetos para executar o **`MQTTV3Sample`** como uma instância de tempo de execução do assinante e do publicador.

Os projetos Publicador e Assinante devem conter os comandos para executar o **MQTTV3Sample**. Eles não são construídos e não contêm nenhum código.

a) No **Solution Explorer**, clique com o botão direito do mouse em **Solution ` MQTTV3Sample `** > **Incluir** > **Novo projeto**.

b) Digite `Subscriber` no campo **Nome** . Deixe **Console de aplicativo do Win32** selecionado. Clique em **OK**.

O **Assistente de aplicativo do Win32** inicia.

c) No **Assistente de aplicativo do Win32**, clique em **Avançar**. Verifique **Esvaziar projeto** > **Concluir**

d) Repita essas etapas para incluir um projeto Publicador.

e) Clique com o botão direito no projeto Assinante e clique em **Configurar como projeto Inicialização**

A janela **Solution Explorer** é mostrada em Figura 12 na página 30.

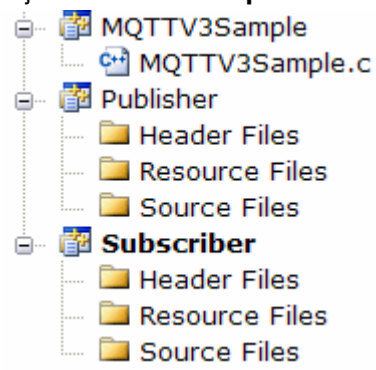


Figura 12. Solução de MQTTV3Sample

11. Configure as páginas de propriedade do assinante.

a) Clique com o botão direito em **Assinante** no Explorador de Soluções **Propriedades** > **Propriedades de Configuração** > **Propriedades** > **Depuração**

Verifique se o título da janela será **Páginas de propriedade do assinante**.

b) Clique em **Ambiente**. Digite `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` e clique em **Aplicar**.

Altere *`sdkroot`* para se adequar ao seu ambiente.

c) Clique em **Comando** e substitua `$(TargetPath)` pelo caminho para o módulo `MQTTV3Sample`

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

Altere *`sdkroot`* para se adequar ao seu ambiente.

d) Clique em **Argumentos de Comando** e digite `-a subscribe -b localhost -p 1883` e clique em **Aplicar**.

12. Configure as páginas de propriedade do publicador.

Sugestão: É possível alternar o projeto de páginas de propriedades, clicando nos projetos na janela **Solution Explorer**.

a) Repita as etapas do assinante para o publicador.

O argumento de comando é `-b localhost -p 1883`

13. Pare o processo de construção que constrói os projetos do publicador e do assinante.

a) Nas páginas de propriedades de qualquer um dos projetos, clique em **Gerenciador de configuração** e limpe **Construir** para o publicador e o assinante nas configurações Liberação e Depuração. Clique em **Fechar**.

14. Executar a amostra.

a) Clique em **F5** para iniciar o assinante

b) Clique com o botão direito em **Publicador** no Solution Explorer, **Depurar** > **Iniciar uma nova instância**

Resultados

A saída do publicador e do assinante para as janelas de comandos. Visual Studio fecha a janela do publicador. Consulte a janela do assinante, que é mostrada na figura a seguir, em seguida, feche a janela do assinante.

```
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. A saída do assinante

Como proceder a seguir

Construa e execute o publicador e o assinante assíncrono.Os exemplos são `MQTTV3ASample.c` e `MQTTV3ASSample.c` em *sdkroot*`\SDK\clients\c\samples`

Construindo o cliente MQTT para as bibliotecas C

Siga estas etapas para construir o cliente MQTT para as bibliotecas C. O tópico inclui os comutadores de compilação e de link para um número de plataformas e exemplos de construção das bibliotecas no iOS e Windows.

Antes de começar

1. Construa a biblioteca do cliente C apenas quando necessário. Vincule as bibliotecas do cliente pré-construídas no (Software Development Kit) SDK no subdiretório SDK`\clients\c` se um corresponder à sua plataforma de destino.
2. Configure um servidor MQTT para testar a biblioteca construída com o Aplicativo C de amostra do cliente do MQTT. Consulte o “Introdução aos servidores MQTT” na página 138. Verifique a configuração do servidor executando um dos aplicativos de amostra do cliente MQTT.
3. Se você está construindo uma versão segura da biblioteca C, que suporta (Secure Sockets Layer) SSL, deve-se também construir a biblioteca do OpenSSL. Consulte “Construindo o pacote do OpenSSL” na página 45.

   Importante: O download e a redistribuição do pacote OpenSSL estão sujeitos a regulamentações rigorosas de importação e exportação e condições de licenciamento de software livre. Tome cuidadosa nota das restrições e avisos antes de decidir fazer download do pacote.

Sobre esta tarefa

Siga as instruções em “Construindo o pacote do OpenSSL” na página 45 para fazer download e construir a biblioteca do OpenSSL. Deve-se construir o OpenSSL para construir uma versão segura do cliente MQTT para a biblioteca C. Você não requer o OpenSSL para construir uma versão descoberta da biblioteca do MQTT. As etapas incluem exemplos de construção da biblioteca para o iOS e Windows.

Construa o cliente MQTT para a biblioteca C fazendo o download das ferramentas de biblioteca de desenvolvimento C e do kit de desenvolvimento de software (SDK) do MQTT em sua plataforma de construção. Grave um makefile para construir a biblioteca para sua plataforma de destino, incorporando as opções documentadas em “MQTT opções de construção para diferentes plataformas” na página 32. As etapas de plataforma específica para construir e executar um makefile são fornecidas aqui:

- iOS “Construindo as bibliotecas do cliente MQTT para C em um Apple Mac para usar com dispositivos do iOS” na página 34
- Windows “Construindo as bibliotecas do MQTT no Windows” na página 40

Procedimento

1. Instale um ambiente de desenvolvimento C na plataforma na qual estiver construindo.

   Os arquivos de criação nos exemplos neste tópico são voltados para as seguintes ferramentas:

   - 
     Para iOS, no Apple Mac com OS X 10.8.2 com as ferramentas de desenvolvimento iOS de Xcode.
   - 
     Para o Linux, a versão gcc 4.4.6 a partir do Red Hat Enterprise Linux versão 6.2.

     O nível mínimo suportado da biblioteca C `glibc` é 2.12, e o kernel Linux é 2.6.32.
   - 
     Para o Microsoft Windows, Visual Studio versão 10.0.
2. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.
   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.
   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*`\SDK`.
3. Expanda o código-fonte para o cliente MQTT para as bibliotecas C.

   O arquivo compactado do código-fonte é *`sdkroot`*`\SDK\clients\c\source.zip`.
4. Opcional: Construa o OpenSSL.

   Consulte “Construindo o pacote do OpenSSL” na página 45.
5. Construa o cliente MQTT para as bibliotecas C.

   Os comandos e as opções para construir as bibliotecas estão listados em “MQTT opções de construção para diferentes plataformas” na página 32.

   Siga as etapas nos exemplos a seguir para gravar um makefile para construir o cliente MQTT para as bibliotecas C em sua plataforma de destino.

   - “Construindo as bibliotecas do cliente MQTT para C em um Apple Mac para usar com dispositivos do iOS” na página 34
   - “Construindo as bibliotecas do MQTT no Windows” na página 40

MQTT opções de construção para diferentes plataformas

A tabela a seguir lista o compilador e as opções de construção para construir o cliente MQTT para as bibliotecas C em várias plataformas.

Tabela 2. MQTT opções de construção para diferentes plataformas

Plataforma	Compiler	Compiler Options	Linker Options	Extra Options
AIX	gcc	-fPIC -Os -Wall -DREVERSED -I *MQTTCLIENT_DIR*	-Wl,-G	
Linux s390x			-shared -Wl,-soname, libmqttv3c.so	-m 64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m 32
Linux ARM (glibc)	arm-linux-gcc			
Linux ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			
Windows 32-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			

Tabela 2. MQTT opções de construção para diferentes plataformas (continuação)				
Plataforma	**Compiler**	**Compiler Options**	**Linker Options**	**Extra Options**
iOS ARMi386	gcc -arch i386	-DUSE_NAMED_SEMAPHORES -DNOSIGPIPE -DOPENSSL -Os -Wall -fomit-frame-pointer -isysroot / Applications/ Xcode.app/\ Contents/Developer/ Platforms/\ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk	-L/Applications/Xcode.app/ Contents/Developer/ Platforms/ iPhoneSimulator.platform/ Developer/SDKs/ iPhoneSimulator6.0.sdk/usr /lib/system	

iOS *Construindo as bibliotecas do cliente MQTT para C em um Apple Mac para usar com dispositivos do iOS*

Siga estas etapas para gravar um makefile para construir as bibliotecas do cliente MQTT para C em um Apple Mac, para uso subsequente com dispositivos do iOS.

Antes de começar

1. Instale as ferramentas de construção, desenvolva e execute o makefile no Apple Mac com OS X 10.8.2 ou posterior.
2. Instale as ferramentas de linha de comandos para o Xcode, que incluem o programa **make**. Faça download das ferramentas da linha de comandos do Xcode.

Sobre esta tarefa

Crie um makefile que constrói as bibliotecas do cliente MQTT para C para o iPhone ou iPad em execução em um processador ARMv7 ou ARMv7s e o simulador iPhone executado em um processador de bit i386-64. Consulte a Lista de dispositivos iOS.

Sugestão: “Lista de makefile MQTTios.mak” na página 38 lista o makefile concluído.

1. Copie e cole a listagem em um arquivo.
2. Converta o caractere principal de cada linha que segue um destino para uma guia; consulte a etapa “8” na página 36.
3. Execute-o com o comando listado na etapa “9” na página 38 do procedimento.

Procedimento

1. Faça download e instale as ferramentas de desenvolvimento do iOS .
   a. Efetue logon com um ID de usuário que tenha privilégios administrativos.
   b. Verifique se o seu Apple Mac está na versão 10.8.2 ou mais recente.

c. Acesse o website Xcode para fazer download do Xcode na loja de aplicativos Mac.

d. Instale o Xcode, o ambiente da linha de comandos e o simulador.

Se o armazenamento de aplicativo Mac oferecer várias versões do simulador, escolha a versão compatível com o nível do iOS que você está destinando para seu aplicativo.

2. Crie o makefile `MQTTios.mak`

   Inclua um prólogo:

```
# A saída de compilação é produzida no diretório atual
# MQTTCLIENT_DIR deve apontar para o diretório base contendo o código-fonte do cliente MQTT.
# MQTTCLIENT_DIR padrão é o diretório atual
# TOOL_DIR padrão é /Applications/Xcode.app/Contents/Developer/Platforms
# O OPENSSL_DIR padrão é sdkroot/openssl, relativo a sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR deve apontar para o diretório base contendo a construção OpenSSL .
# Exemplo: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. Configure o local do código-fonte MQTT .

   Execute o makefile no mesmo diretório que os arquivos de origem MQTT ou configure o parâmetro da linha de comandos `MQTTCLIENT_DIR` :

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

   Inclua as linhas a seguir no makefile:

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

   O exemplo configura VPATH para o diretório em que **make** procura arquivos de origem que não são explicitamente identificados; por exemplo, todos os arquivos de cabeçalho que são necessários na construção.

4. Opcional: Configure o local das OpenSSL bibliotecas.

   Esta etapa é necessária para construir as versões SSL do cliente MQTT para bibliotecas C.

   Configure o caminho padrão para as bibliotecas do OpenSSL para o mesmo diretório que você expandiu o MQTT SDK. Caso contrário, configure `OPENSSL_DIR` como um parâmetro da linha de comandos.

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

   Sugestão: *OpenSSL* é o diretório OpenSSL que contém todos os subdiretórios OpenSSL .. Pode ser necessário mover a árvore de diretórios a partir de onde você o expandiu porque ele contém diretórios pai vazios desnecessários.

5. Configure os diretórios de ferramentas de desenvolvimento.

   Se você instalou o Xcode em um local diferente, configure `TOOL_DIR` na linha de comandos.

```
ifndef TOOL_DIR
   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. Selecione todos os arquivos de origem necessários para construir cada biblioteca do MQTT . Além disso, configure o nome e o local da biblioteca MQTT que será construída.

   Inclua a linha a seguir no makefile para listar todos os MQTT arquivos de origem:

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

Os arquivos de origem dependem se você está construindo uma biblioteca síncrona ou assíncrona e se a biblioteca inclui SSL ou não.

Inclua uma ou mais dessas linhas, que dependem dos destinos para construir. As bibliotecas compartilhadas são criadas no diretório `darwin_x86_64`.

- Síncrono, descoberto:

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

- Síncrono, protegido:

```
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
```

- Assíncrono, descoberto:

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

- Assíncrono seguro:

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

7. Defina o compilador e as opções do compilador.

   Consulte as opções para diferentes plataformas que são mostradas em Opções de construção MQTT para diferentes plataformas

   a) Configure o projeto C do Gnu e C++ (**gcc**) como o compilador.

   Selecione três compiladores cruzados para construir a biblioteca para dispositivos diferentes e o simulador 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) Inclua as opções do compilador.

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

   c) Inclua os caminhos de inclusão.

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

8. Defina os destinos de construção.

   Sugestão: Cada linha sucessiva que define a implementação de um destino deve iniciar com um caractere de tabulação.

   a) Defina o destino **all**.

   O destino "all" constrói todas as bibliotecas.

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

   Listando-o primeiro, ele é o destino padrão.

a) Construa a biblioteca descoberta, síncrona, `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 $@
```

A instrução `rm *.o` exclui todos os arquivos de objeto criados para cada biblioteca. `lipo` concatena todas as três bibliotecas em um arquivo.

b) Construa a biblioteca descoberta, assíncrona, `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 $@
```

A instrução `rm *.o` exclui todos os arquivos de objeto criados para cada biblioteca. `lipo` concatena todas as três bibliotecas em um arquivo.

c) Construa a biblioteca protegida, síncrona, `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 $@
```

A instrução `rm *.o` exclui todos os arquivos de objeto criados para cada biblioteca. `lipo` concatena todas as três bibliotecas em um arquivo.

d) Construa a biblioteca protegida, assíncrona, `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 $@
```

A instrução `rm *.o` exclui todos os arquivos de objeto criados para cada biblioteca. `lipo` concatena todas as três bibliotecas em um arquivo.

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

O destino "clean" remove todos os arquivos e diretórios gerados pelo makefile

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

9. Execute o makefile.

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

Resultados

Os seguintes arquivos são criados no diretório *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
```

Lista de makefile `MQTTios.mak`

```
# A saída de compilação é produzida no diretório atual
# MQTTCLIENT_DIR deve apontar para o diretório base contendo o código-fonte do cliente MQTT.
# MQTTCLIENT_DIR padrão é o diretório atual
# TOOL_DIR padrão é /Applications/Xcode.app/Contents/Developer/Platforms
# O OPENSSL_DIR padrão é sdkroot/openssl, relativo a sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR deve apontar para o diretório base contendo a construção OpenSSL .
# Exemplo: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

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

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

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

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

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

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

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

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

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

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

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

Windows *Construindo as bibliotecas do MQTT no Windows*

Siga estas etapas para gravar um makefile para construir as bibliotecas do cliente MQTT para C no Windows.

Antes de começar

1. Se necessário, instale uma versão do **Make** em sua estação de trabalho de construção compatível com makefiles gravados para o Gnu make; caso contrário, faça download do Gnu make e construa-o. Consulte Gnu Make. O Web site, Make for Windows, fornece uma versão instalável de **Make** para Windows
2. Também requeira comandos do Linux para Windows para usar o destino clean no exemplo de makefile. É possível obter os comandos do Linux para Windows em websites como Cygwin.

Sobre esta tarefa

Crie um makefile que constrói bibliotecas do cliente MQTT para C para o Windows de 32 bits.

Sugestão: “Lista de makefile MQTTwin.mak” na página 44 lista o makefile concluído.

1. Copie e cole a listagem em um arquivo.
2. Converta o caractere principal de cada linha que segue um destino para uma guia; consulte a etapa “7” na página 42.
3. Execute-o com o comando listado na etapa “9” na página 44 do procedimento.

Procedimento

1. Crie o makefile `MQTTwin.mak`

   Inclua um prólogo:

   ```
   # A saída de compilação é produzida no diretório atual
   # MQTTCLIENT_DIR deve apontar para o diretório base contendo o código-fonte do cliente MQTT.
   # MQTTCLIENT_DIR padrão é o diretório atual
   # OPENSSL_DIR padrão é sdkroot\openSSL, relativo a sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR deve apontar para o diretório base contendo a construção OpenSSL .
   # Exemplo: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # Configure o ambiente de compilação, por exemplo
   # %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
   # configure path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
   ```

2. Configure o local do código-fonte MQTT .

   Execute o makefile no mesmo diretório que os arquivos de origem MQTT ou configure o parâmetro da linha de comandos `MQTTCLIENT_DIR` :

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

   Inclua as linhas a seguir no makefile:

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

O exemplo configura VPATH para o diretório em que **make** procura arquivos de origem que não são explicitamente identificados; por exemplo, todos os arquivos de cabeçalho que são necessários na construção.

3. Opcional: Configure o local das OpenSSL bibliotecas.

   Esta etapa é necessária para construir as versões SSL do cliente MQTT para bibliotecas C.

   Configure o caminho padrão para as bibliotecas do OpenSSL para o mesmo diretório que você expandiu o MQTT SDK. Caso contrário, configure `OPENSSL_DIR` como um parâmetro da linha de comandos.

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

   Sugestão: *OpenSSL* é o diretório OpenSSL que contém todos os subdiretórios OpenSSL .. Pode ser necessário mover a árvore de diretórios a partir de onde você o expandiu porque ele contém diretórios pai vazios desnecessários.

4. Selecione todos os arquivos de origem necessários para construir cada biblioteca do MQTT . Além disso, configure o nome e o local da biblioteca MQTT que será construída.

   Inclua a linha a seguir no makefile para listar todos os MQTT arquivos de origem:

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

   Os arquivos de origem dependem se você está construindo uma biblioteca síncrona ou assíncrona e se a biblioteca inclui SSL ou não.

   Inclua uma ou mais dessas linhas, que dependem dos destinos para construir. As bibliotecas compartilhadas e manifests são criados no diretório `windows_ia32`.

   - Síncrono, descoberto:

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

   - Síncrono, protegido:

     ```
     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
     ```

   - Assíncrono, descoberto:

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

   - Assíncrono seguro:

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

5. Defina o compilador e as opções do compilador.

   Consulte as opções para diferentes plataformas que são mostradas em Opções de construção MQTT para diferentes plataformas

   a) Configure Microsoft Visual C++ como o compilador.

```
CC = cl
```

b) Inclua as opções de pré-processador.

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

c) Inclua as opções do compilador.

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

d) Inclua os caminhos de inclusão.

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

e) Opcional: Inclua uma opção do pré-processador, se você estiver construindo uma biblioteca segura.

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

f) Opcional: Inclua os arquivos de cabeçalho do OpenSSL, se você estiver construindo uma biblioteca segura.

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

Sugestão: Os arquivos de cabeçalho estão em `${OPENSSL_DIR}/inc32/openssl`, mas o arquivo `ssl.h` está incluído com o "`openssl/ssl.h`".

6. Configure o vinculador e as opções do vinculador.

a) Configure Microsoft Visual C++ como a vinculador.

```
LD = link
```

b) Inclua as opções do vinculador.

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

c) Inclua os caminhos da biblioteca.

```
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) Inclua os arquivos de saída intermediários.

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

e) Opcional: Inclua as bibliotecas do OpenSSL, se você estiver construindo uma biblioteca segura.

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

f) Opcional: Inclua o caminho da biblioteca do OpenSSL, se você estiver construindo uma biblioteca segura.

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

7. Defina os quatro destinos de construção.

a) Defina o destino **all**.

Sugestão: Cada linha sucessiva que define a implementação de um destino deve iniciar com um caractere de tabulação.

O destino "all" constrói todas as bibliotecas.

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

b) Construa a biblioteca descoberta, síncrona, `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}
```

A instrução `-rm ${CURDIR}/MQTTAsync.obj` exclui qualquer `MQTTAsync.obj` criado para um destino anterior. `MQTTAsync.obj` e `MQTTClient.obj` são mutuamente exclusivos.

c) Construa a biblioteca descoberta, assíncrona, `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}
```

A instrução `-rm ${CURDIR}/MQTTClient.obj` exclui qualquer `MQTTClient.obj` criado para um destino anterior. `MQTTAsync.obj` e `MQTTClient.obj` são mutuamente exclusivos.

d) Construa a biblioteca protegida, síncrona, `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}
```

A instrução `-rm ${CURDIR}/MQTTAsync.obj` exclui qualquer `MQTTAsync.obj` criado para um destino anterior. `MQTTAsync.obj` e `MQTTClient.obj` são mutuamente exclusivos.

e) Construa a biblioteca protegida, assíncrona, `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}
```

A instrução `-rm $(CURDIR}/MQTTClient.obj` exclui qualquer `MQTTClient.obj` criado para um destino anterior. `MQTTAsync.obj` e `MQTTClient.obj` são mutuamente exclusivos.

f) Defina o destino **clean**.

O destino "clean" remove todos os arquivos e diretórios gerados pelo makefile

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

8. Configure o caminho do Windows para executar o makefile.

Configure as partes em itálico para corresponder com sua instalação.

a) Configure o ambiente do Microsoft Visual Studio.

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

b) Configure as variáveis `Path` para incluir o programa make e o ambiente de comando do Linux.

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

9. Execute o makefile.

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

Sugestão: O caractere separador de arquivo deve ser uma barra, não uma barra invertida.

Resultados

Os seguintes arquivos são criados no diretório *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
```

Lista de makefile `MQTTwin.mak`

```
# A saída de compilação é produzida no diretório atual
# MQTTCLIENT_DIR deve apontar para o diretório base contendo o código-fonte do cliente MQTT.
# MQTTCLIENT_DIR padrão é o diretório atual
# OPENSSL_DIR padrão é sdkroot\openSSL, relativo a sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR deve apontar para o diretório base contendo a construção OpenSSL .
# Exemplo: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Configure o ambiente de compilação, por exemplo
# %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
# configure path= %path%; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/ ../../../../../openssl-1.0.1c
endif

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

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

# compiler
CC = cl
CPPFLAGS = /D "WIN32" /D "_UNICODE" /D "UNICODE" /D "_CRT_SECURE_NO_WARNINGS"
CFLAGS = /nologo /c /O2 /W3 /Fd /MD /TC
```

```
INC =  /I ${MQTTCLIENT_DIR} /I ${MQTTCLIENT_DIR}/..
CPPFLAGS_S = ${CPPFLAGS} /D "OPENSSL"
INC_S = ${INC} /I ${OPENSSL_DIR}/inc32/

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

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

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

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

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

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

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

Construindo o pacote do OpenSSL

Construa o pacote do OpenSSL antes de construir as bibliotecas seguras do cliente MQTT para C, `mqttv3cs` e `mqttv3as`. A construção cria as bibliotecas necessárias para construir uma versão segura do cliente MQTT para a biblioteca C e a ferramenta de gerenciamento de certificado do OpenSSL.

Antes de começar

1. iOS A customização de makefile do iOS é para os dispositivos de destino que executam o iOS6. A customização pode ser diferente para versões anteriores ou posteriores do iOS.
2. Windows A customização de makefile do Windows é para janelas de 32 bits.

Sobre esta tarefa

Faça download e instale o pacote do OpenSSL e qualquer software obrigatório. Customize os makefiles do OpenSSL e construa as bibliotecas do OpenSSL para sua plataforma de destino. No Windows e Linux, make também constrói a criação de chaves e ferramenta de gerenciamento do OpenSSL.

Procedimento

1. Instale o pacote do OpenSSL.
   a) Faça o download do pacote OpenSSL a partir do OpenSSL

      Importante: O download e a redistribuição do pacote OpenSSL estão sujeitos a regulamentações rigorosas de importação e exportação e condições de licenciamento de software livre. Tome cuidadosa nota das restrições e avisos antes de decidir fazer download do pacote.

   b) Expanda o conteúdo do arquivo compactado em *sdkroot*.

      Procure na guia **Notícias** no site OpenSSL para localizar o local de download do pacote mais recente. O pacote é compactado como um arquivo tar com a extensão `tar.gz`. Quando expandido, o pacote criará uma pasta de nível superior `openssl`*version*; por exemplo *openssl-1.0.1c*. Os exemplos se referem ao caminho para a pasta como %openssl% no Windows e $openssl no iOS; por exemplo, no Windows, %openssl% é *sdkroot*\openssl-1.0.1c.

      Sugestão: Verifique o caminho de diretório criado extraindo o pacote do OpenSSL. Alguns pacotes têm níveis duplicados da pasta `openssl`*version*.

2. Windows

   Opcional: No Windows, faça download e instale o perl. Consulte perl.org.

   Para o exemplo, perl foi transferido por download a partir do Downloads do ActivePerl.

3. iOS

   Opcional: No iOS, crie mais três diretórios.

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

   Para o iOS, deve-se construir o pacote do OpenSSL para três plataformas de hardware diferentes.

4. Gere o makefile do OpenSSL para construir o pacote do OpenSSL para seu hardware e sistema operacional.
   a) Abra uma janela de comando no diretório %openssl% ou $openssl.
   b) Execute o comando perl **Configure** com os parâmetros apropriados.

   - 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

   No iOS, customize o makefile do OpenSSL gerado para diferentes dispositivos do Apple.
   a) Faça três cópias do makefile gerado, $openssl/Makefile

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

   b) Mude a instrução "CC=gcc" em cada makefile.

      A instrução CC=gcc está na linha 62 ou aproximadamente. Mude-a para os seguintes comandos:

      $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) Mude a instrução "CFLAG=..." em cada makefile.

A instrução está na linha 63 ou aproximadamente (dividida em três linhas para capacidade de leitura):

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

O local dos SDKs é mostrado depende de suas opções de instalação do Xcode. A versão dos SDKs depende do nível do sistema operacional ao qual você está construindo o makefile.

Simulador do iPhone

O makefile de simulador do iPhone é `$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

Os makefiles do iOS são `$openssl/Makefile_arm7` e `$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. Execute o makefile gerado.

- Windows

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

- iOS No iOS:

```
make clean
make -f $openssl/Makefile_arm7
mv $openssl/libcrypto.a $ssarm7/libcrypto.a
mv $openssl/libssl.a $ssarm7/libssl.a
make clean
make -f $openssl/Makefile_arm7s
mv $openssl/libcrypto.a $sslarm7s/libcrypto.a
mv $openssl/libssl.a $sslarm7s/libssl.a
make clean
make -f $openssl/Makefile_i386
mv $openssl/libcrypto.a $ssli386/libcrypto.a
mv $openssl/libssl.a $ssli386/libssl.a
```

Resultados

A construção gera as bibliotecas compartilhadas, lib e arquivos de cabeçalho necessários para construir versões segura da biblioteca do cliente MQTT para C.

Introdução ao cliente MQTT para C no iOS

Saiba como obter os aplicativos iOS para trocar mensagens com um servidor MQTT. Para uso em dispositivos do iOS (ou seja, iPhone e iPad), deve-se criar a biblioteca do cliente MQTT para C a partir do código-fonte fornecido como parte do kit de desenvolvimento de software do MQTT.

Antes de começar

1. Vincule ao iOS Dev Center e saiba como desenvolver aplicativos para o iOS.
2. Obtenha um Apple Mac com OS X 10.8.2 ou posterior para executar o ambiente de desenvolvimento integrado (IDE) do Xcode.
3. (Opcional) Configure um ambiente de desenvolvimento C no Windows ou Linux. Será útil para construir e executar os aplicativos C de amostra do cliente MQTT no Windows ou Linux antes de desenvolver um aplicativo MQTT iOS. Como alternativa, analise o código-fonte de amostra sem construir as amostras.
4. Para as plataformas do cliente MQTT com suporte e referência para C, consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.
5. Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.

Sobre esta tarefa

O procedimento orienta você através das seguintes etapas:

1. Aprenda sobre programação para o MQTT analisando, construindo e executando os aplicativos de amostra do cliente MQTT e as bibliotecas do cliente MQTT para C.
2. Instale o ambiente de desenvolvimento do Xcode para o iOS no Apple Mac.
3. Execute a tarefa “Construindo o cliente MQTT para as bibliotecas C” na página 31 para construir o cliente MQTT para as bibliotecas C para dispositivos do iOS.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 . Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte “Introdução aos servidores MQTT” na página 138.
2. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.
   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.
   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*`\SDK`.
3. Opcional: Familiarize-se com a API do MQTT analisando o Aplicativo C de amostra do cliente do MQTT.

   a) Construa o aplicativo C de amostra do cliente MQTT síncrono `MQTTV3sample.c` para o Windows ou Linux. Consulte o “Introdução ao cliente MQTT para C” na página 27.

   b) Conecte-se a um servidor MQTT version 3, publique e assine os tópicos no servidor.

   c) Estude o código-fonte e a documentação da API do MQTT Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

      Aprenda a criar e retomar os clientes MQTT e publicar e assinar os tópicos do MQTT, analisando a amostra síncrona. A amostra síncrona é mais simples do que a amostra assíncrona. Se você não

tiver programado o MQTT antes, grave um programa síncrono MQTT para se familiarizar com o modelo de programação e a API do MQTT.

d) Construa a biblioteca do cliente assíncrono MQTT para C no Windows ou Linux. Consulte o “Construindo o cliente MQTT para as bibliotecas C” na página 31.

e) Construa e execute o aplicativo C de publicação e assinatura da amostra do cliente assíncrono MQTT.

f) Analise o código-fonte do aplicativo C assíncrono com a amostra do cliente MQTT e a documentação de referência do MQTT.

Deve-se usar a interface assíncrona para gravar um aplicativo MQTT para um dispositivo móvel. Aplicativos bem-gravados que chamam a interface assíncrona são mais responsivos e estenderá a vida útil da bateria além de aplicativos gravados na interface síncrona.

A interface assíncrona possui dois graus de assincronicidade:

i) O primeiro grau é para desbloquear o aplicativo enquanto a biblioteca do cliente MQTT espera por publicações do servidor.

ii) O segundo grau é para desbloquear o aplicativo enquanto a biblioteca do cliente se conectar ao servidor, criar assinaturas e postar publicações.

4. Faça download e instale as ferramentas de desenvolvimento do iOS .
   a. Efetue logon com um ID de usuário que tenha privilégios administrativos.
   b. Verifique se o seu Apple Mac está na versão 10.8.2 ou mais recente.
   c. Acesse o website Xcode para fazer download do Xcode na loja de aplicativos Mac.
   d. Instale o Xcode, o ambiente da linha de comandos e o simulador.

   Se o armazenamento de aplicativo Mac oferecer várias versões do simulador, escolha a versão compatível com o nível do iOS que você está destinando para seu aplicativo.
5. Construa as bibliotecas do cliente MQTT para C no iOS. Consulte “Construindo o cliente MQTT para as bibliotecas C” na página 31.

Como proceder a seguir

1. Verifique se as bibliotecas do cliente MQTT para C que você construiu:
   a. Use o ambiente de desenvolvimento do Xcode para compilar o Aplicativo C de amostra do cliente do MQTT assíncrono e o link para a biblioteca do cliente MQTT descoberta assíncrona para C.
   b. Use o ambiente de desenvolvimento do Xcode para executar o aplicativo C de amostra do cliente MQTT assíncrono em um dispositivo iOS. Conecte a amostra para o servidor MQTT version 3 configurado; consulte “Configurando o serviço MQTT a partir da linha de comandos” na página 142.
2. Crie um aplicativo C do cliente MQTT para o iOS. Os exemplos de codificação para o aplicativo C assíncrono podem ser úteis. Os exemplos são `MQTTV3ASample.c` e `MQTTV3ASSample.c` em *`sdkroot`*`\SDK\clients\c\samples` Como um exercício, inicie implementando a amostra de publicação/assinatura do MQTT.

   Sugestão: Para ter uma ideia do que o aplicativo pode fazer e como ele pode ser, examine a captura de tela do Aplicativo C de amostra do cliente do MQTT. Consulte “Introdução ao Cliente de MQTT para Java no Android” na página 18.

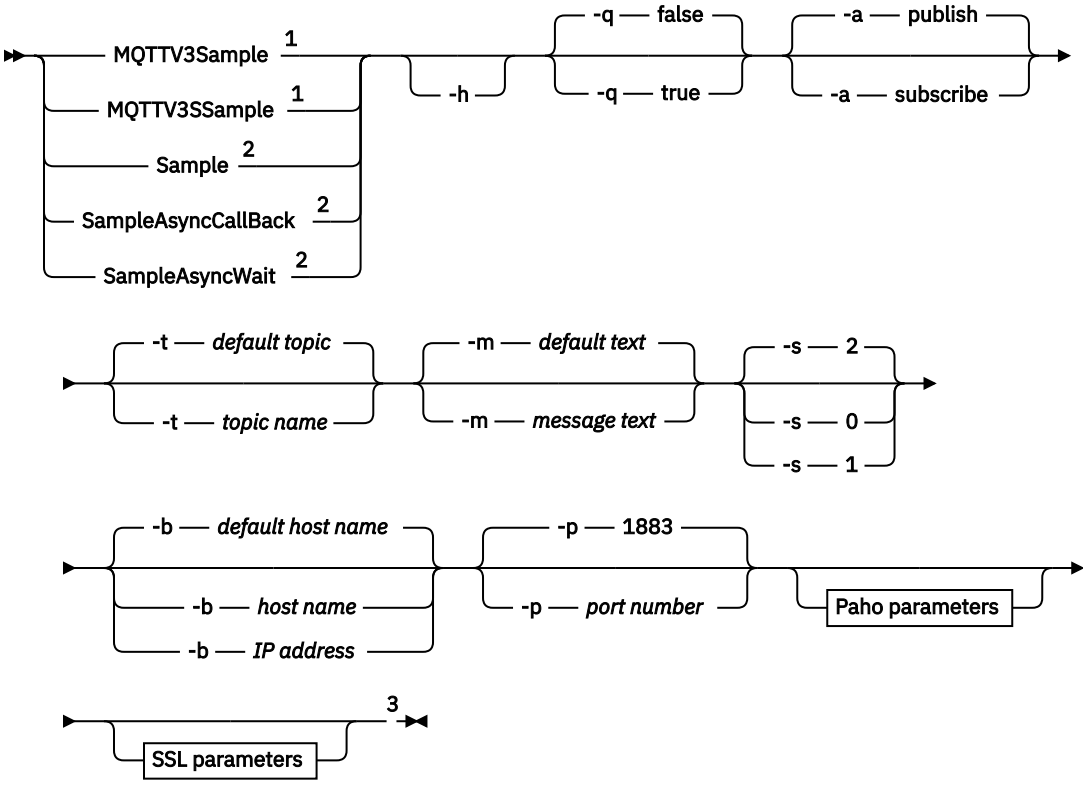
Programas de amostra da linha de comandos doMQTT

A sintaxe e os parâmetros dos programas de amostra da linha de comandos do MQTT.

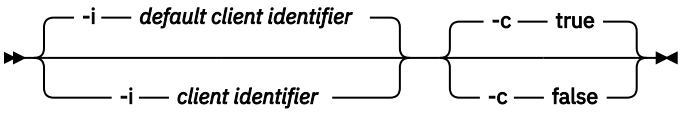
Finalidade

Publicar e assinar um tópico.

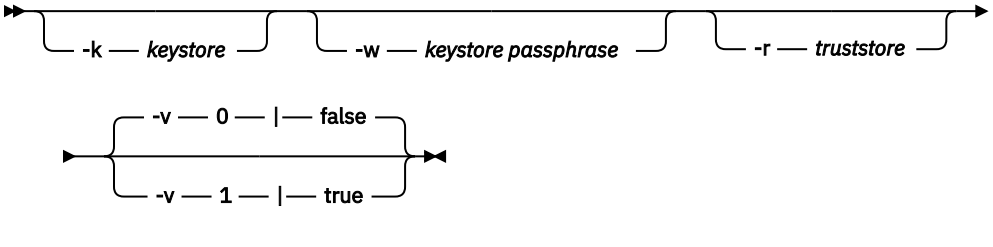
Syntax



Paho parameters



SSL parameters



Notas:

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

Parâmetros

`-h`
Imprimir este texto de ajuda e sair

`-q`
Configure o modo silencioso em vez de usar o modo false padrão.

`-a publish|subscribe`
Configure a ação para `publish` ou `subscribe`, em vez de assumir a ação padrão de publicação.

-t ***topic name***
Publique ou assine para *topic name*, em vez da publicação ou assinatura para o tópico padrão. Os tópicos padrão são os seguintes:

Amostras do Paho

Publicar
`Sample/Java/v3`

Assinar
`Sample/#`

Amostras do IBM WebSphere MQ

Publicar
`MQTTV3Sample/Java/v3` ou `MQTTV3Sample/C/v3`

Assinar
`MQTTV3Sample/#`

-m ***message text***
Publique *message text* em vez de enviar o texto padrão. O texto padrão é "`Message from MQTTv3 C client`" ou "`Message from MQTTv3 Java client`"

-s 0|1|2
Configure a qualidade de serviço (QoS) em vez de usar a QoS padrão 2.

-b ***host name***
Conecte para *host name* ou endereço IP em vez da conexão com o nome do host padrão. O nome do host padrão para as amostras do Paho é `m2m.eclipse.org`. Para as amostras do IBM WebSphere MQ é `localhost`.

-p ***port number***
Use a porta *port number* em vez de usar a porta padrão, 1883.

Parâmetros do Paho

-i ***client identifier***
Configure o identificador de cliente como *client identifier.* O identificador de cliente padrão é `SampleJavaV3_"+action`e `action` é `publish` ou `subscribe`.

-c true|false
Configure o sinalizador de sessão limpo. O padrão é `true`: as assinaturas não são duráveis.

Parâmetros de SSL

-k ***keystore***
Configure o caminho para o keystore que contém a chave privada que identifica o cliente para *keystore*.Para as amostras C, o armazenamento é um arquivo Privacy-Enhanced Mail (PEM). Para as amostras Java é um Java keystore (JKS).

-w ***keystore passphrase***
Configure o passphrase para autorizar o cliente a acessar o keystore para *keystore passphrase*.

-r ***truststore***
Configure o caminho para o keystore que contém as chaves públicas dos servidores MQTT que o cliente confia para *truststore*. O keystore é um arquivo Privacy-Enhanced Mail (PEM). Para as amostras C, o armazenamento é um arquivo Privacy-Enhanced Mail (PEM). Para as amostras Java é um Java keystore (JKS).

-v 0|false|1|true
Configure a opção de verificação para `1|true` para requerer um certificado do servidor. O padrão é `0|false`: o certificado do servidor não é verificado O canal SSL é sempre criptografado.

Configure a opção para `0|1` para programas C e `true|false` para programas Java.

Tarefas relacionadas
“Introdução ao cliente MQTT para o Java” na página 12

Esteja em funcionamento com o cliente MQTT para Java aplicativos de amostra, usando IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor. Os aplicativos de amostra usam uma biblioteca do cliente a partir do kit de desenvolvimento de software (SDK) do MQTT da IBM. O aplicativo de amostra `SampleAsyncCallBack` é um modelo para gravar aplicativos MQTT para Android e outros sistemas operacionais orientados a eventos.

"Introdução ao cliente MQTT para C" na página 27
Funcionamento adequado com o cliente MQTT de amostra para C em qualquer plataforma na qual é possível compilar a origem C. Verifique se é possível executar o cliente MQTT de amostra para C com o IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

"Construindo o cliente MQTT para as bibliotecas C" na página 31
Siga estas etapas para construir o cliente MQTT para as bibliotecas C. O tópico inclui os comutadores de compilação e de link para um número de plataformas e exemplos de construção das bibliotecas no iOS e Windows.

Segurança do MQTT

Três conceitos são fundamentais para a segurança do MQTT: identidade, autenticação e autorização. Identidade é como denominar o cliente que está sendo autorizado e a autoridade fornecida. A autenticação é a comprovação da identidade do cliente e a autorização é como gerenciar os direitos fornecidos com o cliente.

Tente as amostras de segurança

- "Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro" na página 55
- "Conectando o aplicativo Java de amostra do cliente MQTT no Android sobre SSL" na página 64
- "Autenticando um app Java do cliente MQTT com o JAAS" na página 73
- V 7.5.0.1 "Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets" na página 78
- "Construindo e executando o Aplicativo C de amostra do cliente do MQTT seguro" na página 86

Identidade

Identifique um cliente MQTT por seu identificador de cliente, ID do usuário ou certificado digital público. Um ou outro destes atributos define a identidade do cliente. Um servidor MQTT autentica o certificado enviado pelo cliente com o protocolo SSL ou a identidade do cliente com uma senha configurada pelo cliente. O servidor controla quais recursos o cliente pode acessar, com base na identidade do cliente.

O servidor MQTT se identifica para o cliente com seu endereço IP e seu certificado digital. O cliente MQTT usa o protocolo SSL para autenticar o certificado enviado pelo servidor. Em alguns casos, ele usa o nome DNS do servidor para verificar o servidor que enviou o certificado é registrado como o portador do certificado.

Configure a identidade do cliente de uma das seguintes maneiras:

Identificador de cliente
A classe `MqttClient` (`MQTTClient_create` ou `MQTTAsync_create` em C) configura o identificador de cliente. Chame o construtor da classe para configurar o identificador do cliente como um parâmetro ou retornar um identificador de cliente gerado aleatoriamente.O identificador de cliente deve ser exclusivo em todos os clientes que se conectam ao servidor e não deve ser o mesmo que o nome do gerenciador de filas no servidor. Todos os clientes devem ter um identificador de cliente mesmo se ele não for usado para verificação de identidade. Consulte o "Identificador de Cliente" na página 129.

ID do usuário
A classe `MqttClient` (`MQTTClient_create` ou `MQTTAsync_create` em C) configura o ID do usuário cliente como um atributo de `MqttConnectOptions` (`MqttClient_ConnectOptions` em C). O ID do usuário não precisa ser exclusivo para um cliente.

Certificado digital do cliente

O certificado digital do cliente é armazenado no keystore do cliente. O local do keystore depende do cliente:

- **Java**

  Configure o local e as propriedades de keystore do cliente chamando o método `setSSLProperties` de `MqttConnectOptions` e transmitindo as propriedades do keystore. Consulte Modificações SSL para Example.java. A ferramenta **keytool** gerencia chaves Java e keystores.

- **C**

  `MQTTClient_create` ou `MQTTAsync_create` configura as propriedades de keystore como atributos de `MQTTClient_SSLOptions ssl_opts`. A ferramenta **openSSL** cria e gerencia as chaves e keystores acessados pelo cliente MQTT para C.

- **Android**

  Gerencie um keystore do dispositivo Android no menu **Configurações** > **Segurança**. Carregue novos certificados a partir do cartão SD.

Configure a identidade do servidor armazenando sua chave privada no keystore do servidor:

IBM WebSphere MQ

O keystore do servidor MQTT é um atributo do canal de telemetria ao qual o cliente está conectado.

Configure o local do keystore e os atributos com IBM WebSphere MQ Explorerou com o comando **DEFINE CHANNEL** ; consulte DEFINE CHANNEL (MQTT). Diversos canais podem compartilhar um keystore.

Autenticação

Um cliente MQTT pode autenticar o servidor MQTT ao qual ele se conecta e o servidor pode autenticar o cliente que está se conectando a ele.

Um cliente autentica um servidor com o protocolo SSL. Um servidor MQTT autentica um cliente com o protocolo SSL, ou com uma senha, ou ambos.

Se o cliente autenticar o servidor, mas o servidor não autenticar o cliente, o cliente geralmente será conhecido como um cliente anônimo. É comum estabelecer uma conexão do cliente anônimo através do SSL e, em seguida, autenticar o cliente com uma senha criptografada pela sessão SSL. É muito mais comum autenticar um cliente com uma senha do que com um certificado de cliente, devido à distribuição de certificado e problemas de gerenciamento. Provavelmente você localiza certificados do cliente usados em dispositivos de valor alto, como ATMs e máuinas de cartão e em dispositivos customizados, como medidor de eletricidade inteligente.

Autenticação do servidor por um cliente

Um cliente MQTT verifica se ele está conectado ao servidor correto autenticando o certificado do servidor com o protocolo SSL. Esta forma de verificação será familiar para você, ao procurar um website através do protocolo HTTPS.

O servidor envia seu certificado público, assinado por uma autoridade de certificação, para o cliente. O cliente usa a chave pública da autoridade de certificação para verificar a assinatura da autoridade de certificação no certificado do servidor. Ele também verifica se o certificado é atual. Essas verificações estabelecem que o certificado é válido.

Os certificados da autoridade de certificação, frequentemente denominado certificados raiz, são armazenados no armazenamento confiável do cliente:

- **Java**

  Chame o método `setSSLProperties` de `MqttConnectOptions` e transmita as propriedades do armazenamento confiável para configurar o local e as propriedades do armazenamento confiável do cliente. Consulte Modificações SSL para Example.java. Gerencie certificados e armazenamentos confiáveis com a ferramenta **keytool**.

- **C**

  `MQTTClient_create` ou `MQTTAsync_create` configura as propriedades do armazenamento confiável como atributos de `MQTTClient_SSLOptions ssl_opts`. Gerencie certificados e armazenamentos confiáveis com a ferramenta **openSSL**.

- **Android**

  Gerencie um armazenamento confiável do dispositivo Android no menu **Configurações** > **Segurança**. Carregue novos certificados raiz no cartão SD.

Autenticação do cliente por um servidor

Um servidor MQTT verifica se ele está conectado ao cliente correto autenticando o certificado de cliente com o protocolo SSL ou autenticando a identidade do cliente com uma senha.

Ele autentica o cliente com a senha enviada pelo cliente para o servidor em um cabeçalho do MQTT protocol. O servidor pode escolher autenticar o identificador de cliente, o ID do usuário ou o certificado com a senha. Ele depende do servidor. Geralmente, o servidor autentica o ID do usuário. Verifique as senhas através de uma conexão SSL que foi protegida, verificando o servidor, para evitar o envio de senhas na limpeza.

- **IBM WebSphere MQ**

  O IBM WebSphere MQ autentica um certificado de cliente com o protocolo SSL. Armazene os certificados no keystore do IBM WebSphere MQ Telemetry. É possível apenas autenticar um certificado de cliente como parte da autenticação SSL mútua. Ou seja, deve-se fornecer ao cliente com o certificado público do servidor, bem como fornecer o servidor com o certificado público do cliente.

  O IBM WebSphere MQ Telemetry usa o mesmo armazenamento para seu próprio certificado privado e público e outros certificados públicos, como os certificados raiz fornecidos pelas autoridades de certificação.

  Configure o local do keystore e os atributos com IBM WebSphere MQ Explorerou com o comando **`DEFINE CHANNEL`** ; consulte DEFINE CHANNEL (MQTT). Diversos canais podem compartilhar um keystore.

  O IBM WebSphere MQ autentica o ID de usuário cliente ou o identificador de cliente chamando o serviço de autorização e autenticação do Java (JAAS).

  Configure o JAAS em uma sub-rotina de configuração `MQXRConfig` armazenada no arquivo `jaas.config`. O arquivo é armazenado no diretório `qmgrs\`*`QmgrName`*`\mqxr` no caminho de dados do IBM WebSphere MQ.

  Verifique a autenticidade do cliente gravando um método `login` para o `JAASLoginModule`. Consulte “Configuração JAAS do Canal de Telemetria” na página 115.

  IBM WebSphere MQ Telemetry transmite o método `JAASLoginModule.login` para os seguintes parâmetros:

  – ID do usuário
  – Senha
  – Identificador de Cliente
  – Identificador da rede
  – Nome do canal
  – ValidPrompts

Authorization

A autorização não faz parte do MQTT protocol. Ela é fornecida pelos servidores MQTT. O que está autorizado depende do que o servidor faz. Os servidores MQTT são brokers de publicação/assinatura e regras uteis de autorização do MQTT que controlam quais clientes podem se conectar ao servidor e quais

tópicos um cliente pode publicar ou assinar. Se um cliente MQTT pode administrar o servidor, mais regras de autorização controlarão quais clientes podem administrar aspectos diferentes do servidor.

O número de clientes possíveis é enorme, portanto, não é viável autorizar cada cliente separadamente. Um servidor MQTT terá um meio de agrupar clientes por perfis ou grupos.

A identidade de um cliente, do ponto de vista de acesso e autorização, não é algo exclusivo para um cliente MQTT. Não compare a identidade de um cliente com o identificador de cliente. Eles podem ser as mesmos, mas normalmente são diferentes. Por exemplo, você provavelmente terá um nome de usuário comum em vários serviços e alguns desses serviços cooperam com a "conexão única". Um servidor MQTT de escala corporativa é provável que chame um serviço de autorização que ofereça as identidades e as autoridades comuns para diferentes aplicativos.

IBM WebSphere MQ

O IBM WebSphere MQ tem um serviço de autorização que pode ser conectado. O serviço de autorização padrão fornecido em Windows e Linux é o gerenciador de autoridade de objeto (OAM). Consulte Controlando o Acesso aos Objetos Usando o OAM nos Sistemas UNIX, Linux e Windows Ele associa os grupos e IDs de usuário do sistema operacional às operações nos objetos do IBM WebSphere MQ, como tópicos e filas.

É possível configurar um canal de telemetria para acessar o IBM WebSphere MQ com um ID do usuário fixo. Esta é a maneira como o canal de amostra está configurado. Ou é possível acessar o IBM WebSphere MQ com o ID do usuário configurado pelo cliente MQTT. A autorização de clientes MQTT para acessar objetos do WebSphere MQ descreve maneiras de configurar o IBM WebSphere MQ Telemetry para atingir o controle de acesso do cliente de baixa, média e baixa granularidade.

Tarefas relacionadas

“Construindo e executando o Aplicativo C de amostra do cliente do MQTT seguro” na página 86
Com base em um exemplo do Windows, é possível o funcionamento adequado com o aplicativo C seguro de amostra em qualquer sistema operacional ao qual é possível compilar a origem C. Verifique se é possível executar o aplicativo C de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

“Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro” na página 55
Com base em um exemplo do Windows, é possível funcionar adequadamente com o aplicativo seguro Java de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível"

“Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets” na página 78
Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

“Conectando o aplicativo Java de amostra do cliente MQTT no Android sobre SSL” na página 64
Funcionando adequadamente com o cliente Android MQTT de amostra conectado ao IBM WebSphere MQ através do SSL.

“Autenticando um app Java do cliente MQTT com o JAAS” na página 73
Aprenda a autenticar um cliente com JAAS. Conclua as etapas nesta tarefa para modificar o programa de amostra `JAASLoginModule.java` e configurar IBM WebSphere MQ para autenticar um MQTT app Java do cliente com JAAS.

Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro

Com base em um exemplo do Windows, é possível funcionar adequadamente com o aplicativo seguro Java de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível"

Antes de começar

1. Deve-se ter acesso a um servidor MQTT version 3.1 que suporte MQTT protocol sobre SSL.
2. Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.
3. É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível". Consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.
4. Os canais de SSL devem ser iniciados.

Sobre esta tarefa

Como uma ilustração, este artigo mostra como compilar e executar o Aplicativo de amostra do cliente MQTT Java seguro no Windows a partir da linha de comandos.

Proteja o canal SSL com chaves assinadas de autoridade de certificado ou chaves auto-assinadas.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 sobre SSL. Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte “Introdução aos servidores MQTT” na página 138.
2. Opcional: Instale um Java kit de desenvolvimento (JDK) na Versão 7 ou posterior.

   Versão 7 é necessário para executar o comando **keytool** para certificar certificados. Se você não irá certificar certificados, o JDK versão 7 não é necessário.
3. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.

   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta MQTT. O caminho para esta pasta é referido aqui como *sdkroot*.

   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *sdkroot*. A expansão cria uma árvore de diretórios que inicia em *sdkroot*\SDK.
4. Crie e execute os scripts para gerar pares de chaves e certificados e configure IBM WebSphere MQ como o servidor MQTT .

   Siga as etapas em “Gerando Chaves e Certificados” na página 96 para criar e executar os scripts. Os scripts também são listados em “Exemplo de scripts para configurar certificados SSL para Windows” na página 58.
5. Verifique se os canais SSL estão executando e se estão configurados como você espera.

   No IBM WebSphere MQ, digite o seguinte comando em uma janela de 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. Crie os scripts para construir e executar o Aplicativo de amostra do cliente MQTT Java seguro.

a) Crie e execute `ssjavaclient.bat` para testar um canal SSL protegido com certificados autoassinados.

b) Crie e execute `cajavaclient.bat` para testar um canal SSL protegido com certificados assinados com autoridade de certificação.

Scripts para executar o cliente MQTT Java seguro

Execute os scripts em "Exemplo de scripts para configurar certificados SSL para Windows" na página 58 antes de executar esses scripts.

cliente MQTT seguro Java com certificados autoassinados.

Execute este script com os certificados autoassinados criados executando o 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`

Execute o cliente MQTT Java seguro com certificados assinados de autoridade de certificado.

Execute este script com os certificados assinados com autoridade de certificação criados executando o 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`

Conceitos relacionados

“Segurança do MQTT” na página 52
Três conceitos são fundamentais para a segurança do MQTT: identidade, autenticação e autorização. Identidade é como denominar o cliente que está sendo autorizado e a autoridade fornecida. A autenticação é a comprovação da identidade do cliente e a autorização é como gerenciar os direitos fornecidos com o cliente.

Tarefas relacionadas

“Gerando Chaves e Certificados” na página 96
Siga este procedimento para gerar as chaves e os certificados para o Java e os clientes C, incluindo os aplicativos Android e iOS e os servidores IBM WebSphere MQ e IBM MessageSight.

“Conectando o aplicativo Java de amostra do cliente MQTT no Android sobre SSL” na página 64
Funcionando adequadamente com o cliente Android MQTT de amostra conectado ao IBM WebSphere MQ através do SSL.

“Autenticando um app Java do cliente MQTT com o JAAS” na página 73
Aprenda a autenticar um cliente com JAAS. Conclua as etapas nesta tarefa para modificar o programa de amostra `JAASLoginModule.java` e configurar IBM WebSphere MQ para autenticar um MQTT app Java do cliente com JAAS.

Exemplo de scripts para configurar certificados SSL para Windows

** **

Os arquivos de comandos de exemplo criam os certificados e os armazenamentos de certificados, conforme descrito nas etapas da tarefa. Além disso, o exemplo configura o cliente do gerenciador de filas do MQTT para usar o armazenamento de certificados do servidor. O exemplo exclui e recria o gerenciador de filas chamando o script `SampleMQM.bat`, fornecido com o IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` configura os nomes e caminhos para os certificados e outros parâmetros requeridos pelos comandos **keytool** e **openSSL**. As configurações são descritas nos comentários no 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

Os comandos no script `cleancert.bat` excluem o gerenciador de filas do cliente MQTT para assegurar que o armazenamento do certificado do servidor não está bloqueado, e, em seguida, excluem todos os keystores e certificados criados pelos scripts de segurança de amostra.

```
@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

Os comandos no script `genkeys.bat` criam pares de chave para sua autoridade de certificação privada, o servidor, e um cliente.

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

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

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

sscerts.bat

Os comandos no script `sscerts.bat` exportam os certificados autoassinados de cliente e servidor a partir de seus keystores e importam o certificado de servidor no armazenamento confiável do cliente e o certificado de cliente no keystore do servidor. O servidor não tem um armazenamento confiável. Os comandos criam um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

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

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

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

```
authentication)
@rem Omit this step, unless you are authenticating clients.
%keytool% -import -noprompt -alias %cltalias% -file %cltcertselfsigned% -keystore
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

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

cacerts.bat

O script importa o certificado raiz da autoridade de certificação nos keystores privados. O certificado raiz da CA é necessário para criar o keychain entre o certificado raiz e o certificado assinado. O script `cacerts.bat` exporta as solicitações de certificado de cliente e do servidor a partir de seus keystores. O script assina a solicitação de certificado com a chave da autoridade de certificação privada no keystore `cajkskeystore.jkse`, em seguida, importar os certificados assinados de volta para o mesmo keystores a partir do qual o a solicitação chegou. A importação cria a cadeia de certificados com o certificado raiz da CA. O script cria um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

```
@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

O script lista os keystores e certificados no diretório do certificado. Em seguida, cria o gerenciador de filas MQTT de amostra e configura os canais de telemetria seguros.

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

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

```
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
```
V 7.5.0.1
```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Conectando o aplicativo Java de amostra do cliente MQTT no Android sobre SSL

Funcionando adequadamente com o cliente Android MQTT de amostra conectado ao IBM WebSphere MQ através do SSL.

Antes de começar

Este artigo supõe que você esteja executando pelo menos o nível 14 da API do Android (ICS 4.0). Os níveis anteriores tinham um armazenamento de chaves, mas apenas aplicativos do sistema puderam acessá-lo.

1. Deve-se ter acesso a um servidor MQTT version 3.1 que suporte MQTT protocol sobre SSL.
2. Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.
3. Se você estiver testando a conexão em um dispositivo Android inicial, poderá ser necessário um cartão SD para transferir o certificado para o dispositivo.
4. Se você estiver testando a conexão em um dispositivo virtual Android, configure um cartão SD para o dispositivo virtual.
5. Os canais de SSL devem ser iniciados.

Sobre esta tarefa

Conclua esta tarefa para executar o Aplicativo de amostra do cliente MQTT Java para Android através do SSL. Uma conexão SSL bem-sucedida estabelece um canal criptografado seguro entre seu dispositivo Android e o servidor MQTT. A identidade do servidor é autenticada.

Com o Android, é possível autenticar o servidor com SSL. Também é possível autenticar o dispositivo, embora o aplicativo de exemplo não suporte isto. Para autenticar o dispositivo, use a API de keychain ou use o JAAS para autenticar o identificador de cliente, o endereço IP do cliente ou o nome do usuário e a senha fornecida pelo aplicativo MQTT Android.

Quaisquer certificados X.509 instalados no armazenamento confiável do Android devem ser assinados por uma autoridade de certificação. No exemplo, crie uma autoridade de certificação, que assina o certificado instalado em seu dispositivo Android. Inúmeros certificados raiz são pré-instalados nos dispositivos Android.

Deve-se criar um bloqueio em seu dispositivo Android antes de instalar um certificado confiável. O bloqueio evita que alguém instale certificados no dispositivo sem o seu conhecimento.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 sobre SSL. Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte “Introdução aos servidores MQTT” na página 138.
2. Execute o aplicativo de amostra do cliente MQTT for Android "`MQTTExerciser`" em um canal descoberto do MQTT. Consulte “Introdução ao Cliente de MQTT para Java no Android” na página 18.

Use o aplicativo novamente para testar o canal seguro.

Se você iniciou um dispositivo virtual Android, deixe-o em execução.

3. Opcional: Instale um Java kit de desenvolvimento (JDK) na Versão 7 ou posterior.

   Versão 7 é necessário para executar o comando **keytool** para certificar certificados. Se você não irá certificar certificados, o JDK versão 7 não é necessário.

4. Crie e execute os scripts para gerar pares de chaves e certificados e configure IBM WebSphere MQ como o servidor MQTT .

   Siga as etapas em "Gerando Chaves e Certificados" na página 96 para criar e executar os scripts. Os scripts também são listados em "Exemplo de scripts para configurar certificados SSL para Windows" na página 68.

   É necessário o certificado público de autoridade de certificação e o armazenamento de chaves do servidor. Não é necessário os certificados de cliente ou quaisquer certificados no formato `.pem` ou `.p12`.

5. Verifique se os canais SSL estão executando e se estão configurados como você espera.

   No IBM WebSphere MQ, digite o seguinte comando em uma janela de comando:

   - 


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

   - 


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

6. Instale o certificado com autoridade de certificação no armazenamento confiável do Android.

   O arquivo de autoridade de certificação no exemplo é `cacert.cer`.

   a) Renomeie o certificado para `cacert.crt`

   b) Copie o certificado para o armazenamento interno raiz ou no cartão SD.

      Para obter um dispositivo virtual em execução, abra Eclipse ou execute o Android Debug Bridge (ADB) para copiar o certificado para o dispositivo virtual:

      Eclipse

      i) Execute Eclipse e abra a perspectiva DDMS.

      ii) Na visualização principal, abra a janela **Explorador de arquivos**.

      iii) Abra o diretório `mnt/sdcard`.

      iv) Arraste o arquivo `cacert.crt` para o diretório `mnt/sdcard`.

      ADB

      i) Abra uma janela de comando e configure seu diretório atual para `android-sdk\platform-tools` no diretório de instalação do android; por exemplo, `C:\Program Files\Android\android-sdk\platform-tools`

      ii) Copie o certificado para o diretório `mnt/sdcard`:

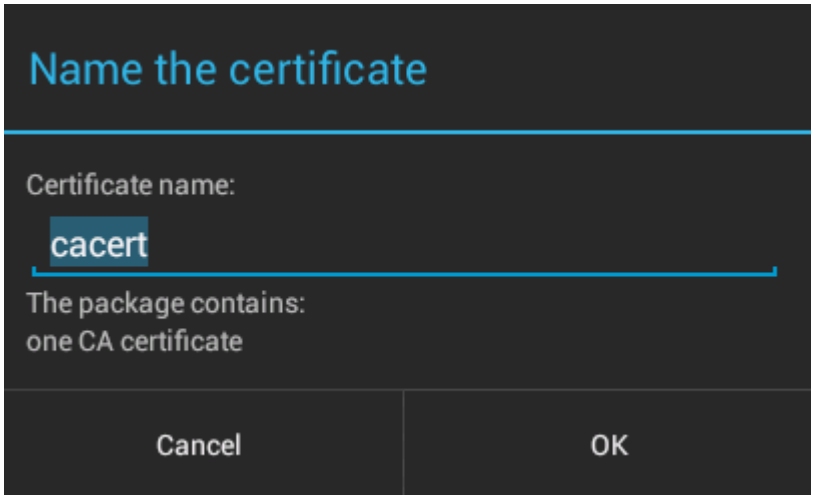
      ```
      adb push %cacert% /mnt/sdcard/cacert.crt
      ```

7. Instale o certificado no armazenamento confiável de certificado no dispositivo Android.

   O certificado deve ter uma cláusula Basic Constraints com o valor `Subject Type=CA`.

   a) Desbloqueie o dispositivo e clique no botão **widgets**.

   b) Clique em **Configurações** > **Segurança** > **Armazenamento de Credencial** > **Instalar do cartão SD**.

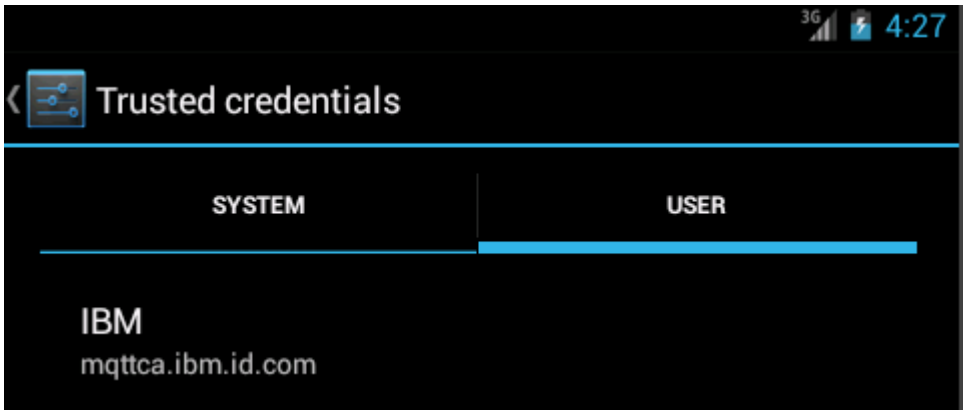


c) Confirme se o nome do arquivo de certificado está correto e clique em **OK**.

Nota: Se você não tiver definido um bloqueio para o dispositivo, será agora solicitado pelo Android para configurar um bloqueio.

8. Confirme se o certificado está instalado no dispositivo.

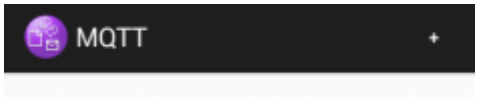
a) Clique em **Credenciais confiáveis** > **Usuário** e aguarde vários minutos para que seu certificado seja exibido na lista de certificados de usuário.



9. Execute novamente o aplicativo `MQTTExerciser` e conecte a um canal do MQTT configurado para clientes SSL anônimos.

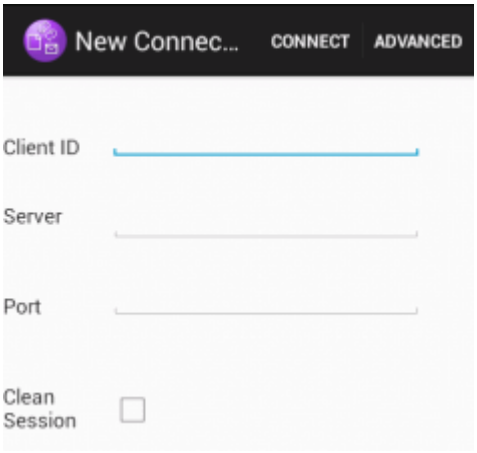
a) Abra o Aplicativo de amostra do cliente MQTT Java para Android.

Esta janela é aberta em seu dispositivo Android:



b) Conecte-se a um servidor MQTT .

i) Clique no sinal **+** para abrir uma conexão MQTT nova.



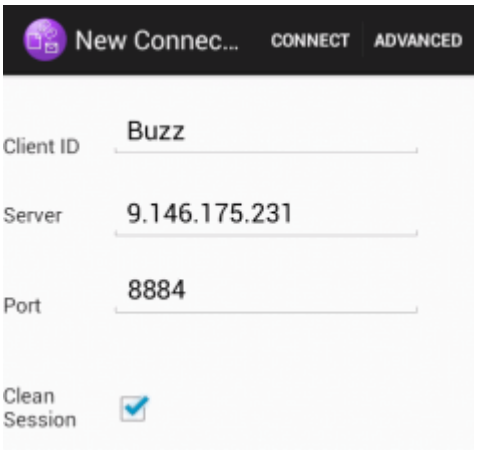
ii) Digite qualquer identificador exclusivo no campo **ID do cliente**. Seja paciente, os pressionamentos de tecla podem ser lentos.

iii) Insira no campo **Servidor** o endereço IP de seu servidor MQTT.

Este é o servidor que você escolheu na primeira etapa principal. O endereço IP não deve ser `127.0.0.1`

iv) Insira o número da porta da conexão do MQTT.

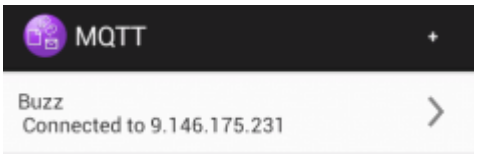
Configure o número da porta para 8884, que é configurado pela variável `%sslportopt%` nos scripts de exemplo. Esse número de porta é o número da porta do canal do MQTT configurado para clientes SSL anônimos executando os scripts de amostra na etapa "4" na página 65.



v) Clique na guia **Avançado** e selecione a opção **SSL**. Clique em **Salvar**.

vi) Clique em **Conectar**.

Se a conexão for bem-sucedida, você verá uma mensagem "Conectando", seguida por esta janela:



Resultados

O aplicativo `MQTTExerciser` demora um pouco mais para se conectar e trocar mensagens, mas, caso contrário, não se comporta de forma diferente para ser conectado em uma conexão sem segurança.

Conceitos relacionados

"Segurança do MQTT" na página 52

Três conceitos são fundamentais para a segurança do MQTT: identidade, autenticação e autorização. Identidade é como denominar o cliente que está sendo autorizado e a autoridade fornecida. A autenticação é a comprovação da identidade do cliente e a autorização é como gerenciar os direitos fornecidos com o cliente.

Tarefas relacionadas

“Gerando Chaves e Certificados” na página 96
Siga este procedimento para gerar as chaves e os certificados para o Java e os clientes C, incluindo os aplicativos Android e iOS e os servidores IBM WebSphere MQ e IBM MessageSight.

“Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro” na página 55
Com base em um exemplo do Windows, é possível funcionar adequadamente com o aplicativo seguro Java de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível"

“Autenticando um app Java do cliente MQTT com o JAAS” na página 73
Aprenda a autenticar um cliente com JAAS. Conclua as etapas nesta tarefa para modificar o programa de amostra `JAASLoginModule.java` e configurar IBM WebSphere MQ para autenticar um MQTT app Java do cliente com JAAS.

Exemplo de scripts para configurar certificados SSL para Windows

** **

Os arquivos de comandos de exemplo criam os certificados e os armazenamentos de certificados, conforme descrito nas etapas da tarefa. Além disso, o exemplo configura o cliente do gerenciador de filas do MQTT para usar o armazenamento de certificados do servidor. O exemplo exclui e recria o gerenciador de filas chamando o script `SampleMQM.bat`, fornecido com o IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` configura os nomes e caminhos para os certificados e outros parâmetros requeridos pelos comandos **keytool** e **openSSL**. As configurações são descritas nos comentários no 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

Os comandos no script `cleancert.bat` excluem o gerenciador de filas do cliente MQTT para assegurar que o armazenamento do certificado do servidor não está bloqueado, e, em seguida, excluem todos os keystores e certificados criados pelos scripts de segurança de amostra.

```
@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

Os comandos no script `genkeys.bat` criam pares de chave para sua autoridade de certificação privada, o servidor, e um cliente.

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

```
@rem -- Generate a server certificate and private key pair
%keytool% -genkeypair -noprompt -alias %srvalias% -dname %srvdname% -keystore
```

```
%srvjkskeystore% -storepass %srvjkskeystorepass% -keypass %srvkeypass% -keyalg %algorithm%
-validity %validity%
```

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

sscerts.bat

Os comandos no script `sscerts.bat` exportam os certificados autoassinados de cliente e servidor a partir de seus keystores e importam o certificado de servidor no armazenamento confiável do cliente e o certificado de cliente no keystore do servidor. O servidor não tem um armazenamento confiável. Os comandos criam um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

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

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

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

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

cacerts.bat

O script importa o certificado raiz da autoridade de certificação nos keystores privados. O certificado raiz da CA é necessário para criar o keychain entre o certificado raiz e o certificado assinado. O

script `cacerts.bat` exporta as solicitações de certificado de cliente e do servidor a partir de seus keystores. O script assina a solicitação de certificado com a chave da autoridade de certificação privada no keystore `cajkskeystore.jkse`, em seguida, importar os certificados assinados de volta para o mesmo keystores a partir do qual o a solicitação chegou. A importação cria a cadeia de certificados com o certificado raiz da CA. O script cria um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

```
@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

O script lista os keystores e certificados no diretório do certificado. Em seguida, cria o gerenciador de filas MQTT de amostra e configura os canais de telemetria seguros.

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

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

Autenticando um app Java do cliente MQTT com o JAAS

Aprenda a autenticar um cliente com JAAS. Conclua as etapas nesta tarefa para modificar o programa de amostra JAASLoginModule.java e configurar IBM WebSphere MQ para autenticar um MQTT app Java do cliente com JAAS.

Antes de começar

1. É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível". Consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente.
2. Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.

3. Você deve ter acesso às amostras MQXR `JAASLoginModule` e `JAASPrincipal` Java em uma instalação IBM WebSphere MQ . As amostras estão no caminho `%MQ_FILE_PATH%\mqxr\samples.`
4. Conclua as etapas em Windows ou Linux; os exemplos são obtidos a partir do Windows.
5. Para concluir a etapa "1" na página 74, deve-se ter autorização para criar o gerenciador de filas `MQXR_SAMPLE_QM` no IBM WebSphere MQ.

Sobre esta tarefa

Na tarefa, gere os parâmetros de identificação do cliente MQTT `Sample` a partir de sua versão do `JAASLoginModule`. A gravação de parâmetros do cliente ocasiona a modificação do programa `JAASLoginModule` de amostra e a configuração do IBM WebSphere MQ para carregar sua versão do `JAASLoginModule`.

Procedimento

1. Conclua as etapas em "Compile e execute todos os aplicativos de MQTT amostra de cliente Java a partir do Eclipse" na página 16 para executar o cliente Paho MQTT `Sample`.

   Seu objetivo é preparar um ambiente de desenvolvimento para desenvolver e testar a autenticação do JAAS. Requeira um ambiente de desenvolvimento do Java para customizar o módulo de autenticação do JAAS. No exemplo, execute o cliente Paho de amostra para o Java testar sua configuração do JAAS. Para simplificar, use o mesmo ambiente de desenvolvimento para modificar o cliente de amostra e o módulo de login de amostra do JAAS. Como alternativa, teste seu módulo de login do JAAS com o cliente MQTT para C ou qualquer outro cliente MQTT.

2. Opcional: Inclua um nome de usuário e um parâmetro de senha na amostra do MQTT Paho.

   Nota: Se seu cliente Paho para Java incluir o nome do usuário e os parâmetros de senha, essa etapa será desnecessária. Verifique o site de download para uma atualização. Consulte downloads da comunidade do sistema de mensagens daIBM, caso contrário, altere sua cópia do `Sample.java`

   a) Abra o Package Explorer no pacote `org.eclipse.paho.sample.mqttv3app` no projeto de amostras do Paho.

   b) Clique com o botão direito em `Sample.java` **Copiar** > **Colar**. Na janela **Conflito de nome**, digite o nome `SampleForJAAS`.

   c) Inclua as seguintes linhas de código para o método `main`.

      i) Após a linha "`boolean ssl = false;`", declare as variáveis `userName` e `password` :

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

      Para a compatibilidade com antigos servidores MQTT, por padrão não configure os parâmetros de nome do usuário e senha.

      ii) Após a linha, "`case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;`", analise os dois novos parâmetros de entrada:

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

      iii) Antes da linha, "`if (action.equals("publish")) {`", inclua `userName` e `password` nos argumentos do construtor `Sample`:

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

   d) Inclua `userName` e `password` no construtor do `Sample`.

      Mude:

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

Para:

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

e) Inclua as seguintes linhas de código no construtor `Sample`.

Após a linha "`conOpt.setCleanSession(clean);`", configure as variáveis `userName` e `password` no objeto `conOpt` no método `Sample`:

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

f) Nos métodos `publish` e `subscribe`, modifique as seguintes linhas de código:

Mude a linha "`client.connect();`" para

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

3. Crie um projeto do Java, `JAASSample`, por exemplo do JAAS.
   a) Na área de trabalho do Eclipse, abra o assistente **Novo projeto Java**: clique em **Arquivo** > **Novo** > **Projeto do Java**.
   b) No campo **Nome do projeto**, digite `JAASSample`.
   c) Nas opções do JRE, selecione `J2SE-1.5` como o JRE do ambiente de execução e clique em **Avançar**.

      O JRE deve corresponder ao JRE que seu servidor IBM WebSphere MQ executa. IBM WebSphere MQ Version 7.5 executa `J2SE-1.5`.
   d) Na janela **Configurações de Java**, clique na guia **Bibliotecas** e clique em **Incluir JARs externos ...**. Vá até o`%MQ_FILE_PATH%\mqxr\lib` e selecione **MQXR.jar**; clique em **Concluir**.
4. Importe as classes de JAAS amostras `JAASLoginModule` e `JAASPrincipal`
   a) Clique com o botão direito no projeto `JAASSample` no Package Explorer **Importar ...** > **Geral** > **Sistema de Arquivos** e clique em **Avançar**..
   b) Navegue até `%MQ_FILE_PATH%\mqxr\samples` e verifique `JAASLoginModule.java` e `JAASPrincipal.java`; clique em **Concluir**.
   c) Selecione e clique com o botão direito em ambos os arquivos Java no Package Explorer, **Refatorar ...** > **Move**.
   d) Na janela **Mover**, verifique se `JAASSample` está selecionado como o destino para ambos os elementos e clique em **Criar pacote...**
   e) Digite `samples` no campo **Nome** no assistente **Novo pacote Java**; clique em **Concluir** > **OK**

   O Eclipse constrói as classes do Java importadas com um número de avisos sobre os valores não usados.
5. Renomeie a classe `JAASLoginModule`

   Renomeie a classe para que seja mais fácil distingui-la a partir da classe `JAASLoginModule` de amostra fornecida com o IBM WebSphere MQ.
   a) Clique com o botão direito em `JAASloginModule.java` no explorador de pacotes **Refatorar ...** > **Renomear**.
   b) Na janela **Renomear unidade de compilação**, mude o campo **Novo nome** de `JAASloginModule` para `MyJAASloginModule`; clique em **Concluir**.
6. Modifique a classe `MyJAASloginModule` para gerar o conteúdo dos campos de retorno de chamada.

a) Inclua a seguinte linha de código para `MyJAASloginModule.java`.

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

Coloque as linhas logo antes da instrução:`"if (true) loggedIn = true;"`

b) Pressione `CTRL+Shift+O` para reorganizar as importações e salvar o arquivo.

7. Renomeie `JAASPrincipal` para `MyJAASPrincipal`.

Renomeie a classe para evitar confusão com a classe `JAASPrincipal` de amostra. No exemplo, deixe o conteúdo da classe `MyJAASPrincipal` inalterado.

8. Fornecer o ID do usuário que está executando o gerenciador de filas processa as permissões `read` e `execute` para as classes JAAS.

a) No Windows Explorer, abra seu diretório da área de trabalho do Eclipse. No exemplo, o local da área de trabalho do Eclipse é representado pela variável do Eclipse , *`workspace_loc`*.

b) Navegue até o diretório que contém suas classes `MyJAASLoginModule` e `MyJAASPrincipal`.

O caminho do diretório é *`workspace_loc`*`\JAASSample\bin\samples`

c) Selecione e, em seguida, clique com o botão direito nas duas classes e clique em **Propriedades**; clique na guia **Segurança** na janela **Propriedades**.

d) Clique em **Incluir ...**, digite o nome do objeto `mqm`e clique em **Verificar nomes** para verificá-lo; clique em **OK**.

e) Selecione **mqm** na lista de **Grupo ou nomes de usuário** e verifique **ler e executar** e **ler** na lista de permissões para `mqm`; clique em OK.

9. Configure o IBM WebSphere MQ para executar sua classe `MyJAASLoginModule`.

a) Inclua um arquivo `service.env` na configuração do IBM WebSphere MQ para definir os caminhos de classe para carregar sua classe `MyJAASLoginModule`.

Crie um arquivo `service.env` com a instrução de caminho de classe a seguir em seu diretório *`WMQ_DATA_PATH`*:

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

Em que *`user.dir`* é a raiz do diretório para os arquivos de classe compilados em sua área de trabalho do Eclipse. O diretório *`WMQ_DATA_PATH`* contém o diretório `qmgrs`. Consulte Variáveis de ambiente adicionais.

Sugestão: `CLASSPATH=`*`user.dir`*`\JAASSample\bin` pode ser a única instrução no arquivo `service.env`

b) Inclua uma sub-rotina, `MyJAASStanza`, no arquivo `jaas.config` para identificar sua classe `MyJAASLoginModule` relativa aos caminhos de classe no arquivo `service.env`.

`jaas.config` está no diretório `mqxr` do gerenciador de filas; *`WMQ_DATA_PATH`*`\Qmgrs\MQXR_SAMPLE_QM\mqxr`.

A sub-rotina é:

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

c) Configure um canal do IBM WebSphere MQ Telemetry com o nome de sua sub-rotina de configuração do JAAS.

Execute o seguinte comando a partir de uma janela de comando:

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

O comando liga o canal MyJAAS ao MyJAASStanza no arquivo jaas.config. Não especificando uma opção MCAUSER ou especificando a opção USECLTID na definição de canal, o canal autoriza o acesso a recursos do gerenciador de filas com o nome do usuário fornecido pelo programa do cliente MQTT. No exemplo, o nome do usuário fornecido pelo cliente é configurado como "Guest". As autorizações existentes para Guest configuradas pelo arquivo de comando SampleMQM também são usadas neste exemplo.

10. Reinicie o serviço IBM WebSphere MQ Telemetry para ler os novos dados de configuração.
    Para reiniciar o serviço do IBM WebSphere MQ Telemetry, inicie o gerenciador de filas ou o serviço a partir de IBM WebSphere MQ Explorer ou execute os seguintes comandos para a configuração de amostra:

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

11. Execute o programa Sample.

    Para definir uma configuração de execução para SampleForJAAS siga o mesmo procedimento que na etapa "1" na página 74, com as seguintes modificações:

    a) Configure o número da porta para 1890 para corresponder à configuração de canal do MQTT.

    b) Inclua os parâmetros -u Guest -z password nas senhas na guia **(x)= Argumentos** para as configurações Assinante e Publicador criadas para o programa Sample

    Os programas de amostra são executados sem nenhuma mudança na saída, exceto o número da porta agora é 1890 em vez de 1883.

    No diretório *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM, abra o arquivo mqxr.stdout. A saída de MyJAASLoginModule é gravada para mqxr.stdout:

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

Como proceder a seguir

Se seu exemplo não funcionar, leia o tópico de resolução de problemas para JAAS; "Resolução do problema: o módulo de login JAAS não é chamado pelo serviço de telemetria" na página 187 e tente estas dicas de depuração.

1. Inclua -verbose nos parâmetros em *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties. Nesse log, é possível ver se sua classe foi carregada com sucesso.

   A saída é gravada para *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr.
2. Consulte *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log para verificar as exceções lançadas em MyJAASLoginModule. Por exemplo, se você tentar gerar uma password nula, que é uma matriz de caracteres e não uma sequência de caracteres, uma exceção será lançada.
3. Consulte *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log. Se o nome do usuário no userName não for autorizado a acessar os recursos do gerenciador de filas e o canal for configurado sem nenhuma opção MCAUSER ou USECLTID, nenhum erro será relatado aqui.
4. Verifique se o nome da sub-rotina em *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config é o mesmo que o nome no canal do MQTT configurado para a porta a qual o cliente Sample está tentando se conectar.

5. Verifique se o caminho na sub-rotina corresponde ao caminho para a classe `MyJAASLoginModule` em Eclipse; por exemplo:

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

6. Para eliminar se a falha que reside no caminho de classe no arquivo `service.env` em *WMQ_DATA_PATH* não está sendo captada corretamente, mude a linha "`set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%`" em `%MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT` para incluir o caminho de classe. Também é possível repetir o caminho de classe. No entanto, o caminho de classe não contém o caminho da classe que é configurado em `service.env`, portanto, isto funcionará somente se você modificar o arquivo `controlMQXR.BAT`.

Conceitos relacionados

“Segurança do MQTT” na página 52
Três conceitos são fundamentais para a segurança do MQTT: identidade, autenticação e autorização. Identidade é como denominar o cliente que está sendo autorizado e a autoridade fornecida. A autenticação é a comprovação da identidade do cliente e a autorização é como gerenciar os direitos fornecidos com o cliente.

“Configuração JAAS do Canal de Telemetria” na página 115
Configure JAAS para autenticar o `Username` enviado pelo cliente.

Tarefas relacionadas

“Resolução do problema: o módulo de login JAAS não é chamado pelo serviço de telemetria” na página 187
Descubra se o módulo de login JAAS não está sendo chamado pelo serviço de telemetria (MQXR) e configure o JAAS para corrigir o problema.

“Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro” na página 55
Com base em um exemplo do Windows, é possível funcionar adequadamente com o aplicativo seguro Java de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível"

“Conectando o aplicativo Java de amostra do cliente MQTT no Android sobre SSL” na página 64
Funcionando adequadamente com o cliente Android MQTT de amostra conectado ao IBM WebSphere MQ através do SSL.

Informações relacionadas

Variáveis de ambiente adicionais

Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets

Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

Antes de começar

1. Deve-se ter acesso a um servidor MQTT version 3 que suporte o MQTT protocol através do WebSockets.
2. O navegador deve suportar SSL e o WebSocket protocol. Consulte o “Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL” na página 175.
3. Os canais de SSL devem ser iniciados.

Sobre esta tarefa

Conclua esta tarefa para executar o cliente do sistema de mensagens do MQTT para as páginas de amostra do JavaScript através do SSL. A tarefa direciona você para o “Gerando Chaves e Certificados” na página 96 criar certificados e configurar o IBM WebSphere MQ.

Proteja o canal SSL com chaves assinadas de autoridade de certificado ou chaves auto-assinadas.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o MQTT protocol através do WebSockets seguro.

   - IBM MessageSight e liberações do IBM WebSphere MQ Versão 7.5.0.1 e posterior, faça isso.
2. Opcional: Instale um kit de desenvolvimento do Java (JDK) na Versão 7 ou posterior.

   Se você estiver configurando um sistema de teste e deseja usar certificados autoassinados, será necessário usar o comando **keytool** do JDK versão 7 para confirmar seus certificados. Se você estiver configurando um sistema de produção e enviando as solicitações de assinatura de certificado (CSR) a uma autoridade externa de certificado, não será necessário o JDK Versão 7.
3. Crie e execute os scripts para gerar pares de chaves e certificados e configure IBM WebSphere MQ como o servidor MQTT .

   Siga as etapas em “Gerando Chaves e Certificados” na página 96 para criar e executar os scripts. Os scripts também são listados em “Exemplo de scripts para configurar certificados SSL para Windows” na página 81.

   O nome comum do certificado do servidor deve corresponder ao nome de DNS do canal do servidor. Alguns navegadores aceitam certificados que contêm uma lista de nomes comuns; por exemplo:

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

   Outros navegadores aceitam apenas um nome comum. Por exemplo, Firefox, até a versão 18, aceita apenas um nome comum. As versões posteriores podem ser diferentes.
4. Verifique se os canais SSL estão executando e se estão configurados como você espera.

   No IBM WebSphere MQ, digite o seguinte comando em uma janela de 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. Instale os certificados no armazenamento de certificados do navegador.

   Por exemplo, escolha um dos seguintes certificados do servidor:

   a. Para obter um certificado de servidor autoassinado, o certificado é `srvcertselfsigned.cer`.

   b. Para obter um certificado do servidor assinado por sua autoridade privada de certificação, o certificado é `cacert.cer`.

   c. Para obter um certificado do servidor assinado por uma autoridade externa de certificação, verifique se o certificado raiz da autoridade de certificação já está instalado no armazenamento de certificados.

   Dependendo do suporte disponível em seu navegador, instale o `cacert.cer` na lista de Autoridades de Certificação de Raiz Confiável. Consulte o “Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL” na página 175.

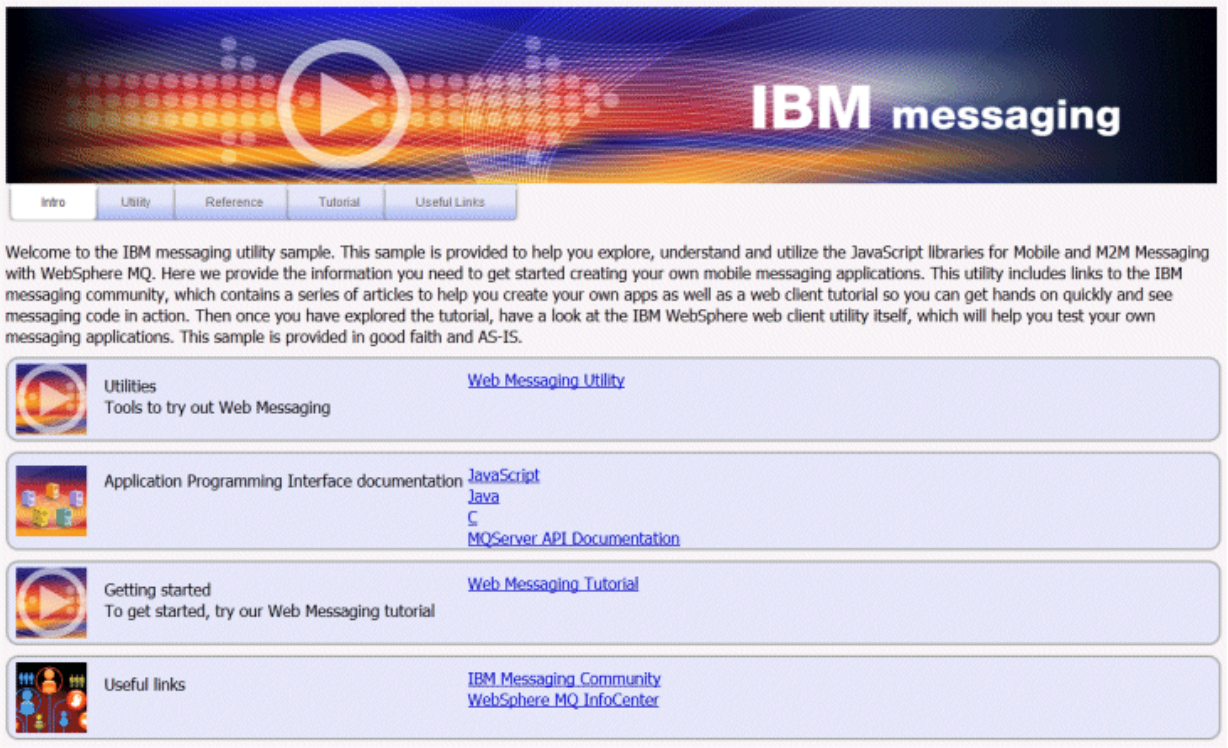
6. Opcional: Autenticar o cliente.

   No exemplo, instale o `cacert.cer` para autenticar o servidor e criptografar o canal, mas não para autenticar o cliente. Para autenticar o cliente, deve-se instalar o keystore do cliente, `cltkeystore.p12`, no navegador. Nem todos os navegadores suportam a autenticação de clientes. Consulte o "Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL" na página 175.
7. Conecte-se ao canal seguro do WebSockets.

   Abra o navegador e digite a URL do canal do WebSockets na barra de endereço; no exemplo:

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

   IBM WebSphere MQ responde com a primeira página do Páginas JavaScript de amostra do cliente de sistema de mensagens MQTT.

   Se a conexão falhar e você executou os scripts de exemplo para configurar o gerenciador de filas do MQTT de amostra, tente conectar-se ao canal normal do WebSockets na porta 1886. O sucesso na porta 1886 isola a falha para a conexão SSL.

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

Conceitos relacionados

"O Cliente de sistema de mensagens do MQTT para JavaScript e os aplicativos da web" na página 117

"Como programar os aplicativos do sistema de mensagens no JavaScript" na página 121

Tarefas relacionadas

"Gerando Chaves e Certificados" na página 96

Siga este procedimento para gerar as chaves e os certificados para o Java e os clientes C, incluindo os aplicativos Android e iOS e os servidores IBM WebSphere MQ e IBM MessageSight.

"Introdução ao Cliente de sistema de mensagens do MQTT para JavaScript" na página 24

É possível iniciar com o Cliente de sistema de mensagens do MQTT para JavaScript, exibindo a página inicial de amostra do cliente com o sistema de mensagens e procurando os recursos aos quais ele se vincula. Para exibir esta página inicial, configure um servidor MQTT para aceitar conexões do Páginas JavaScript de amostra do cliente de sistema de mensagens MQTT, em seguida, digite a URL configurada no servidor em um navegador da web. O Cliente de sistema de mensagens do MQTT para JavaScript

inicia automaticamente em seu dispositivo e a página inicial de amostra do cliente com o sistema de mensagens é exibida. Esta página contém links para utilitários, documentação da interface de programação, um tutorial e outras informações úteis.

Referências relacionadas

“Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL” na página 175

Há diferenças na capacidade entre navegadores diferentes, em plataformas diferentes. Entender essas diferenças ajuda você a configurar aplicativos, autoridades de certificação (CAs) e certificados de cliente para se conectar usando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets.

Exemplo de scripts para configurar certificados SSL para Windows

** **

Os arquivos de comandos de exemplo criam os certificados e os armazenamentos de certificados, conforme descrito nas etapas da tarefa. Além disso, o exemplo configura o cliente do gerenciador de filas do MQTT para usar o armazenamento de certificados do servidor. O exemplo exclui e recria o gerenciador de filas chamando o script `SampleMQM.bat`, fornecido com o IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` configura os nomes e caminhos para os certificados e outros parâmetros requeridos pelos comandos **`keytool`** e **openSSL**. As configurações são descritas nos comentários no 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.
```

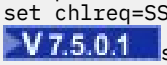
```
@rem Alternatively, if you export a certificate signing request (CSR)
@rem from the server keystore for the server key,
@rem and import the signed certificate back into the same keystore,
@rem it forms a certificate chain.
@rem The certificate chain links the server certificate to the CA.
@rem When 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

Os comandos no script `cleancert.bat` excluem o gerenciador de filas do cliente MQTT para assegurar que o armazenamento do certificado do servidor não está bloqueado, e, em seguida, excluem todos os keystores e certificados criados pelos scripts de segurança de amostra.

```
@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

Os comandos no script `genkeys.bat` criam pares de chave para sua autoridade de certificação privada, o servidor, e um cliente.

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

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

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

sscerts.bat

Os comandos no script `sscerts.bat` exportam os certificados autoassinados de cliente e servidor a partir de seus keystores e importam o certificado de servidor no armazenamento confiável do cliente e o certificado de cliente no keystore do servidor. O servidor não tem um armazenamento confiável. Os comandos criam um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

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

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

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

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

cacerts.bat

O script importa o certificado raiz da autoridade de certificação nos keystores privados. O certificado raiz da CA é necessário para criar o keychain entre o certificado raiz e o certificado assinado. O script `cacerts.bat` exporta as solicitações de certificado de cliente e do servidor a partir de seus keystores. O script assina a solicitação de certificado com a chave da autoridade de certificação privada no keystore `cajkskeystore.jkse`, em seguida, importar os certificados assinados de volta para o mesmo keystores a partir do qual o a solicitação chegou. A importação cria a cadeia de

certificados com o certificado raiz da CA. O script cria um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

```
@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

O script lista os keystores e certificados no diretório do certificado. Em seguida, cria o gerenciador de filas MQTT de amostra e configura os canais de telemetria seguros.

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

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

Construindo e executando o Aplicativo C de amostra do cliente do MQTT seguro

Com base em um exemplo do Windows, é possível o funcionamento adequado com o aplicativo C seguro de amostra em qualquer sistema operacional ao qual é possível compilar a origem C. Verifique se é possível executar o aplicativo C de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

Antes de começar

1. Deve-se ter acesso a um servidor MQTT version 3.1 que suporte MQTT protocol sobre SSL.
2. Se houver um firewall entre seu cliente e o servidor, verifique se ele não bloqueia o MQTT tráfego.
3. As versões binárias do cliente para as bibliotecas C são fornecidas para um número de sistemas operacionais. Para alguns desses sistemas operacionais, a versão segura do cliente não é fornecida como um arquivo binário. Para esses sistemas operacionais, deve-se seguir as instruções em “Construindo o cliente MQTT para as bibliotecas C” na página 31.

4. Para resolução de problemas, o suporte IBM pode requerer que você execute o cliente MQTT para C em uma plataforma de referência.
5. Os canais de SSL devem ser iniciados.

Para obter uma visão geral de plataformas com suporte e referência, consulte Requisitos do sistema para IBM Mobile Messaging and M2M Pacote do Cliente. Para obter detalhes do que é suportado para o cliente C, consulte as seções relevantes de Requisitos do Sistema para WebSphere MQ V7.5 Telemetry.

Sobre esta tarefa

Como uma ilustração, este artigo mostra como compilar e executar o Aplicativo C de amostra do cliente do MQTT seguro no Windows a partir da linha de comandos. Na ilustração, o Microsoft Visual Studio 2010 é usado para compilar o cliente. É possível modificar os scripts da linha de comandos para compilar e executar o aplicativo de amostra em outros sistemas operacionais, como Linux e iOS.

Nota:

Os scripts do Windows fornecidos neste artigo assumem a construção de todo o pacote OpenSSL de origem. Se você optar por usar as bibliotecas pré-compiladas que a IBM fornece, você também pode preferir obter uma liberação binária pré-compilada de OpenSSL As bibliotecas pré-compiladas não estão disponíveis para uso com o iOS.

Proteja o canal SSL com chaves assinadas de autoridade de certificado ou chaves auto-assinadas.

Procedimento

1. Escolha um servidor MQTT ao qual é possível conectar o app cliente.

   O servidor deve suportar o protocolo MQTT version 3.1 sobre SSL. Todos os servidores MQTT do IBM fazem isso, incluindo IBM WebSphere MQ e IBM MessageSight Consulte “Introdução aos servidores MQTT” na página 138.
2. Opcional: Instale um Java kit de desenvolvimento (JDK) na Versão 7 ou posterior.

   Versão 7 é necessário para executar o comando **keytool** para certificar certificados. Se você não irá certificar certificados, o JDK versão 7 não é necessário.
3. Instale um ambiente de desenvolvimento C na plataforma na qual estiver construindo.

   Os arquivos de criação nos exemplos neste tópico são voltados para as seguintes ferramentas:

   - iOS Para iOS, no Apple Mac com OS X 10.8.2 com as ferramentas de desenvolvimento iOS de Xcode.
   - Linux Para o Linux, a versão gcc 4.4.6 a partir do Red Hat Enterprise Linux versão 6.2.

     O nível mínimo suportado da biblioteca C `glibc` é 2.12, e o kernel Linux é 2.6.32.
   - Windows Para o Microsoft Windows, Visual Studio versão 10.0.
4. Faça download do Mobile Messaging and M2M Pacote do Cliente e instale o MQTT SDK.

   Não há nenhum programa de instalação, apenas expanda o arquivo transferido por download.

   a. Faça download do Mobile Messaging and M2M Pacote do Cliente.
   b. Crie uma pasta na qual você irá instalar o SDK.

      Você pode desejar nomear a pasta `MQTT`. O caminho para esta pasta é referido aqui como *`sdkroot`*.
   c. Expanda o arquivo compactado do Mobile Messaging and M2M Pacote do Cliente em *`sdkroot`*. A expansão cria uma árvore de diretórios que inicia em *`sdkroot`*\SDK.
5. Opcional: Siga as etapas em “Construindo o cliente MQTT para as bibliotecas C” na página 31.

   Execute esta etapa apenas se o MQTT SDK não incluir a biblioteca segura do cliente C para o sistema operacional de destino.

- Windows As bibliotecas são `mqttv3cs.lib` para compilação e `mqttv3cs.dll` para execução.
- Linux A biblioteca é `libmqttv3cs.so`
- iOS A biblioteca é `libmqttv3cs.a`

6. Crie e execute os scripts para gerar pares de chaves e certificados e configure IBM WebSphere MQ como o servidor MQTT .

   Siga as etapas em "Gerando Chaves e Certificados" na página 96 para criar e executar os scripts. Os scripts também são listados em "Exemplo de scripts para configurar certificados SSL para Windows" na página 90.
7. Verifique se os canais SSL estão executando e se estão configurados como você espera.

   No IBM WebSphere MQ, digite o seguinte comando em uma janela de 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. Crie os scripts para construir e executar o Aplicativo C de amostra do cliente do MQTT seguro.
   a) Crie e execute `sscclient.bat` para testar um canal SSL protegido com certificados autoassinados.
   b) Crie e execute `cacclient.bat` para testar um canal SSL protegido com certificados assinados com autoridade de certificação.

Resultados

Os resultados são semelhantes à execução do cliente descoberto.

```
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. Assinante seguro

```
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
MQTTV3SSample.c
Connected to ssl://localhost:8884
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
Connected to ssl://localhost:8885
Publishing to topic "MQTTV3SSample/C/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 17. Publicador seguro

Scripts para executar o Aplicativo C de amostra do cliente do MQTT seguro

Execute os scripts em "Exemplo de scripts para configurar certificados SSL para Windows" na página 90 antes de executar esses scripts.

O Aplicativo C de amostra do cliente do MQTT seguro com certificados autoassinados.

Execute este script com os certificados autoassinados criados executando o 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`

Execute o aplicativo C seguro de amostra do cliente MQTT com certificados assinados com autoridade de certificação.

Execute este script com os certificados assinados com autoridade de certificação criados executando o 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`

Conceitos relacionados

“Segurança do MQTT” na página 52
Três conceitos são fundamentais para a segurança do MQTT: identidade, autenticação e autorização. Identidade é como denominar o cliente que está sendo autorizado e a autoridade fornecida. A autenticação é a comprovação da identidade do cliente e a autorização é como gerenciar os direitos fornecidos com o cliente.

Tarefas relacionadas

“Gerando Chaves e Certificados” na página 96
Siga este procedimento para gerar as chaves e os certificados para o Java e os clientes C, incluindo os aplicativos Android e iOS e os servidores IBM WebSphere MQ e IBM MessageSight.

Exemplo de scripts para configurar certificados SSL para Windows

Exemplo

Os arquivos de comandos de exemplo criam os certificados e os armazenamentos de certificados, conforme descrito nas etapas da tarefa. Além disso, o exemplo configura o cliente do gerenciador de filas do MQTT para usar o armazenamento de certificados do servidor. O exemplo exclui e recria o gerenciador de filas chamando o script `SampleMQM.bat`, fornecido com o IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` configura os nomes e caminhos para os certificados e outros parâmetros requeridos pelos comandos **`keytool`** e **openSSL**. As configurações são descritas nos comentários no 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

Os comandos no script `cleancert.bat` excluem o gerenciador de filas do cliente MQTT para assegurar que o armazenamento do certificado do servidor não está bloqueado, e, em seguida, excluem todos os keystores e certificados criados pelos scripts de segurança de amostra.

```
@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

Os comandos no script genkeys.bat criam pares de chave para sua autoridade de certificação privada, o servidor, e um cliente.

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

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

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

sscerts.bat

Os comandos no script sscerts.bat exportam os certificados autoassinados de cliente e servidor a partir de seus keystores e importam o certificado de servidor no armazenamento confiável do cliente e o certificado de cliente no keystore do servidor. O servidor não tem um armazenamento confiável. Os comandos criam um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

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

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

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

```
%srvjkskeystore% -storepass %srvjkskeystorepass%
```

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

cacerts.bat

O script importa o certificado raiz da autoridade de certificação nos keystores privados. O certificado raiz da CA é necessário para criar o keychain entre o certificado raiz e o certificado assinado. O script `cacerts.bat` exporta as solicitações de certificado de cliente e do servidor a partir de seus keystores. O script assina a solicitação de certificado com a chave da autoridade de certificação privada no keystore `cajkskeystore.jkse`, em seguida, importar os certificados assinados de volta para o mesmo keystores a partir do qual o a solicitação chegou. A importação cria a cadeia de certificados com o certificado raiz da CA. O script cria um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

```
@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

O script lista os keystores e certificados no diretório do certificado. Em seguida, cria o gerenciador de filas MQTT de amostra e configura os canais de telemetria seguros.

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

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

V 7.5.0.1

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

Gerando Chaves e Certificados

Siga este procedimento para gerar as chaves e os certificados para o Java e os clientes C, incluindo os aplicativos Android e iOS e os servidores IBM WebSphere MQ e IBM MessageSight.

Antes de começar

1. Deve-se ter uma cópia do comando **keytool**. Nem todas as versões do **keytool** suportam a conversão de keystores do Java keystore (JKS) para o Public Key Cryptographic System (PKCS) ou a assinatura de solicitações de certificado. O exemplo usa o comando **keytool** em JDK Versão 7.0, que suporta esses recursos.
2. Se você pretende gerar chaves e certificados para o cliente para C, que estão no formato Privacy-Enhanced Mail (PEM), deve-se ter uma cópia do comando **openSSL**. Siga as etapas no "Construindo o cliente MQTT para as bibliotecas C" na página 31 para criar o pacote openSSL.
3. Mude os valores de parâmetros no script `initcert.bat` para atender às suas necessidades. Em particular, você pode escolher omitir os parâmetros de senha para evitar a gravação de senhas. O comando **keytool** solicitará as senhas ausentes.

Sobre esta tarefa

É necessário chaves e certificados para criar conexões SSL seguras entre clientes e servidoresMQTT. Esta tarefa mostra duas maneiras diferentes para criar as chaves e os certificados necessários: autoassinado e assinado por sua própria autoridade de certificação. O método seguido depende de como você planeja gerenciar keystores e certificados.

Para usar certificados assinados por uma autoridade de certificação externa, substitua a etapa de assinatura em `cacerts.bat`, com o envio de solicitações de certificado para uma autoridade de certificação externa. A autoridade de certificação pode retornar um certificado raiz e um intermediário em adição ao certificado assinado. Siga a orientação fornecida pela CA externa sobre o local para instalar os certificados retornados.

O servidor IBM WebSphere MQ procura os certificados apenas no armazenamento de certificados especificado nos parâmetros de configuração do canal de telemetria. Ele não procura adicionalmente no armazenamento JSE `cacerts`. Um cliente Java procura certificados no armazenamento confiável especificado. Se você não especificar um armazenamento confiável, ele procurará no keystore `cacerts` no diretório JSE `jre\lib\security` . Os clientes Android procuram certificados no armazenamento de certificados predefinido no dispositivo Android. Os aplicativos do cliente C e os aplicativos do iOS procuram apenas nos armazenamentos de certificados que o aplicativo especificar.

Os clientes Android e Java procuram um truststore pré-configurado para certificados confiáveis. Os certificados raiz de CA são armazenados no armazenamento de certificados confiáveis do Android e no armazenamento do JSE `jre\lib\security\cacerts` . Se o certificado raiz da CA que notificou o certificado do servidor já estiver instalado no armazenamento confiável pré-configurado, não defina um armazenamento confiável do cliente. A única configuração necessária é definir a porta TCP/IP para o servidor protegido de canal do MQTT.

As ferramentas para criar chaves e certificados e gerenciar todos os formatos diferentes, não são simples de usar. Elas possuem muitos parâmetros para gerenciar e **openSSL** requer um arquivo de configuração, `openssl.cnf` e os parâmetros da linha de comandos. Nenhuma ferramenta fornece todas as funções necessárias para gerenciar chaves e certificados para aplicativos executados no C e Java. Os canais de telemetria no IBM WebSphere MQ requerem um keystore JKS e, portanto, os exemplos usam

principalmente as ferramentas de certificado do Java, **ikeyman** e **keytool**. No entanto, as ferramentas do Java não suportam o formato PEM, que é necessário para aplicativos clientes C. Para criar keystores no formato PEM, execute a ferramenta **openSSL**. A ferramenta **openSSL** converte os keystores do formato PKCS12 para PEM e **keytool** converte os keystores entre o formato JKS e PKCS12. Nenhum arquivo `openssl.cnf` é necessário para conversão de keystore. **openSSL** será necessário apenas se você planeja construir aplicativos cliente C ou aplicativos iOS. Se você preferir trabalhar com **openSSL**, será possível usá-lo para assinar certificados em vez de certificados de assinatura com o **keytool**.

Procedimento

1. Abra uma janela de comandos para executar os seguintes scripts.
2. Crie e execute o script `initcert.bat` para configurar os parâmetros necessários para executar os clientes seguros de amostra do MQTT.
3. Crie e execute o script `cleancert.bat` para limpar o ambiente pronto para criar novos keystores e certificados.
4. Crie e execute o script `genkeys.bat` para gerar os pares de chaves necessários.
5. Efetue uma das seguintes opções:
   - Crie e execute o script `sscerts.bat` para criar certificados autoassinados.
   - Crie e execute o script `cacerts.bat` para criar as cadeias de certificados assinados com autoridade de certificação.
6. Crie e execute o script `mqcerts.bat` para criar o gerenciador de filas `MQXR_SAMPLE_QM` e configurar seus canais de telemetria.

Tarefas relacionadas

“Construindo e executando o Aplicativo C de amostra do cliente do MQTT seguro” na página 86
Com base em um exemplo do Windows, é possível o funcionamento adequado com o aplicativo C seguro de amostra em qualquer sistema operacional ao qual é possível compilar a origem C. Verifique se é possível executar o aplicativo C de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

“Construindo e executando o Aplicativo de amostra do cliente MQTT Java seguro” na página 55
Com base em um exemplo do Windows, é possível funcionar adequadamente com o aplicativo seguro Java de amostra no IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.É possível executar um app Cliente de MQTT para Java em qualquer plataforma com JSE 1.5 ou superior que seja "Java Compatível"

Exemplo de scripts para configurar certificados SSL para Windows

Exemplo

Os arquivos de comandos de exemplo criam os certificados e os armazenamentos de certificados, conforme descrito nas etapas da tarefa. Além disso, o exemplo configura o cliente do gerenciador de filas do MQTT para usar o armazenamento de certificados do servidor. O exemplo exclui e recria o gerenciador de filas chamando o script `SampleMQM.bat`, fornecido com o IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` configura os nomes e caminhos para os certificados e outros parâmetros requeridos pelos comandos **keytool** e **openSSL**. As configurações são descritas nos comentários no 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

Os comandos no script `cleancert.bat` excluem o gerenciador de filas do cliente MQTT para assegurar que o armazenamento do certificado do servidor não está bloqueado, e, em seguida, excluem todos os keystores e certificados criados pelos scripts de segurança de amostra.

```
@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

Os comandos no script `genkeys.bat` criam pares de chave para sua autoridade de certificação privada, o servidor, e um cliente.

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

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

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

sscerts.bat

Os comandos no script `sscerts.bat` exportam os certificados autoassinados de cliente e servidor a partir de seus keystores e importam o certificado de servidor no armazenamento confiável do cliente e o certificado de cliente no keystore do servidor. O servidor não tem um armazenamento confiável. Os comandos criam um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

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

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

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

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

cacerts.bat

O script importa o certificado raiz da autoridade de certificação nos keystores privados. O certificado raiz da CA é necessário para criar o keychain entre o certificado raiz e o certificado assinado. O script `cacerts.bat` exporta as solicitações de certificado de cliente e do servidor a partir de seus keystores. O script assina a solicitação de certificado com a chave da autoridade de certificação privada no keystore `cajkskeystore.jkse`, em seguida, importar os certificados assinados de volta para o mesmo keystores a partir do qual o a solicitação chegou. A importação cria a cadeia de certificados com o certificado raiz da CA. O script cria um armazenamento confiável do cliente em formato PEM do armazenamento confiável JKS do cliente.

```
@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

O script lista os keystores e certificados no diretório do certificado. Em seguida, cria o gerenciador de filas MQTT de amostra e configura os canais de telemetria seguros.

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

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

V 7.5.0.1

```
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopttws%)
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
```

Identificação, Autorização e Autenticação de Cliente MQTT

O serviço de telemetria (MQXR) publica ou assina os tópicos do WebSphere MQ em nome de clientes MQTT usando canais do MQTT. O administrador do WebSphere MQ configura a identidade do canal MQTT que é usado para autorização do WebSphere MQ. O administrador pode definir uma identidade comum para o canal ou usar `Username` ou `ClientIdentifier` de um cliente conectado ao canal.

O serviço de telemetria (MQXR) pode autenticar o cliente usando o `Username` fornecido pelo cliente ou usando um certificado de cliente. O `Username` é autentiou do usando uma senha fornecida pelo cliente.

Resumindo: Identificação de cliente é a seleção da identidade do cliente. Dependendo do contexto, o cliente é identifiou do por `ClientIdentifier`, `Username`, uma identidade de cliente comum criada pelo administrador ou um certifiou do de cliente. O identifiou dor de cliente usado para verifiou ção de autenticidade não precisa ser o mesmo identifiou dor que é usado para autorização.

Programas clientes MQTT configuram `Username` e `Password` que são enviados para o servidor com o uso de um canal MQTT. Eles também podem configurar as propriedades SSL necessárias para criptografar e autentiou r a conexão. O administrador decide se irá autenticar o canal MQTT e como irá autenticar.

Para autorizar um cliente MQTT a acessar objetos IBM WebSphere MQ , autorize o `ClientIdentifier`ou `Username` do cliente ou autorize uma identidade comum do cliente. Para permitir que um cliente se conecte ao IBM WebSphere MQ, autentique `Username` ou use um certificado de cliente. Configure JAAS para autentiou r `Username` e configure SSL para autentiou r um certifiou do de cliente.

Se você configurar `Password` no cliente, criptografe a conexão usando VPN ou configure o canal MQTT para usar SSL para manter a senha particular.

É difícil gerenciar certificados de cliente. Por essa razão, se os riscos associados à autenticação de senha forem aceitáveis, a autenticação de senha será usada na maioria das vezes para autenticar clientes.

Se houver uma maneira segura de gerenciar e armazenar o certificado de cliente, é possível contar com a autenticação de certificado. Porém, raramente os certificados podem ser gerenciados com segurança nos tipos de ambientes em que a telemetria é usada. Em vez disso, a autenticação dos dispositivos usando certificados de cliente é complementada pela autenticação de senhas de clientes no servidor. Devido a essa complexidade adicional, o uso de certifiou dos de cliente é restrito a apliou tivos altamente sensíveis. O uso de duas formas de autentiou ção é chamado de autentiou ção de dois fatores. Deve-se conhecer um dos fatores, como uma senha, e ter o outro, como um certifiou do.

Em um aplicativo altamente sensível, como um dispositivo chip-and-pin, o dispositivo é bloqueado durante a fabricação para evitar violação com hardware e software internos. Um certificado de cliente confiável com tempo limitado é copiado no dispositivo. O dispositivo é implementado no loou l onde deve ser usado. Outras autentiou ções são feitas ou da vez que o dispositivo é usado, por meio de senha ou de outro certifiou do de smart ou rd.

Identidade e Autorização de Cliente MQTT

Use `ClientIdentifier`, `Username` ou uma identidade de cliente comum para obter autorização para acessar objetos do WebSphere MQ.

O administrador do IBM WebSphere MQ tem três opções para selecionar a identidade do canal MQTT. O administrador faz a escolha ao definir ou modificar o canal MQTT usado pelo cliente. A identidade é usada para autorizar o acesso aos tópicos do IBM WebSphere MQ. As opções são:

1. O identificador de cliente.
2. Uma identidade que o administrador fornece para o canal.
3. O `Username` passado do cliente MQTT.

Username é um atributo da classe MqttConnectOptions. Ele deve ser configurado antes de o cliente se conectar ao serviço. Seu valor padrão é nulo.

Use o comando IBM WebSphere MQ **setmqaut** para selecionar quais objetos e quais ações estão autorizados a serem usados pela identidade associada ao canal MQTT. Por exemplo, para autorizar uma identidade de canal, MQTTClient, fornecida pelo administrador do gerenciador de filas, 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
```

Informações relacionadas

Autorizando clientes MQTT a acessar objetos do WebSphere MQ

Autenticação de Cliente MQTT Usando uma Senha

Autentique Username usando a senha do cliente. É possível autenticar o cliente usando uma identidade diferente da usada para autorizar o cliente para publicar e assinar tópicos.

O serviço de telemetria (MQXR) usa JAAS para autenticar o cliente Username. JAAS usa o Password fornecido pelo cliente MQTT.

O administrador do IBM WebSphere MQ decide se deseja autenticar o Usernameou não autenticar, configurando o canal MQTT ao qual um cliente se conecta. Clientes podem ser designados a diferentes canais, e cada canal pode ser configurado para autenticar seus clientes de maneiras diferentes. Usando JAAS, é possível configurar quais métodos devem autenticar o cliente e quais podem opcionalmente autenticar o cliente.

A opção de identidade para autenticação não afeta a escolha da identidade para autorização. É possível querer configurar uma identidade comum para autorização para comodidade administrativa, mas autenticar cada usuário para usar essa identidade. O procedimento a seguir descreve as etapas para autenticar usuários individuais para usarem uma identidade comum:

1. O administrador do IBM WebSphere MQ configura a identidade do canal MQTT para qualquer nome, como MQTTClientUser, usando o IBM WebSphere MQ Explorer.
2. O administrador do IBM WebSphere MQ autoriza o MQTTClient a publicar e assinar qualquer tópico:

   ```
   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. O desenvolvedor do aplicativo cliente MQTT cria um objeto MqttConnectOptions e configurará Username e Password antes de se conectar ao servidor.
4. O desenvolvedor de segurança cria um LoginModule JAAS para autenticar Username com Password e incluí-lo no arquivo de configuração JAAS.
5. O administrador do IBM WebSphere MQ configura o canal MQTT para autenticar o UserName do cliente usando JAAS.

Autenticação de Cliente MQTT Usando SSL

Conexões entre o cliente MQTT e o gerenciador de filas são sempre iniciadas pelo cliente MQTT. O cliente MQTT sempre o cliente SSL. A autenticação de cliente para o servidor e a autenticação de servidor para o cliente MQTT são opcionais.

Ao fornecer ao cliente um certificado digital assinado privado, é possível autenticar o cliente MQTT no IBM WebSphere MQ. O Administrador do IBM WebSphere MQ pode forçar os clientes MQTT a se autenticarem no gerenciador de filas usando SSL. Só é possível solicitar autenticação de cliente como parte da autenticação mútua.

Como uma alternativa para o uso de SSL, alguns tipos de Virtual Private Network (VPN), como IPsec, autenticam os terminais de uma conexão TCP/IP. A VPN criptografa cada pacote IP que flui pela rede. Após uma conexão VPN ser estabelecida, você estabeleceu uma rede confiável. É possível conectar clientes MQTT a canais de telemetria usando TCP/IP sobre a rede VPN.

A autenticação de cliente usando SSL acredita que o cliente tenha um segredo. O segredo é a chave privada do cliente no caso de um certificado autoassinado ou uma chave fornecida por uma autoridade de certificação. A chave é usada para assinar o certificado digital do cliente. Qualquer pessoa em posse de uma chave pública correspondente pode verificar o certificado digital. Certificados podem ser confiáveis ou, se estiverem em cadeia, rastreados de volta por uma cadeia de certificados para um certificado de raiz confiável. A verificação de cliente envia todos os certificados na cadeia de certificados fornecida pelo cliente para o servidor. O servidor verifica a cadeia de certificados até localizar um certificado de sua confiança. O certificado confiável é o certificado público gerado a partir de um certificado autoassinado, ou um certificado raiz normalmente emitido por uma autoridade de certificação. Como etapa final opcional, o certificado confiável pode ser comparado com uma lista de revogação de certificado de "produção".

O certificado confiável pode ter sido emitido por uma autoridade de certificação e já estar incluído no armazenamento de certificados JRE. Ele pode ser um certificado autoassinado ou qualquer certificado que tenha sido incluído no keystore de canal de telemetria como um certificado confiável.

Nota: O canal de telemetria tem um keystore/armazenamento confiável combinados que retêm chaves privadas para um ou mais canais de telemetria e quaisquer certificados públicos necessários para autenticar clientes. Como um canal SSL deve ter um keystore, e esse é o mesmo arquivo que do armazenamento confiável de canal, o armazenamento de certificados JRE nunca é usado como referência. A implicação é que se a autenticação de um cliente exigir um certificado raiz de CA, você deverá colocar o certificado raiz no keystore para o canal, mesmo se esse certificado raiz de CA já estiver no armazenamento de certificados JRE. O armazenamento de certificados JRE nunca é usado como referência.

Pense nas ameaças que a autenticação de cliente deverá contar e nas funções que o cliente e o servidor desempenham na contagem de ameaças. A autenticação de um certificado de cliente sozinha é insuficiente para evitar acesso não autorizado a um sistema. Se alguma outra pessoa tiver retido o dispositivo do cliente, o dispositivo do cliente não estará agindo necessariamente com a autoridade do portador do certificado. Nunca conte com apenas uma defesa contra ataques indesejados. Use pelo menos uma abordagem de autenticação de dois fatores e tenha um suplemento de certificado com conhecimento de informações particulares. Por exemplo, use JAAS e autentique o cliente usando uma senha emitida pelo servidor.

A principal ameaça para o certificado de cliente é cair nãos mãos erradas. O certificado fica retido em um keystore protegido por senha no cliente. Como ele é colocado no keystore? Como o cliente MQTT consegue a senha para o keystore? Até que ponto a proteção da senha é segura? Os dispositivos de telemetria costumam ser fáceis de remover e, por isso, podem ser hackeados em particular. O hardware de dispositivo deve ser à prova de violação? A distribuição e a proteção de certificados do lado do cliente são reconhecidas como difíceis; elas são chamadas de problema de gerenciamento de chaves.

Uma segunda ameaça é o mau uso do dispositivo para acessar servidores de maneiras indesejadas. Por exemplo, se o aplicativo MQTT estiver corrompido, talvez seja possível usar um ponto fraco da configuração do servidor usando a identidade do cliente autenticado.

Para autenticar um cliente MQTT usando SSL, configure o canal de telemetria e o cliente.

-
-

Configuração do cliente MQTT para autenticação de cliente usando o SSL

Para autenticar o cliente MQTT usando SSL, o cliente se conecta a um canal de telemetria usando SSL. Ele deve especificar uma porta TCP que corresponda a um canal de telemetria configurado para autenticar clientes SSL.

Por exemplo, no cliente:

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

A JVM do cliente deve usar o socket factory padrão do JSSE. Se você estiver usando Java ME, deverá assegurar que o pacote JSSE seja carregado. Se você estiver usando Java SE, JSSE foi incluído com o JRE desde a versão Java 1.4.1.

A conexão SSL requer que inúmeras propriedades SSL sejam configuradas antes da conexão. É possível configurar as propriedades transmitindo-as para a JVM usando o comutador `-D` ou configurar as propriedades usando o método `MqttConnectionOptions.setSSLProperties`.

Se você carregar um socket factory não padrão, chamando o método `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, então a maneira como as configurações SSL são transmitidas para o soquete de rede é definida pelo aplicativo.

Inclua os certificados digitais do cliente, assinados usando a chave privada do cliente ou por uma CA, no keystore protegido por senha no cliente. Se o certificado tiver uma cadeia de chaves, será possível incluir os certificados da cadeia de chaves no armazenamento. Quando o servidor verifica o certificado de cliente, ele usa os certificados enviados pelo cliente para corresponderem aos certificados em seu keystore. Ele procura a primeira correspondência na cadeia de chaves com um certificado que possui. O restante da cadeia de chaves é ignorado.

O cliente MQTT envia todos os certificados em seu keystore para o servidor. Se o servidor autenticar alguma das cadeias de chaves enviadas pelo cliente, o cliente será autenticado.

Também é possível usar conjuntos de criptografia SSL para autenticação de cliente.Aqui está uma lista alfabética dos conjuntos de cifras SSL que são atualmente suportados:

- `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 você planeja usar conjuntos de cifras SHA-2 , consulte "Requisitos do sistema para usar conjuntos de cifras SHA-2 com clientes do MQTT" na página 174.

Conceitos relacionados

"Configuração do cliente MQTT para autenticação de canal usando o SSL" na página 108
Para autenticar o canal de telemetria usando SSL, o cliente deve se conectar ao canal de telemetria usando SSL. Ele deve especificar uma porta que corresponda a um canal de telemetria configurado para SSL. A configuração deve incluir um keystore protegido por passphrase contendo o certificado digital assinado em particular do servidor.

Autenticação de Canal de Telemetria Usando SSL

Conexões entre o cliente MQTT e o gerenciador de filas são sempre iniciadas pelo cliente MQTT. O cliente MQTT sempre o cliente SSL. A autenticação de cliente para o servidor e a autenticação de servidor para o cliente MQTT são opcionais.

O cliente sempre tenta autenticar o servidor, a menos que o cliente esteja configurado para usar um CipherSpec que suporte conexão anônima. Se a autenticação falhar, a conexão não será estabelecida.

Como uma alternativa para o uso de SSL, alguns tipos de Virtual Private Network (VPN), como IPsec, autenticam os terminais de uma conexão TCP/IP. A VPN criptografa cada pacote IP que flui pela rede. Após uma conexão VPN ser estabelecida, você estabeleceu uma rede confiável. É possível conectar clientes MQTT a canais de telemetria usando TCP/IP sobre a rede VPN.

A autenticação de servidor usando SSL autentica o servidor para o qual você está prestes a enviar informações confidenciais. O cliente executa as verificações correspondentes aos certificados enviados do servidor em relação aos certificados colocados em seu armazenamento confiável ou em seu armazenamento `cacerts` JRE.

O armazenamento de certificados JRE é um arquivo JKS `cacerts`. Ele está localizado em `JRE InstallPath\lib\security\`. Ele é instalado com a senha padrão `changeit`. É possível armazenar certificados de sua confiança no armazenamento de certificados JRE ou no armazenamento confiável do cliente. Não é possível usar os dois armazenamentos. Use o armazenamento confiável do cliente se você deseja manter os certificados públicos nos quais o cliente confia separados dos certificados que são usados por outros aplicativos Java. Use o armazenamento de certificados JRE se você deseja usar um armazenamento de certificados comum para todos os aplicativos Java em execução no cliente. Se você decidir usar o armazenamento de certificados JRE, revise os certificados que ele contém para se certificar de que confia neles.

É possível modificar a configuração JSSE fornecendo um provedor de confiança diferente. É possível customizar um provedor de confiança para executar diferentes verificações em um certificado. Em alguns ambientes OGSi que usaram o cliente MQTT, o ambiente fornece um provedor de confiança diferente.

Para autenticar o canal de telemetria usando SSL, configure o servidor e o cliente.

•

Conceitos relacionados

"Configuração do cliente MQTT para autenticação de canal usando o SSL" na página 108
Para autenticar o canal de telemetria usando SSL, o cliente deve se conectar ao canal de telemetria usando SSL. Ele deve especificar uma porta que corresponda a um canal de telemetria configurado para SSL. A configuração deve incluir um keystore protegido por passphrase contendo o certificado digital assinado em particular do servidor.

Configuração do cliente MQTT para autenticação de canal usando o SSL

Para autenticar o canal de telemetria usando SSL, o cliente deve se conectar ao canal de telemetria usando SSL. Ele deve especificar uma porta que corresponda a um canal de telemetria configurado para SSL. A configuração deve incluir um keystore protegido por passphrase contendo o certificado digital assinado em particular do servidor.

Por exemplo, no cliente:

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

A JVM do cliente deve usar o socket factory padrão do JSSE. Se você estiver usando Java ME, deverá assegurar que o pacote JSSE seja carregado. Se você estiver usando Java SE, JSSE foi incluído com o JRE desde a versão Java 1.4.1.

A conexão SSL requer que inúmeras propriedades SSL sejam configuradas antes da conexão. É possível configurar as propriedades transmitindo-as para a JVM usando o comutador `-D` ou configurar as propriedades usando o método `MqttConnectionOptions.setSSLProperties` .

Se você carregar um socket factory não padrão, chamando o método `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, então a maneira como as configurações SSL são transmitidas para o soquete de rede é definida pelo aplicativo.

Codifique o cliente para se conectar ao canal de telemetria usando SSL e configure o cliente para confiar em um certificado do servidor de uma destas três formas:

Usando um certificado do servidor assinado por uma autoridade de certificação conhecida no armazenamento `cacerts`.

Nenhuma configuração adicional se o servidor enviar todas as chaves intermediárias na cadeia de certificados. Você será avisado para revisar os certificados no armazenamento `cacerts` do JRE do cliente e alterar a senha para o armazenamento `cacerts`

Outros certificados

Armazene os certificados nos quais confia no armazenamento confiável do cliente. Deve-se armazenar pelo menos um dos certificados na cadeia de certificados no armazenamento confiável. Configure os parâmetros de armazenamento confiável em `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Usando um gerenciador de confiança customizado

Implemente um provedor de confiança e passe para ele o nome do algoritmo usado. Configure o nome da classe do provedor e o algoritmo que deve ser usado em `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStoreProvider`

- com.ibm.ssl.trustStoreManager

Também é possível usar conjuntos de criptografia SSL para autenticação de canal.Aqui está uma lista alfabética dos conjuntos de cifras SSL que são atualmente suportados:

- 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

V 7.5.0.2 Se você planeja usar conjuntos de cifras SHA-2 , consulte “Requisitos do sistema para usar conjuntos de cifras SHA-2 com clientes do MQTT” na página 174.

Conceitos relacionados

“Configuração do cliente MQTT para autenticação de cliente usando o SSL” na página 105
Para autenticar o cliente MQTT usando SSL, o cliente se conecta a um canal de telemetria usando SSL. Ele deve especificar uma porta TCP que corresponda a um canal de telemetria configurado para autenticar clientes SSL.

Privacidade de publicação em canais de telemetria

A privacidade de publicações MQTT enviadas em qualquer direção em canais de telemetria é protegida usando o SSL para criptografar transmissões através da conexão.

Os clientes MQTT que se conectam aos canais de telemetria usam o SSL para proteger a privacidade de publicações transmitidas no canal usando a criptografia de chave simétrica. Como os terminais não são autenticados, não é possível confiar somente na criptografia do canal. Combine a proteção da privacidade com a autenticação mútua ou de servidor.

Como uma alternativa para o uso de SSL, alguns tipos de Virtual Private Network (VPN), como IPsec, autenticam os terminais de uma conexão TCP/IP. A VPN criptografa cada pacote IP que flui pela rede. Após uma conexão VPN ser estabelecida, você estabeleceu uma rede confiável. É possível conectar clientes MQTT a canais de telemetria usando TCP/IP sobre a rede VPN.

Para uma configuração típica, que criptografa o canal e autentica o servidor, consulte “Autenticação de Canal de Telemetria Usando SSL” na página 107.

Criptografar conexões SSL sem autenticar o servidor expõe a conexão com ataques man-in-the-middle. Embora as informações que você troca estejam protegidas contra interceptação, você não sabe com quem está trocando-as. A menos que você controle a rede, você está exposto a alguém que intercepte suas transmissões de IP e se disfarce como o terminal.

É possível criar uma conexão SSL criptografada, sem autenticar o servidor, utilizando uma CipherSpec de troca de chave Diffie-Hellman anônimo que suporta SSL. O segredo principal, compartilhado entre o cliente e o servidor e utilizado para criptografar transmissões SSL é estabelecido sem a troca de um certificado do servidor assinado em particular.

Como as conexões anônimas são precárias, a maioria das implementações SSL não assumem como padrão o uso de CipherSpecs anônimo. Se uma solicitação de cliente para conexão SSL é aceita por um canal de telemetria, o canal deve ter um keystore protegido por uma passphrase. Por padrão, desde que as implementações de SSL não usem CipherSpecs anônimo, o keystore deve conter um certificado assinado em particular que o cliente possa autenticar.

Se você usar CipherSpecs anônimo, o armazenamento de chaves do servidor deve existir, mas ele não precisa conter quaisquer certificados assinados particularmente.

Outra maneira de estabelecer uma conexão criptografada é substituir o provedor de confiança no cliente por sua própria implementação. O provedor de confiança não autenticaria o certificado do servidor, mas a conexão seria criptografada.

Configuração de SSL dos Clientes MQTT e Canais de Telemetria

Os clientes MQTT e o serviço WebSphere MQ Telemetry (MQXR) usam Java Secure Socket Extension (JSSE) para conectar canais de telemetria usando SSL. Os clientes MQTT C e o daemon Telemetry do WebSphere MQ para dispositivos não suportam SSL.

Configure SSL para autenticar o canal de telemetria e o cliente MQTT e criptografar a transferência de mensagens entre clientes e o canal de telemetria.

Como uma alternativa para o uso de SSL, alguns tipos de Virtual Private Network (VPN), como IPsec, autenticam os terminais de uma conexão TCP/IP. A VPN criptografa cada pacote IP que flui pela rede. Após uma conexão VPN ser estabelecida, você estabeleceu uma rede confiável. É possível conectar clientes MQTT a canais de telemetria usando TCP/IP sobre a rede VPN.

É possível configurar a conexão entre um cliente do MQTT Java e um canal de telemetria para usar o protocolo SSL sobre TCP/IP. O que será assegurado depende de como você configura SSL para usar JSSE. Começando com a configuração mais segura, é possível configurar três níveis diferentes de segurança:

1. Permita que apenas clientes MQTT confiáveis se conectem. Conecte um cliente MQTT apenas a um canal de telemetria confiável. Criptografe mensagens entre o cliente e o gerenciador de filas; consulte "Autenticação de Cliente MQTT Usando SSL" na página 104.
2. Conecte um cliente MQTT apenas a um canal de telemetria confiável. Criptografe mensagens entre o cliente e o gerenciador de filas; consulte "Autenticação de Canal de Telemetria Usando SSL" na página 107.
3. Criptografe mensagens entre o cliente e o gerenciador de filas; consulte "Privacidade de publicação em canais de telemetria" na página 110.

Parâmetros de Configuração de JSSE

Modifique os parâmetros de JSSE para alterar a maneira como a conexão SSL é configurada. Os parâmetros de configuração de JSSE são organizados em três conjuntos:

1. IBM WebSphere MQ Canal de telemetria
2. Cliente MQTT Java
3. JRE

Configure os parâmetros de canal de telemetria usando IBM WebSphere MQ Explorer. Configure os parâmetros do MQTT Java Client no atributo `MqttConnectionOptions.SSLProperties` Modifique os parâmetros de segurança do JRE editando arquivos no diretório de segurança do JRE no cliente e no servidor.

IBM WebSphere MQ Canal de telemetria

Configure todos os parâmetros SSL do canal de telemetria usando WebSphere MQ Explorer.

`ChannelName`

`ChannelName` é um parâmetro necessário em todos os canais.

O nome do canal identifica o canal associado a um determinado número de porta. Nomeie canais para ajudá-lo a administrar conjuntos de clientes MQTT.

`PortNumber`

`PortNumber` é um parâmetro opcional em todos os canais. Ele é padronizado como 1883 para canais TCP e 8883 para canais SSL.

O número da porta TCP/IP associado a esse canal. Clientes MQTT são conectados a um canal especificando a porta definida para o canal. Se o canal tiver propriedades SSL, o cliente deverá se conectar usando o protocolo SSL; por exemplo:

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

`KeyFileName`

`KeyFileName` é um parâmetro necessário para canais SSL. Ele deve ser omitido para canais TCP.

`KeyFileNome` é o caminho para o Java keystore contendo certificados digitais que você fornece. Use JKS, JCEKS ou PKCS12 como tipo de keystore no servidor.

Identifique o tipo de keystore usando uma das seguintes extensões de arquivo:

`.jks`

```
.jceks
.p12
.pkcs12
```

Um keystore com qualquer outra extensão de arquivo é assumido como um keystore JKS.

É possível combinar um tipo de keystore no servidor com outros tipos de keystore no cliente.

Coloque o certificado particular do servidor no keystore. O certificado é conhecido como o certificado do servidor. O certificado pode ser autoassinado ou fazer parte de uma sequência de certificados que é assinada por uma autoridade de assinatura.

Se você estiver usando uma sequência de certificados, coloque os certificados associados no keystore do servidor.

O certificado do servidor, e quaisquer certificados na ou de certificados, é enviado para os clientes autenticarem a identidade do servidor.

Se você tiver configurado `ClientAuth` como `Required`, o keystore deverá conter quaisquer certificados necessários para autenticar o cliente. O cliente envia um certificado autoassinado ou uma sequência de certificados e o cliente é autenticado pela primeira verificação desse material a um certificado no keystore. Usando uma sequência de certificados, um certificado pode verificar muitos clientes, mesmo se eles forem emitidos com diferentes certificados de cliente.

PassPhrase

`PassPhrase` é um parâmetro necessário para canais SSL. Ele deve ser omitido para canais TCP.

O passphrase é usado para proteger o keystore.

ClientAuth

`ClientAuth` é um parâmetro SSL opcional. Ele é padronizado para não autenticação de cliente. Ele deve ser omitido para canais TCP.

Configure `ClientAuth` se quiser que o serviço de telemetria (MQXR) autentique o cliente antes de permitir que o cliente se conecte ao canal de telemetria.

Se você configurar `ClientAuth`, o cliente deverá se conectar ao servidor usando SSL e autenticar o servidor. Em resposta à configuração de `ClientAuth`, o cliente envia seu certificado digital para o servidor e quaisquer outros certificados em seu keystore. Seu certificado digital é conhecido como certificado de cliente. Esses certificados são autenticados com relação aqueles retidos no keystore do canal e no armazenamento `cacerts` do JRE.

CipherSuite

`CipherSuite` é um parâmetro SSL opcional. Ele é padronizado para tentar todos os CipherSpecs ativados. Ele deve ser omitido para canais TCP.

Se quiser usar um determinado CipherSpec, configure `CipherSuite` para o nome do CipherSpec que deve ser usado para estabelecer a conexão SSL.

O serviço de telemetria e o cliente MQTT negociam um CipherSpec comum de todos os CipherSpecs ativados em cada extremidade. Se um CipherSpec específico for especificado em ambas as extremidades da conexão, ele deverá corresponder ao CipherSpec na outra extremidade.

Instale cifras adicionais incluindo provedores adicionais no JSSE.

Federal Information Processing Standards (FIPS)

FIPS é uma configuração opcional. Por padrão, ela não é configurada.

No painel de propriedades do gerenciador de filas ou usando o comando **runmqsc**, configure SSLFIPS. SSLFIPS especifica se apenas algoritmos certificados por FIPS devem ser usados.

Lista de nomes de revogação

A lista de nomes de revogação é uma configuração opcional. Por padrão, ela não é configurada.

No painel de propriedades do gerenciador de filas ou usando o comando **runmqsc**, configure SSLCRLNL. SSLCRLNL especifica uma lista de nomes de objetos de informações sobre autenticação que são usados para fornecer locais de revogação de certificado.

Nenhum outro parâmetro de gerenciador de filas que configura propriedades de SSL é usado.

cliente MQTT Java

Configure as propriedades SSL para o cliente Java em `MqttConnectionOptions.SSLProperties`; por exemplo:

```
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);
```

Os nomes e valores das propriedades específicas são descritos na documentação da API para `MqttConnectOptions`.Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

Protocolo

`Protocolo` é opcional.

O protocolo é selecionado na negociação com o servidor de telemetria. Se você requerer um protocolo específico, é possível selecionar um. Se o servidor de telemetria não suportar o protocolo, a conexão falhará.

ContextProvider

`ContextProvider` é opcional.

KeyStore

`KeyStore` é opcional. Configure-o se `ClientAuth` estiver configurado no servidor para forçar a autenticação do cliente.

Coloque o certificado digital do cliente, assinado com o uso de sua chave privada, no keystore. Especifique a senha e o caminho do keystore. O tipo e o provedor são opcionais. JKS é o tipo padrão e IBMJCE é o provedor padrão.

Especifique um provedor de keystore diferente para fazer referência a uma classe que inclui um novo provedor de keystore. Passe o nome do algoritmo usado pelo provedor de keystore para instanciar `KeyManagerFactory` configurando o nome do gerenciador de chave.

TrustStore

`TrustStore` é opcional. É possível colocar todos os certificados nos quais você confia no armazenamento `cacerts` do JRE.

Configure o armazenamento confiável se quiser ter um armazenamento confiável diferente para o cliente. É possível não configurar o armazenamento confiável se o servidor estiver usando um certificado emitido por uma CA conhecida que já tem seu certificado raiz armazenado em `cacerts`.

Inclua o certificado assinado publicamente do servidor ou o certificado raiz do armazenamento confiável e especifique o caminho e a senha do armazenamento confiável. JKS é o tipo padrão e IBMJCE é o provedor padrão.

Especifique um provedor de armazenamento confiável diferente para fazer referência a uma classe que inclui um novo provedor de armazenamento confiável. Passe o nome do algoritmo usado pelo provedor de armazenamento confiável para instanciar `TrustManagerFactory` configurando o nome do gerenciador de confiança.

JRE

Outros aspectos da segurança Java que afetam o comportamento do SSL no cliente e no servidor são configurados no JRE. Os arquivos de configuração no Windows estão no *Java Installation*

Directory\jre\lib\security Se você estiver usando o JRE fornecido com o IBM WebSphere MQ, o caminho será igual ao mostrado na seguinte tabela:

Tabela 3. Caminhos de Arquivos por Plataforma para Arquivos de Configuração SSL do JRE	
Plataforma	**Caminho de arquivo..**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux para System x 32 bits	*WMQ Installation Directory*/java/jre/lib/security
Outras plataformas UNIX and Linux	*WMQ Installation Directory*/java/jre64/jre/lib/security

Autoridades de certificação conhecidas

O arquivo `cacerts` contém os certificados raiz de autoridades de certificação conhecidas. O `cacerts` é usado por padrão, a menos que você especifique o armazenamento confiável. Se usar o armazenamento `cacertse` não fornecer um armazenamento confiável, você deverá revisar e editar a lista de assinantes no `cacerts` para atender aos requisitos de segurança.

É possível abrir o `cacerts` usando o comando WebSphere MQ `strmqikm`.que executa o utilitário IBM Key Management Abra o `cacerts` como um arquivo JKS usando a senha `changeit`. Modifique a senha para assegurar o arquivo.

Configurando classes de segurança

Use o arquivo `java.security` para registrar provedores de segurança adicionais e outras propriedades de segurança padrão.

Permissões

Use o arquivo `java.policy` para modificar as permissões concedidas a recursos. `javaws.policy` concede permissões para `javaws.jar`

Grau de Intensidade da Criptografia

Alguns JREs são fornecidos com segurança de criptografia reduzida. Se você não puder importar as chaves para os keystores, a criptografia de força reduzida poderá ser a causa. Tente iniciar o **ikeyman** usando o comando **strmqikm** ou fazer o download de arquivos de jurisdição fortes, mas limitados, em IBM Developer Kits, Informações de Segurança.

Importante: Seu país de origem pode ter restrições quanto à importação, à posse, ao uso ou à reexportação para outro país de um software de criptografia. Antes de fazer download ou usar os arquivos de políticas irrestritas, você deve verificar as leis do seu país. Verifique seus regulamentos e suas políticas com relação à importação, à posse, ao uso e à reexportação do software de criptografia para determinar se essas ações são permitidas.

Modificando o Provedor de Confiança para Permitir que o Cliente se Conecte a qualquer Servidor

O exemplo ilustra como incluir um provedor de confiança e fazer referência a ele a partir do código do cliente MQTT. O exemplo não faz autenticação do cliente ou servidor. A conexão SSL resultante é criptografada sem ser autenticada.

O fragmento de código em Figura 20 na página 114 configura o provedor de confiança e o gerenciador de confiança `AcceptAllProviders` para o cliente MQTT.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

Figura 20. Fragmento de Código do Cliente MQTT

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
                AcceptAllTrustManagerFactory.class.getName());
    }
```

Figura 21. AcceptAllProvider.java

```
    protected static class AcceptAllTrustManagerFactory extends
            javax.net.ssl.TrustManagerFactorySpi {
        public AcceptAllTrustManagerFactory() {}
        protected void engineInit(java.security.KeyStore keystore) {}
        protected void engineInit(
                javax.net.ssl.ManagerFactoryParameters parameters) {}
        protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
            return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
        }
```

Figura 22. AcceptAllTrustManagerFactory.java

```
    protected static class AcceptAllX509TrustManager implements
            javax.net.ssl.X509TrustManager {
        public void checkClientTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Client authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public void checkServerTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Server authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return new java.security.cert.X509Certificate[0];
        }
    private static void report(String string) {
           System.out.println(string);
      }
    }
```

Figura 23. AcceptAllX509TrustManager.java

Configuração JAAS do Canal de Telemetria

Configure JAAS para autenticar o `Username` enviado pelo cliente.

O administrador do WebSphere MQ configura quais canais do MQTT requerem autenticação de cliente usando JAAS Especifique o nome de uma configuração JAAS para cada canal que deve executar a autenticação JAAS. Todos os canais podem usar a mesma configuração JAAS ou podem usar diferentes configurações JAAS. As configurações são definidas no *WMQData directory*`\qmgrs\`*qMgrName*`\mqxr\jaas.config`

O arquivo `jaas.config` é organizado pelo nome da configuração JAAS. Sob cada nome de configuração há uma lista de configurações de login; consulte Figura 24 na página 116.

O JAAS fornece quatro módulos de login padrão. Os módulos de Login padrão NT e UNIX são de valor limitado.

JndiLoginModule
Autentica com relação a um serviço de diretório configurado em JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Autentica usando protocolos Kerberos.

NTLoginModule
Autentica usando informações de segurança de NT para o usuário atual.

UnixLoginModule
Autentica usando as informações de segurança UNIX para o usuário atual.

O problema com o uso de `NTLoginModule` ou `UnixLoginModule` é que o serviço de telemetria (MQXR) é executado com a identidade `mqm` e não com a identidade de canal do MQTT. `mqm` é a identidade passada para `NTLoginModule` ou `UnixLoginModule` para autenticação e não a identidade do cliente.

Para superar esse problema, grave seu próprio módulo de login ou use os outros módulos de login padrão. Uma amostra `JAASLoginModule.java` é fornecida com WebSphere MQ Telemetry. Ele é uma implementação da interface `javax.security.auth.spi.LoginModule`. Use-o para desenvolver seu próprio método de autenticação.

Quaisquer novas classes LoginModule que você fornecer deverão estar no caminho da classe do serviço de telemetria (MQXR). Não coloque suas classes nos diretórios do WebSphere MQ que estão no caminho da classe Crie seus próprios diretórios e defina o caminho da classe inteiro para o serviço de telemetria (MQXR).

É possível aumentar o caminho da classe usado pelo serviço de telemetria (MQXR) configurando o caminho da classe no arquivo `service.env`. `CLASSPATH` deve ser capitalizado e a instrução do caminho da classe só pode conter literais. Não é possível usar variáveis em CLASSPATH; por exemplo, `CLASSPATH=%CLASSPATH%` está incorreto. O serviço de telemetria (MQXR) configura seu próprio caminho de classe. O CLASSPATH definido em `service.env` é incluído nele.

O serviço de telemetria (MQXR) fornece dois retornos de chamada que retornam `Username` e o `Password` para um cliente conectado ao canal do MQTT. O `Nome do Usuário` e a `Senha` são configurados no objeto `MqttConnectOptions` Consulte Figura 25 na página 117 para obter um exemplo de como acessar `Username` e `Password`.

Examples

Exemplo de um Arquivo de Configuração JAAS com uma Configuração Nomeada, `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. Arquivo `jaas.config` de Amostra

Um exemplo de módulo de login JAAS codificado para receber o `Username` e o `Password` fornecidos por um cliente MQTT.

```
public boolean login()
    throws javax.security.auth.login.LoginException {
  javax.security.auth.callback.Callback[] callbacks =
    new javax.security.auth.callback.Callback[2];
  callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
  callbacks[1] = new javax.security.auth.callback.PasswordCallback(
      "PasswordCallback", false);
  try {
    callbackHandler.handle(callbacks);
    String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
        .getName();
    char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
        .getPassword();
    // Accept everything.
    if (true) {
      loggedIn = true;
    } else
      throw new javax.security.auth.login.FailedLoginException("Login failed");

    principal= new JAASPrincipal(username);

  } catch (java.io.IOException exception) {
    throw new javax.security.auth.login.LoginException(exception.toString());
  } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
    throw new javax.security.auth.login.LoginException(exception.toString());
  }

  return loggedIn;
}
```

Figura 25. Método `JAASLoginModule.Login()` de Amostra

Conceitos de programação do cliente

Os conceitos descritos nesta seção o ajudam a entender o cliente do Java para a versão 3.1 do MQTT protocol. Os conceitos complementam a documentação da API que acompanha o pacote `com.ibm.micro.client.mqttv3`.

`com.ibm.micro.client.mqttv3` contém as classes que fornecem os métodos públicos para as implementações Java do protocolo MQTT version 3.1 . O pacote `com.ibm.micro.client.mqttv3` e os pacotes que acompanham que implementam o protocolo para o Java SE e ME, são fornecidos com a instalação do IBM WebSphere MQ Telemetry

Para desenvolver e executar um cliente MQTT, é necessário copiar ou instalar esses pacotes no dispositivo do cliente. Não é necessário instalar um tempo de execução de cliente separado.

As condições de licenciamento para clientes são associadas ao servidor ao qual você está conectando os clientes.

O cliente Java é uma implementação de referência da versão 3.1 do MQTT protocol. É possível implementar seus próprios clientes em diferentes idiomas apropriados para diferentes plataformas de dispositivo. Consulte Formato e protocolo do MQ Telemetry Transport para obter os detalhes.

A documentação da API do cliente para o pacote `com.ibm.micro.client.mqttv3` não faz suposições sobre em qual servidor o cliente está conectado. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..
O comportamento do cliente poderá diferir um pouco quando conectados a diferentes servidores. As descrições que seguem descrevem o comportamento do cliente quando conectados ao serviço (MQXR) de telemetria doIBM WebSphere MQ.

O Cliente de sistema de mensagens do MQTT para JavaScript e os aplicativos da web

Até recentemente, os aplicativos da web de programação e a criação de aplicativos do sistema de mensagens foram disciplinas separadas. Não importa qual foi sua experiência anterior, há vantagens

significativas em usar o JavaScript e o sistema de mensagens juntos. Ao codificar seu aplicativo do sistema de mensagens como um aplicativo da web, ele poderá ser extraído e executado em qualquer navegador atualizado. Se você mudar o aplicativo, a versão mais recente será extraída sempre que o navegador for atualizado. O navegador também procura segurança e a transmissão confiável de mensagens.

Como o uso de um aplicativo da web facilita a implementação do aplicativo

Se você tiver experiência de desenvolvimento e implementação dos aplicativos tradicionais do sistema de mensagens em (por exemplo) IBM WebSphere MQ, você poderá estar familiarizado com o processo de implementação a seguir:

1. O administrador do sistema instala ou integra a biblioteca do cliente.
2. O administrador do sistema se prepara para o aplicativo do sistema de mensagens a serem distribuídos para os usuários finais e instalado em seus sistemas locais.
3. Quando o código for mudado, o administrador do sistema repetirá as etapas anteriores (então o gerenciamento de mudanças é complexo).

Se você codificar seu aplicativo do sistema de mensagens como um aplicativo da web, este é o processo de implementação:

1. O administrador do sistema serve o aplicativo da web e a biblioteca do cliente em uma URL.
2. O navegador do usuário final extrai no aplicativo da web e na biblioteca do cliente juntos.
3. Quando o código for mudado, a versão atualizada será selecionada quando o navegador for atualizado (então o gerenciamento de mudanças é simples).

Por que você talvez deseje usar o sistema de mensagens diretamente do navegador em seus aplicativos da web

Se você tiver experiência com aplicativos de programação em JavaScript, você poderá estar interessado em saber os benefícios fornecidos pelos sistemas de mensagens como IBM WebSphere MQ:

- Se você enviar e receber mensagens através de um sistema de mensagens, esse sistema será responsável por assegurar-se de que as mensagens sejam entregues.
- Como o sistema de mensagens procura a entrega, seu aplicativo da web poderá efetuar "fire and forget". Isso simplifica bastante a lógica de programação. Se as mensagens forem entregues a você, seu código não precisará verificar se eles chegaram lá. Seu aplicativo não precisa mais manipular a confirmação de recebimento ou salvar mensagens não entregues e tentá-los novamente posteriormente.
- Os sistemas de mensagens fornecem mensagens orientadas pelo evento. Seu aplicativo cliente não é mais necessário para enviar uma solicitação, então, continuamente pesquisa uma resposta. Em vez disso, o servidor do sistema de mensagens enviará uma mensagem para seu aplicativo cliente quando um evento interessante ocorrer. Isso também significa que o aplicativo cliente é alertado assim que o evento acontecer, em vez de esperar até a próxima vez que o aplicativo pesquisar o servidor.
- O sistema de mensagens orientado pelo evento também reduz maciçamente o carregamento no dispositivo que hospeda o aplicativo cliente, o tráfego de rede entre o navegador e o servidor do sistema de mensagens e o carregamento no servidor do sistema de mensagens. Isso é cada vez mais importante, como mais e mais sistemas estão em execução em dispositivos móveis e se conectando através de redes wireless.

Como as partes se ajustam

O Cliente de sistema de mensagens do MQTT para JavaScript inclui uma biblioteca do cliente e um aplicativo da web de exemplo que usa a biblioteca. Codifique seu próprio aplicativo da web que usa a biblioteca. O aplicativo da web e a biblioteca do cliente são, então, disponibilizados em sua URL escolhida, por exemplo, por um gerenciador de filas MQ (como no diagrama a seguir) ou por um servidor de aplicativos. O navegador extrai no aplicativo da web, na biblioteca do cliente e no aplicativo da

web, em seguida, usa o navegador para se conectar e trocar mensagens, um servidor MQTT como IBM WebSphere MQ Telemetry ou IBM MessageSight.

Estes são os fluxos:

1. Cada instância do navegador atualiza sua conexão com a URL na qual o aplicativo da web está disponível e uma versão atualizada do aplicativo da web e a biblioteca do cliente é carregada no navegador.
2. O aplicativo da web se conecta a um gerenciador de filas, usando MQTT através do WebSocket protocol e assina um tópico de interesse.
3. O gerenciador de filas usa a mesma conexão para enviar mensagens que correspondem à assinatura de volta para o aplicativo da web.

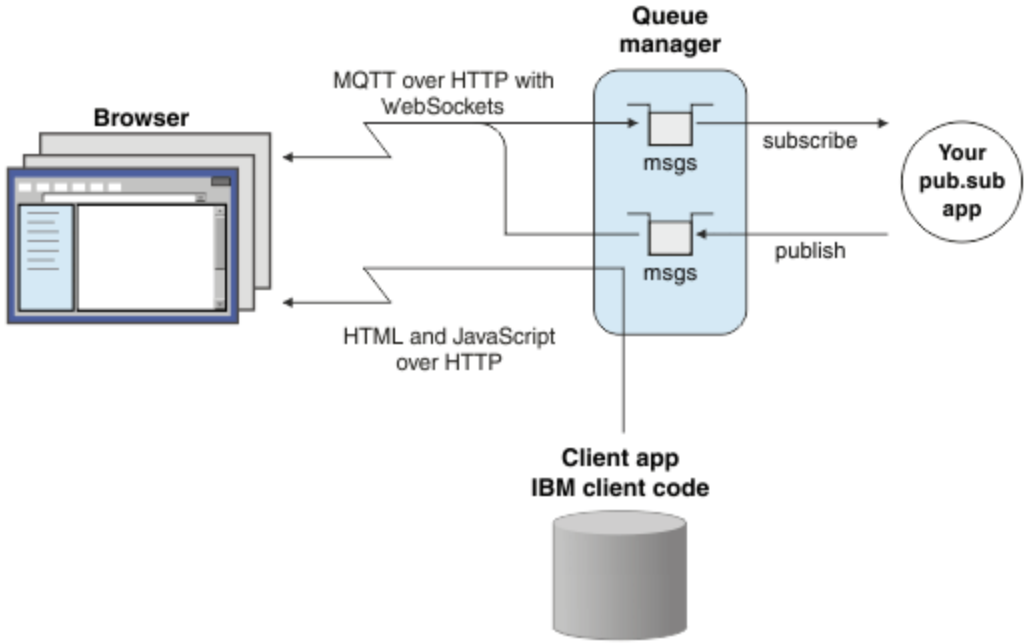


Figura 26. Usando o Cliente de sistema de mensagens do MQTT para JavaScript com o sistema de mensagens da publicação e assinatura

O aplicativo da web contém a lógica do aplicativo e a URL do servidor MQTT. Quando aberto em um navegador, o aplicativo se conectará ao servidor MQTT, criará as assinaturas que necessita, em seguida, aguardará para receber alertas orientadas por eventos e agir sobre elas.

O aplicativo da web se conecta usando MQTT como o protocolo de transporte, em execução através do WebSockets. A maioria dos navegadores modernos pode fazer conexões com o WebSockets. Usando o WebSockets, o aplicativo da web pode transmitir mensagens através de firewalls que aceitam HTTP e o WebSocket protocol e pode enviar pacotes de dados (conhecido como "quadros") tão como usar o TCP através do IP.

Quando uma mensagem enviada pelo aplicativo da web chegar ao servidor MQTT, o aplicativo do lado do servidor apenas irá vê-lo como uma mensagem. Ele não conhece a mensagem que chegou a partir de um navegador.

Administrando e controlando um servidor MQTT

O servidor MQTT manipula a complexidade do lado do servidor do sistema de mensagens. Ele garante a entrega das mensagens que ele recebe do aplicativo da web e ele hospeda os aplicativos da publicação e da assinatura que respondem ao aplicativo da web. Para qualquer servidor MQTT, é necessário concluir as seguintes etapas:

- Crie um servidor.
- Selecione uma porta.
- Defina um novo canal do MQTT.
- Configure seu aplicativo da web do cliente para se conectar à porta escolhida através do novo canal do MQTT.

Também é necessário servir o aplicativo executável da web JavaScript para o navegador. Se você estiver usando o IBM WebSphere MQ Telemetry, por padrão, o servidor MQTT fará isso por você, usando o mesmo canal MQTT que o aplicativo da web usa para se conectar ao servidor MQTT. Se você estiver testando o MQTT, isso poderá ajudá-lo rapidamente no funcionamento adequado e na execução. Para uso de produção, particularmente em ambientes de alto rendimento do processamento, você pode preferir servir o aplicativo executável da web JavaScript em um canal separado, usando um servidor de aplicativos dedicado como WebSphere Application Server.

Nota: Como ele é projetado para ambientes de alto rendimento, o IBM MessageSight espera que você faça isso.

Por exemplo, se você estiver usando IBM WebSphere MQ Telemetry, use o assistente IBM WebSphere MQ Explorer **Novo Canal de Telemetria** para concluir as etapas a seguir:

1. Crie um servidor.
2. Selecione uma porta (1883 por padrão).
3. Definir um novo canal MQTT.
4. Configure seu aplicativo da web do cliente para se conectar à porta escolhida através do novo canal do MQTT.

O aplicativo executável da web JavaScript está (opcionalmente) também servidos por meio do gerenciador de filas no mesmo canal. Para fazer isso, o gerenciador de filas deve suportar ambos MQTT e HTTP. Se você já tiver um gerenciador de filas configurado para MQTT, será possível usar a ferramenta da linha de comandos MQSC para alterar o protocolo na definição de canal para suportar ambos MQTT e HTTP. Consulte ALTER CHANNEL.

O aplicativo da web e a biblioteca do cliente do Cliente de sistema de mensagens do MQTT para JavaScript são armazenados em disco em uma estrutura definida por seu servidor de aplicativos ou gerenciador de filas. Se você estiver usando o IBM WebSphere MQ Telemetry, o aplicativo da web e a biblioteca do cliente serão armazenados na estrutura de diretórios a seguir:

```
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)
```

O aplicativo da web de amostra e a biblioteca do cliente são armazenados no diretório *MQINSTALL*/mqxr/SDK/WebContent. O material neste diretório é atendido por todos os gerenciadores de filas. Se você não desejar que seus usuários vejam e usem todo esse material, será necessário criar sua própria versão do aplicativo. Para fazer este aplicativo ou seu próprio aplicativo de substituição, disponível em gerenciadores de filas específicos, coloque o aplicativo no diretório *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent. Para selecionar o aplicativo e as classes associadas ao JavaScript para servir em

uma URL, o gerenciador de filas procurará primeiro em seu próprio diretório `WebContent`, em seguida, no diretório global `WebContent`. Na árvore de diretórios do exemplo anterior, o gerenciador de filas serve o *your_client_app* e Global Copy das classes do JavaScript.

Para parar o gerenciador de filas de servir os arquivos executáveis do aplicativo da web ou modificar o local em que o gerenciador de filas procurará os arquivos executáveis, configure a propriedade **webcontentpath** e inclua-a no arquivo `mqxr.properties`. Consulte Propriedades MQXR.

Conceitos relacionados

"Como programar os aplicativos do sistema de mensagens no JavaScript" na página 121

Tarefas relacionadas

"Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets" na página 78

Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

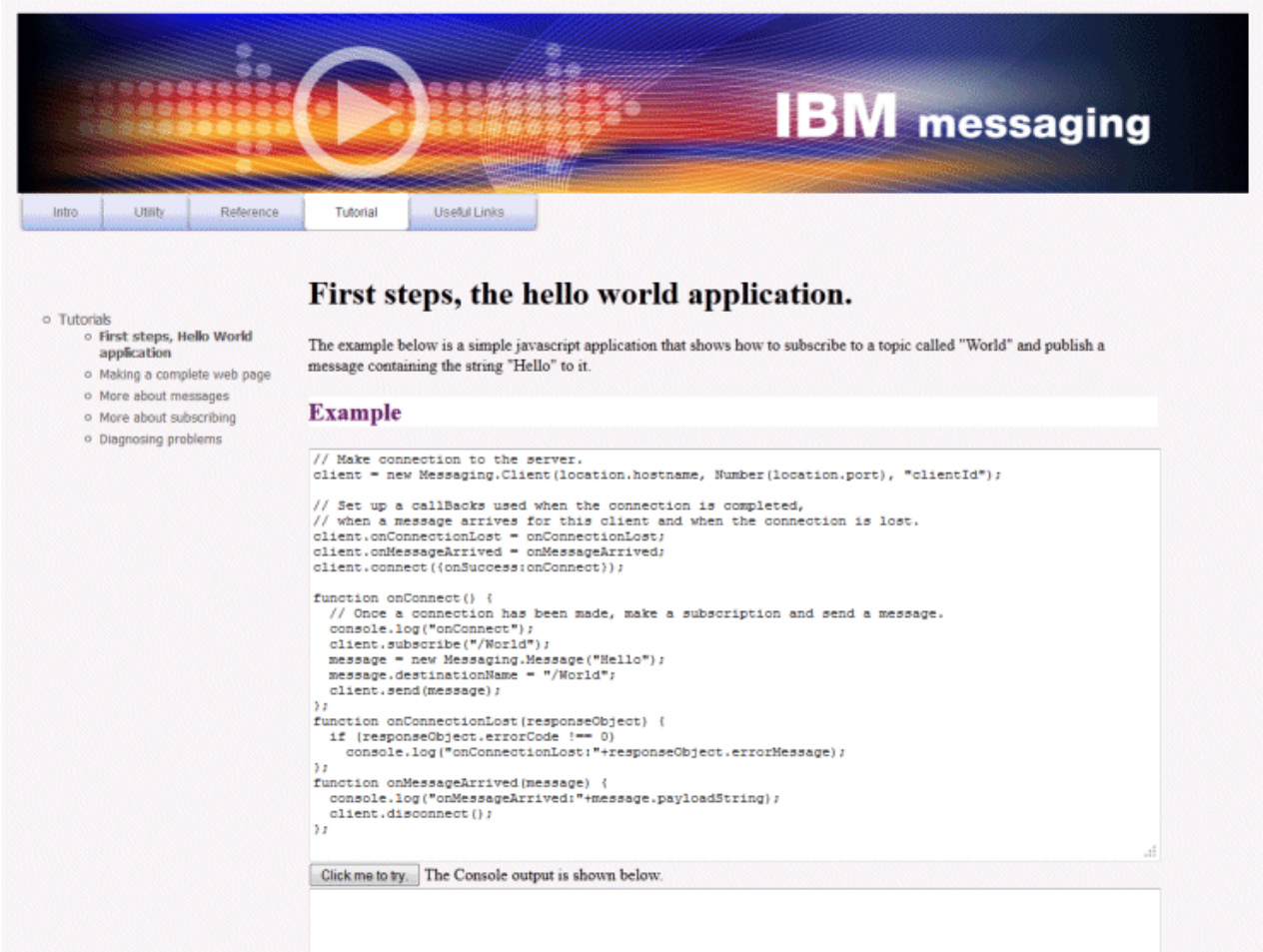
"Introdução ao Cliente de sistema de mensagens do MQTT para JavaScript" na página 24

É possível iniciar com o Cliente de sistema de mensagens do MQTT para JavaScript, exibindo a página inicial de amostra do cliente com o sistema de mensagens e procurando os recursos aos quais ele se vincula. Para exibir esta página inicial, configure um servidor MQTT para aceitar conexões do Páginas JavaScript de amostra do cliente de sistema de mensagens MQTT, em seguida, digite a URL configurada no servidor em um navegador da web. O Cliente de sistema de mensagens do MQTT para JavaScript inicia automaticamente em seu dispositivo e a página inicial de amostra do cliente com o sistema de mensagens é exibida. Esta página contém links para utilitários, documentação da interface de programação, um tutorial e outras informações úteis.

Como programar os aplicativos do sistema de mensagens no JavaScript

O Cliente de sistema de mensagens do MQTT para JavaScript inclui um tutorial que demonstra como criar um aplicativo da web simples de publicação e assinatura. Ao explorar o código de aplicativo "First Steps, Hello world", será possível obter um entendimento básico dos mecanismos de programação de aplicativos da web para o sistema de mensagens.

Se a sua experiência até o momento tem sido sobretudo no desenvolvimento e na implementação de aplicativos do sistema de mensagens tradicional, você também poderá localizar a seção útil do "Dicas de codificação do JavaScript" na página 122. Se você for um desenvolvedor experiente do JavaScript que é nova para o sistema de mensagens, você encontrará uma breve introdução aos conceitos do sistema de mensagens de chave na seção "Informações básicas do sistema de mensagens" na página 124.



Dicas de codificação do JavaScript

Se você estiver usado para o desenvolvimento de aplicativos do sistema de mensagens, mas novas para aplicativos da web, você pode encontrar os seguintes dicas úteis:

Agrupamento do código para cada evento em um retorno de chamada onSuccess

Ao codificar um aplicativo do sistema de mensagens, você codifica os seguintes eventos na seguinte ordem:

1. connect
2. assinar
3. publicação
4. mensagem recebida

A API do Cliente de sistema de mensagens do MQTT para JavaScript é totalmente assíncrona, o que significa que seu encadeamento de aplicativo não será bloqueado enquanto espera por chamadas como conectar ou assinar para entrar em vigor. Em vez disso, essas chamadas sinalizam a conclusão chamando um retorno de chamada `onSuccess` ou `onFailure`. Para ter certeza de que cada evento foi concluído antes de o próximo evento ser acionado, será necessário agrupar o código para cada evento em um retorno de chamada `onSuccess`. Por exemplo, o aplicativo JavaScript poderá retornar de fazer a chamada de conexão antes de a conexão ter sido criada. Para ter certeza de que a conexão aconteceu antes da assinatura, é necessário colocar o código de assinatura em um retorno de chamada `onSuccess` na conexão.

O código de aplicativo "First Steps, Hello World" usa essa abordagem.

Integrando o código do aplicativo na marcação HTML

Aqui está uma página de exemplo do JavaScript:

Aqui está a origem para a página anterior, para mostrar como o código do aplicativo está integrado dentro de marcação 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>
```

Informações básicas do sistema de mensagens

Aqui estão algumas informações do sistema de mensagens do segundo plano para desenvolvedores de aplicativo da web que são novas para o sistema de mensagens:

Sistema de mensagens assíncrono e fire-and-forget.

O protocolo MQTT suporta entrega garantida e transferências fire-and-forget. No protocolo, a entrega da mensagem é assíncrona: o aplicativo transmite a mensagem para a API do cliente e não toma nenhuma ação adicional para assegurar-se de que a mensagem seja entregue. Esta abordagem é conhecida como *fire-and-forget*. Quando uma resposta estiver disponível, ela será automaticamente enviada ao aplicativo.

A entrega assíncrona libera o aplicativo de qualquer conexão do servidor e da espera por mensagens. O modelo de interação é como e-mail, mas otimizado para a programação de aplicativos.

Consulte também a seção do "protocolo MQTT" do “Introdução ao MQTT” na página 5

Uma visão geral do sistema de mensagens de publicação e assinatura.

O provedor de informações é chamado de *publicador*. Um publicador fornece informações sobre um assunto, sem precisar saber nada sobre os aplicativos interessados nessas informações. Um publicador escolhe um *tópico*, que é um contêiner para mensagens em um assunto específico. O publicador gera, então, cada parte das informações para esse assunto como uma mensagem, denominada uma *publicação* e a posta no tópico associado.

O consumidor das informações é denominado um *assinante*. O assinante cria uma *assinatura* para um tópico ao qual está interessado. Quando uma nova mensagem for postada para o tópico, a mensagem

será encaminhada para todos os assinantes do tópico. Os assinantes podem fazer várias assinaturas e podem receber informações de vários publicadores diferentes.

Consulte também Introdução ao IBM WebSphere MQ sistema de mensagens de publicação / assinatura

Como assinaturas e tópicos correspondem.

Se você estiver usando IBM WebSphere MQ como seu servidor MQTT, será necessário entender como o IBM WebSphere MQ especifica tópicos. No IBM WebSphere MQ, um publicador cria uma mensagem e a publica com uma sequência de tópicos que melhor se ajuste ao assunto da publicação. Para receber publicações, um assinante cria uma assinatura com uma sequência de tópicos com correspondência de padrões para selecionar tópicos de publicação. O gerenciador de filas entrega as publicações para os assinantes que têm assinaturas correspondentes ao tópico da publicação e que estão autorizados a receber as publicações.

Normalmente os assuntos são organizados hierarquicamente, em árvores de tópicos, usando o caractere '/' para criar subtópicos na sequência de tópicos. Tópicos são nós na árvore de tópicos. Tópicos podem ser nós folha sem subtópicos ou nós intermediários com subtópicos. Os assinantes podem usar curingas para assinar mais de um tópico por vez. Por exemplo, uma assinatura para `/sport/tennis` apenas obtém mensagens postadas para o subtópico tênis, enquanto que uma assinatura para `/sport/#` recebe mensagens postadas para qualquer subtópico de `/sport`.

Consulte também Tópicos, Árvores de tópicos e Esquemas de curinga.

Conceitos relacionados

"O Cliente de sistema de mensagens do MQTT para JavaScript e os aplicativos da web" na página 117

Tarefas relacionadas

"Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets" na página 78

Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

"Introdução ao Cliente de sistema de mensagens do MQTT para JavaScript" na página 24

É possível iniciar com o Cliente de sistema de mensagens do MQTT para JavaScript, exibindo a página inicial de amostra do cliente com o sistema de mensagens e procurando os recursos aos quais ele se vincula. Para exibir esta página inicial, configure um servidor MQTT para aceitar conexões do Páginas JavaScript de amostra do cliente de sistema de mensagens MQTT, em seguida, digite a URL configurada no servidor em um navegador da web. O Cliente de sistema de mensagens do MQTT para JavaScript inicia automaticamente em seu dispositivo e a página inicial de amostra do cliente com o sistema de mensagens é exibida. Esta página contém links para utilitários, documentação da interface de programação, um tutorial e outras informações úteis.

Retornos de chamada e sincronização em aplicativos clientes do MQTT

O modelo de programação cliente MQTT usa encadeamentos extensivamente. Os encadeamentos desacoplam um aplicativo cliente MQTT , tanto quanto possível, de atrasos na transmissão de mensagens para e do servidor. Publicações, tokens de entrega e eventos perdidos de conexão são entregues para os métodos em uma classe de retorno de chamada que implementa `MqttCallback`.

Retornos de chamada

A interface `MqttCallback` possui três métodos de retorno de chamada; consulte uma implementação de exemplo em Callback.java.

`connectionLost(java.lang.Throwable cause)`

`connectionLost` será chamado quando um erro de comunicação levar ao descarte da conexão. Ele também será chamado, se o servidor descartar a conexão como resultado de um erro no servidor após a conexão ter sido estabelecida. Os erros do servidor são registrados no log de erros do gerenciador de filas. O servidor descarta a conexão com o cliente e o cliente chama o `MqttCallback.connectionLost`.

Apenas os erros remotos lançados como exceções no mesmo encadeamento que o aplicativo cliente são exceções a partir de `MqttClient.connect`. Os erros detectados pelo servidor após a conexão ser estabelecida serão relatados de volta ao método de retorno de chamada `MqttCallback.connectionLost` como `throwables`.

Os erros típicos do servidor que resultam em `connectionLost` são erros de autorização. Por exemplo, o servidor de telemetria tenta publicar em um tópico em nome de um cliente que não está autorizado a publicar no tópico. Qualquer coisa que resulte no retorno de um código de condição `MQCC_FAIL` ao servidor de telemetria pode resultar no descarte da conexão.

`deliveryComplete(MqttDeliveryToken token)`

`deliveryComplete` é chamado pelo cliente do MQTT para transmitir um token de entrega de volta para o app do cliente; consulte "Tokens de entrega" na página 130 Usando o token de entrega, o retorno de chamada pode acessar a mensagem publicada com o método `token.getMessage`.

Quando o retorno de chamada do aplicativo retornar o controle para o cliente MQTT após ser chamado pelo método `deliveryComplete`, a entrega será concluída. Até que a entrega seja concluída, mensagens com QoS 1 ou 2 serão retidas pela classe de persistência.

A chamada para `deliveryComplete` é um ponto de sincronização entre o aplicativo e a classe de persistência. O método `deliveryComplete` nunca é chamado duas vezes para a mesma mensagem.

Quando o retorno de chamada do aplicativo retorna de `deliveryComplete` para o cliente MQTT, o cliente chama `MqttClientPersistence.remove` para mensagens com QoS 1 ou 2. `MqttClientPersistence.remove` exclui a cópia armazenada localmente da mensagem publicada.

De uma perspectiva de processamento de transações, a chamada para `deliveryComplete` é uma transação de fase única que confirma a entrega. Se o processamento falhar durante o retorno de chamada, na reinicialização do cliente, `MqttClientPersistence.remove` será chamado novamente para excluir a cópia local da mensagem publicada. O retorno de chamada não é chamado novamente. Se você estiver usando o retorno de chamada para armazenar um log de mensagens entregues, não será possível sincronizar o log com o cliente MQTT. Se você deseja armazenar um log confiavelmente, então atualize o log na classe `MqttClientPersistence`.

O token de entrega e a mensagem são referenciados pelo encadeamento de aplicativos principal e o cliente MQTT. O cliente MQTT desreferenciará do objeto `MqttMessage` quando a entrega for concluída e o objeto do token de entrega quando o cliente for desconectado. O objeto `MqttMessage` poderá ser lixo coletado após a entrega ser concluída se o aplicativo cliente desreferenciar ele. O token de entrega poderá ser lixo coletado após a sessão ser desconectada.

É possível obter os atributos `MqttDeliveryToken` e `MqttMessage` após uma mensagem ter sido publicada. Se você tentar configurar quaisquer atributos `MqttMessage` após a mensagem ter sido publicada, o resultado será indefinido.

O cliente MQTT continuará a processar confirmações de entrega, se o cliente se reconectar à sessão anterior com o mesmo `ClientIdentifier`; consulte "Sessões limpas" na página 128. O aplicativo cliente do MQTT deve configurar `MqttClient.CleanSession` como `false` para a sessão anterior e configurá-lo como `false` na nova sessão O cliente MQTT cria novos tokens de entrega e objetos de mensagem na nova sessão para as entregas pendentes. Ele recupera os objetos usando a classe `MqttClientPersistence`. Se o aplicativo cliente ainda tiver referências ao antigo tokens de entrega e mensagens, os desreferenciem. O retorno de chamada do aplicativo é chamado na nova sessão para quaisquer entregas iniciada na sessão anterior e concluídas nessa sessão.

O retorno de chamada do aplicativo será chamado depois que o aplicativo cliente se conectar, quando uma entrega pendente for concluída. Antes de o aplicativo cliente se conectar, ele poderá recuperar entregas pendentes usando o método `MqttClient.getPendingDeliveryTokens`.

Observe que o aplicativo cliente criou originalmente o objeto de mensagem publicada e sua matriz de bytes da carga útil. O cliente MQTT referencia esses objetos. O objeto de mensagem retornado pelo token de entrega no método `token.getMessage` não é necessariamente o mesmo objeto de mensagem criado pelo cliente. Se uma nova instância do cliente MQTT recriar o token de entrega, a classe `MqttClientPersistence` recriará o objeto `MqttMessage`.

Para consistência, o `token.getMessage` retornará `null` se o `token.isCompleted` for `true`, independentemente se o objeto de mensagem foi criado pelo aplicativo cliente ou pela classe `MqttClientPersistence`.

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` será chamado quando uma publicação chegar ao cliente que correspondeu a um tópico de assinatura. `topic` é o tópico da publicação, não o filtro de assinatura. Os dois poderão ser diferentes se o filtro contiver caracteres curingas.

Se o tópico corresponder a diversas assinaturas criadas pelo cliente, o cliente receberá diversas cópias da publicação. Se um cliente publicar em um tópico que também assina, ele receberá uma cópia da sua própria publicação.

Se uma mensagem for enviada com um QoS de 1 ou 2, a mensagem será armazenada pelo `MqttClientPersistence` de classe antes de o cliente MQTT chamar o `messageArrived`. `messageArrived` se comporta como `deliveryComplete`: é chamado apenas uma vez para uma publicação e a cópia local da publicação será removida por `MqttClientPersistence.remove` quando `messageArrived` retornar ao cliente MQTT. O cliente MQTT descartará suas referências para o tópico e a mensagem quando `messageArrived` retornar ao cliente MQTT. Os objetos de tópicos e mensagens serão o lixo coletado, se o aplicativo cliente não tiver retido em uma referência aos objetos.

Retornos de chamadas, encadeamento e sincronização do aplicativo cliente

O cliente MQTT chama um método de retorno de chamada em um encadeamento separado para o encadeamento de aplicativo principal. O aplicativo cliente não cria um encadeamento para o retorno de chamada, ele é criado pelo cliente MQTT .

O cliente MQTT sincroniza os métodos de retorno de chamada. Apenas uma instância do método de retorno de chamada é executada por vez. A sincronização torna mais fácil a atualização de um objeto que registra a contagem total que as publicações foram entregues. Uma instância do `MqttCallback.deliveryComplete` é executada por vez, e, portanto, é seguro para atualizar a contagem total sem sincronização adicional. Nesse caso, somente uma publicação chega por vez. Seu código no método `messageArrived` pode atualizar um objeto sem sincronizá-lo. Se você estiver se referindo à contagem total ou ao objeto que está sendo atualizado, em outro encadeamento, sincronize a contagem total ou o objeto.

O token de entrega fornece um mecanismo de sincronização entre o encadeamento principal do aplicativo e a entrega de uma publicação. O método `token.waitForCompletion` aguarda até que a entrega de uma publicação específica seja concluída ou até que um tempo limite opcional expire. Você pode usar `token.waitForCompletion` de algumas maneiras simples para processar uma publicação de cada vez:

1. Para pausar o cliente aplicativo até a entrega da publicação seja concluída; consulte PubSync.java.
2. Para sincronizar com o método `MqttCallback.deliveryComplete`. Somente quando o `MqttCallback.deliveryComplete` retornar para o cliente MQTT, `token.waitForCompletion` continuará. Usando esse mecanismo será possível sincronizar a execução de código em `MqttCallback.deliveryComplete` antes de o código ser executado no encadeamento de aplicativos principal.

E se você desejava publicar sem aguardar que cada publicação seja entregue, mas deseja confirmação quando todas as publicações foram entregues? Se você publicar em um único encadeamento, a última publicação a ser enviada será também a última a ser entregue.

Sincronização de solicitações enviadas ao servidor

Tabela 4. Comportamento de sincronização de métodos que resultam em solicitações para o servidor.

Esta tabela lista os métodos no cliente Java do MQTT que enviam uma solicitação para o servidor Para cada método, a tabela descreve as condições sob as quais o método aguarda ou retorna e por quanto tempo ele aguarda.

Método	Sincronização	Intervalo de tempo limite
`MqttClient.Connect`	Aguarda para que uma conexão seja estabelecida com o servidor.	30 segundos por padrão ou como configurado por um parâmetro.
`MqttClient.Disconnect`	Aguarda o cliente MQTT concluir qualquer trabalho que deve ser realizado e a sessão TCP/IP para se desconectar.	30 segundos por padrão ou como configurado por um parâmetro.
`MqttClient.Subscribe`	Aguarda a solicitação de assinatura ser concluída.	30 segundos por padrão ou como configurado por um parâmetro.
`MqttClient.UnSubscribe`	Aguarda a solicitação de cancelamento de assinatura ser concluída.	30 segundos por padrão ou como configurado por um parâmetro.
`MqttClient.Publish`	Retorna imediatamente para o encadeamento de aplicativos após transmitir a solicitação ao cliente MQTT.	Nenhum.
`MqttDeliveryToken.waitForCompletion`	Aguarda o token de entrega a ser retornado.	Indefinido por padrão ou como configurado por um parâmetro.

Sessões limpas

O cliente MQTT e o serviço de telemetria (MQXR) mantém as informações de estado da sessão. As informações de estado são usadas para assegurar a entrega "pelo menos uma vez" e "exatamente uma vez" e o recebimento "exatamente uma vez" das publicações. O estado da sessão também inclui assinaturas criadas por um cliente MQTT. É possível optar por executar um cliente MQTT com ou sem manter informações de estado entre as sessões. Altere o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de conectar.

Ao se conectar a um aplicativo cliente MQTT usando o método `MqttClient.connect`, o cliente identificará a conexão usando o identificador de cliente e o endereço do servidor. O servidor verifica se informações da sessão foram salvas de uma conexão anterior no servidor. Se uma sessão anterior ainda existir e `cleanSession=true`, então, as informações da sessão anterior no cliente e no servidor serão limpas. Se `cleanSession=false`, a sessão anterior será continuada. Se não existir nenhuma sessão anterior, uma nova sessão será iniciada.

Nota: O Administrador do WebSphere MQ pode forçar o fechamento de uma sessão aberta e excluir todas as informações da sessão Se o cliente reabrir a sessão com `cleanSession=false`, uma nova sessão será iniciada.

Publicações

Se você usar o padrão `MqttConnectOptions` ou configurar `MqttConnectOptions.cleanSession` para `true` antes de se conectar ao cliente, todas as entregas de publicação pendentes para o cliente serão removidas quando o cliente se conectar.

A configuração de sessão limpa não tem efeito sobre as publicações enviadas com `QoS=0`. Para `QoS=1` e `QoS=2`, o uso de `cleanSession=true` pode resultar na perda de uma publicação.

Assinaturas

Se você usar o `MqttConnectOptions`padrão ou configurar `MqttConnectOptions.cleanSession` como `true` antes de conectar o cliente, quaisquer assinaturas antigas para o cliente serão removidas quando o cliente se conectar.. Quaisquer novas assinaturas feitas pelo cliente durante a sessão serão removidas quando ele se desconectar.

Se você configurar `MqttConnectOptions.cleanSession` como `false` antes de se conectar, quaisquer assinaturas criadas pelo cliente serão incluídas em todas as assinaturas que existiam para o cliente antes de ele se conectar. Todas as assinaturas permanecem ativas quando o cliente se desconecta.

Outra maneira de entender a maneira como o atributo `cleanSession` afeta assinaturas é pensar nele como um atributo modal. Em seu modo padrão, `cleanSession=true`, o cliente cria assinaturas e recebe publicações apenas dentro do escopo da sessão. No modo alternativo, `cleanSession=false`, assinaturas são duráveis. O cliente pode se conectar e se desconectar e suas assinaturas permanecem ativas. Ao se reconectar, o cliente recebe todas as publicações não entregues. Enquanto estiver conectado, ele pode modificar o conjunto de assinaturas que estão ativas em seu nome.

Deve-se configurar o modo `cleanSession` antes de conectar; o modo dura toda a sessão. Para alterar essa configuração, você deve desconectar e reconectar o cliente. Se você mudar os modos de uso de `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartados

Identificador de Cliente

O identificador de cliente é uma sequência de 23 bytes que identifica um cliente MQTT . O identificador de cliente deve ser exclusivo em todos os clientes que se conectam ao servidor e não deve ser o mesmo que o nome do gerenciador de filas no servidor. Nessas restrições, será possível usar qualquer sequência de identificação. É importante ter um procedimento para alocar identificadores de cliente e um meio de configurar um cliente com seu identificador escolhido.

O identificador do cliente é usado na administração de um sistema MQTT. Com centenas de milhares de clientes em potencial para administrar, é necessário estar apto para identificar rapidamente um cliente específico. Suponha por exemplo, que há um dispositivo e você é notificado, talvez por um cliente ligando para um help desk. Como o cliente identifica o dispositivo e como você correlaciona essa identificação com o servidor que normalmente conectado ao cliente? Você tem que consultar um banco de dados que mapeia cada dispositivo para um identificador do cliente e para um servidor? O nome do dispositivo identifica a qual servidor ele está conectado? Ao navegar pelas conexões do cliente MQTT, cada conexão será rotulada com o identificador do cliente. É necessário consultar uma tabela para mapear um identificador do cliente para um dispositivo físico?

O identificador do cliente identifica um determinado dispositivo, usuário ou aplicativo em execução no cliente? Se um cliente substituir um dispositivo com falha por um novo, o novo dispositivo terá o mesmo identificador que o dispositivo antigo? Você aloca um novo identificador? Se você mudar um dispositivo físico, mas mantiver o mesmo identificador, as publicações pendentes e assinaturas ativas serão automaticamente transferidas para o novo dispositivo.

Como você assegura que identificadores do cliente sejam exclusivos? Assim como um sistema para gerar identificadores exclusivos, deve-se ter um processo confiável para configurar o identificador no cliente. Talvez o dispositivo do cliente seja uma "caixa preta" sem interface com o usuário. Você fabrica o dispositivo com um identificador do cliente - como o uso de um endereço de Controle de Acesso à Mídia?

Ou você tem um processo de instalação e configuração de software que configura o dispositivo antes de ele ser ativado?

Você pode criar um identificador do cliente a partir do endereço de Controle de Acesso à Mídia do dispositivo com 48 bits para manter o identificador curto e exclusivo. Se o tamanho da transmissão não for um problema crítico, você poderá usar os 17 bytes restantes para deixar o endereço mais fácil de administrar.

Tokens de entrega

Quando um cliente publica em um tópico, um novo token de entrega é criado. Use o token de entrega para monitorar a entrega de uma publicação ou para bloquear o aplicativo cliente até que a entrega seja concluída.

O token é um objeto `MqttDeliveryToken`. Ele é criado chamando o método MqttTopic.publish() e é retido pelo cliente MQTT até que a sessão do cliente seja desconectada e a entrega seja concluída.

O uso normal do token é para verificar se a entrega foi concluída. Bloqueie o aplicativo cliente até que a entrega seja concluída usando o token retornado para chamar `token.waitForCompletion`. Como alternativa, forneça um identificador `MqttCallBack`. Quando o cliente MQTT tiver recebido todas as confirmações esperadas como parte da entrega da publicação, ele chamará `MqttCallBack.deliveryComplete` transmitindo o token de entrega como um parâmetro.

Até a entrega ser concluída, será possível inspecionar a publicação usando o token de entrega retornado chamando `token.getMessage`.

Entregas concluídas

A conclusão de entregas é assíncrona e depende da qualidade de serviço associada à publicação.

No máximo uma vez

`QoS=0`

A entrega é concluída imediatamente no retorno de `MqttTopic.publish`. `MqttCallback.deliveryComplete` é chamado imediatamente.

Pelo menos uma vez

`QoS=1`

A entrega será concluída quando uma confirmação da publicação tiver sido recebida do gerenciador de filas. `MqttCallback.deliveryComplete` será chamado quando a confirmação for recebida. A mensagem poderá ser entregue mais de uma vez antes de `MqttCallback.deliveryComplete` ser chamado, se as comunicações forem lentas ou não confiáveis.

Exatamente uma vez

`QoS=2`

A entrega será concluída quando o cliente receber uma mensagem de conclusão de que a publicação foi publicada para os assinantes. `MqttCallback.deliveryComplete` será chamado assim que a mensagem de publicação for recebida. Ele não espera a mensagem de conclusão.

Em raras circunstâncias, o aplicativo cliente pode não retornar normalmente ao cliente MQTT a partir do `MqttCallback.deliveryComplete`. Você sabe que a entrega foi concluída porque o `MqttCallback.deliveryComplete` foi chamado. Se o cliente reiniciar a mesma sessão, `MqttCallback.deliveryComplete` não será chamado novamente.

Entregas incompletas

Se a entrega não for concluída após a sessão do cliente ser desconectada, será possível conectar o cliente novamente e concluir a entrega. Só será possível concluir a entrega de uma mensagem, se a mensagem foi publicada em uma sessão com o atributo `MqttConnectionOptions` configurado como `false`.

Crie o cliente usando o mesmo identificador de cliente e endereço do servidor e, em seguida, se conecte configurando o atributo `cleanSession` `MqttConnectionOptions` como `false` novamente. Se você configurar `cleanSession` como `true`, os tokens de entrega pendentes serão descartados.

Será possível verificar se há alguma entrega pendente chamando `MqttClient.getPendingDeliveryTokens`. Será possível chamar `MqttClient.getPendingDeliveryTokens` antes de se conectar ao cliente.

Publicação last will and testament

Se uma conexão do cliente MQTT for encerrada inesperadamente, será possível configurar o WebSphere MQ Telemetry para enviar uma publicação de "último testamento". Predefina o conteúdo da publicação e o tópico para enviá-lo. A "last will and testament" é uma propriedade da conexão. Crie-o antes de conectar o cliente.

Crie um tópico para a publicação last will and testament. Você pode criar um tópico, como `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`, por exemplo,

Configure um "último testamento" usando o método `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` .

Considere criar um registro de data e hora na mensagem `lastWillPayload`. Inclua outras informações do cliente que auxiliem na identificação do cliente e nas circunstâncias da conexão. Transmita o objeto `MqttConnectionOptions` para o construtor `MqttClient`.

Configure `lastWillQos` como 1 ou 2, para tornar a mensagem persistente em IBM WebSphere MQe para assegurar a entrega Para reter as informações da última conexão perdida, configure `lastWillRetained` como `true`.

A publicação "last will and testament" será enviada para os assinantes, se a conexão terminar inesperadamente. Ela será enviada se a conexão terminar sem que o cliente chame o método `MqttClient.disconnect`.

Para monitorar conexões, complemente a publicação "last will and testament" com outras publicações para registrar conexões e desconexões programadas.

Persistência de Mensagem em Clientes MQTT

As mensagens de publicação serão transformadas em persistentes se forem enviadas com uma qualidade de serviço de "pelo menos uma vez" ou "exatamente uma vez". É possível implementar seu próprio mecanismo de persistência no cliente ou usar o mecanismo de persistência padrão fornecido com o cliente. A persistência funciona em ambas as direções, para publicações que são enviadas para ou pelo o cliente.

No MQTT, a persistência de mensagem possui dois aspectos; como a mensagem é transferida e se ela está enfileirada no servidor MQTT como uma mensagem persistente.

1. O cliente MQTT acopla a persistência da mensagem com a qualidade de serviço. Dependendo da qualidade de serviço escolhida para uma mensagem, a mensagem se torna persistente. A persistência de mensagem é necessária para implementar a qualidade de serviço necessária.

   Se você especificar "no máximo uma vez", `QoS=0`, o cliente descartará a mensagem assim que for publicada. Se houver alguma falha no processamento de envio de dados da mensagem, ela não será enviada novamente. Mesmo se o cliente permanecer ativo, a mensagem não será enviada novamente. O comportamento das mensagens `QoS=0` é o mesmo que as mensagens não persistentes rápidas do IBM WebSphere MQ.

   Se uma mensagem for publicada por um cliente com `QoS` de 1 ou 2, ela se tornará persistente. A mensagem é armazenada localmente e descartada apenas do cliente quando ela não é mais necessária para assegurar "pelo menos uma vez", `QoS=1`ou "exatamente uma vez", `QoS=2`, entrega.
2. Se uma mensagem estiver marcada como `QoS` 1 ou 2, ela será enfileirada como uma mensagem persistente. Se estiver marcada como `QoS=0`, então ela estará enfileirada como uma mensagem

não persistente. Em IBM WebSphere MQ , as mensagens não persistentes são transferidas entre os gerenciadores de filas "exatamente uma vez", a menos que o canal de mensagens tenha o atributo NPMSPEED configurado como FAST.

Uma publicação persistente é armazenada no cliente até ser recebida por um aplicativo cliente. Para QoS=2, a publicação será descartada a partir do cliente quando o retorno de chamada do aplicativo retornar ao controle. Para QoS=1, o aplicativo poderá receber a publicação novamente, se ocorrer uma falha. Para QoS=0, o retorno de chamada não recebe a publicação mais de uma vez. Ele poderá não receber a publicação se houver uma falha ou se o cliente estiver desconectado no momento da publicação.

Quando você se subscrever para um tópico, será possível reduzir a QoS com a qual o assinante recebe as mensagens para corresponder a suas capacidades de persistência. As publicações criadas em um QoS maior, são enviadas com o QoS mais alto solicitado pelo assinante.

Armazenando de mensagens

A implementação do armazenamento de dados em pequenos dispositivos varia bastante. O modelo de mensagens persistentes de salvamento temporariamente em armazenamento gerenciado pelo cliente MQTT, pode ser muito lento ou demandar muito armazenamento. Em dispositivos móveis, o sistema operacional móvel poderá fornecer um serviço de armazenamento ideal para mensagens do MQTT.

Para fornecer flexibilidade para atender às restrições de pequenos dispositivos, o cliente MQTT tem duas interfaces de persistência. As interfaces definem as operações envolvidas no armazenamento de mensagens persistentes. As interfaces são descritas na documentação da API para o Cliente de MQTT para Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT.. É possível implementar as interfaces para se adequar a um dispositivo. O cliente MQTT executado no Java SE tem uma implementação padrão das interfaces que armazenam mensagens persistentes no sistema de arquivos. Ele usa o pacote `java.io`. O cliente também possui uma implementação padrão para o Java ME, `MqttDefaultMIDPPersistence`

Classes de persistência

`MqttClientPersistence`
Transmite uma instância de sua implementação de `MqttClientPersistence` para o cliente MQTT como um parâmetro do construtor `MqttClient`. Se você omitir o parâmetro `MqttClientPersistence` do construtor `MqttClient`, o cliente MQTT armazenará as mensagens persistentes usando a classe `MqttDefaultFilePersistence` ou `MqttDefaultMIDPPersistence`.

`MqttPersistable`
`MqttClientPersistence` obtém e coloca objetos `MqttPersistable` usando uma chave de armazenamento. Deve-se fornecer uma implementação de `MqttPersistable`, bem como a implementação de `MqttClientPersistence`, se não estiver usando o `MqttDefaultFilePersistence` ou `MqttDefaultMIDPPersistence`.

`MqttDefaultFilePersistence`

O cliente MQTT fornece a classe `MqttDefaultFilePersistence`. Se você instanciar `MqttDefaultFilePersistence` em seu aplicativo cliente, será possível fornecer o diretório para armazenar mensagens persistentes como um parâmetro do construtor `MqttDefaultFilePersistence`.

Como alternativa, o cliente MQTT pode instanciar `MqttDefaultFilePersistence` e colocar arquivos em um diretório padrão. O nome do diretório é *`client identifier`*`-tcp` *`hostname portnumber`* `"\", "\\", "/", ":"` e `" "` são removidos da sequência do nome do diretório

O caminho para o diretório é o valor da propriedade de sistema `rcp.data`. Se `rcp.data` não estiver configurado, o caminho será o valor da propriedade de sistema `usr.data`.

`rcp.data` é uma propriedade associada à instalação de uma Eclipse Rich Client Platform (RCP) ou um OSGi.

usr.data é o diretório no qual o comando Java que iniciou o aplicativo foi ativado..

MqttDefaultMIDPPersistence
MqttDefaultMIDPPersistence tem um construtor padrão e nenhum parâmetro. Ele usa o pacote javax.microedition.rms.RecordStore para armazenar mensagens.

Publicações

Publicações são instâncias de MqttMessage associadas a uma sequência de tópicos. O cliente do MQTT pode criar publicações para enviar para o IBM WebSphere MQe assinar tópicos no IBM WebSphere MQ MQ para receber publicações

Um MqttMessage tem uma matriz de bytes como sua carga útil. Tente manter as mensagens o menor possível. O comprimento máximo de mensagem permitido pelo protocolo do MQTT é 250 MB.

Geralmente, um programa cliente do MQTT usa java.lang.String ou java.lang.StringBuffer para manipular o conteúdo da mensagem. Por conveniência, a classe MqttMessage tem um método toString para converter sua carga útil para uma sequência. Para criar a carga útil da matriz de bytes a partir de um java.lang.String ou java.lang.StringBuffer, use o método getBytes.

O método getBytes converte uma sequência para o conjunto de caracteres padrão para a plataforma. O conjunto de caracteres padrão geralmente é UTF-8. As publicações do MQTT que contêm apenas texto são normalmente codificadas em UTF-8. Use o método getBytes("UTF8") para substituir o conjunto de caracteres padrão.

No IBM WebSphere MQ, uma publicação do MQTT é recebida como uma mensagem jms-bytes. A mensagem inclui uma pasta MQRFH2 que contém um <mqtt> e uma pasta <mqps>. A pasta <mqtt> contém o clientId e o qos, mas esse conteúdo poderá mudar no futuro.

Um MqttMessage tem três atributos adicionais: qualidade de serviço, se está retida e se é uma duplicata. O sinalizador duplicado será configurado somente se a qualidade de serviço for "pelo menos uma vez" ou "exatamente uma vez". Se a mensagem foi enviada anteriormente e não for confirmada rápido suficiente pelo cliente MQTT, a mensagem será enviada novamente, com o atributo duplicado configurado como true.

Publicando

Para criar uma publicação em um aplicativo cliente do MQTT , crie um MqttMessage Configure sua carga útil, qualidade de serviço e se ela está retida, e chame o método MqttTopic.publish(MqttMessage message); MqttDeliveryToken é retornado e a conclusão da publicação é assíncrona.

Como alternativa, o cliente MQTT pode criar um objeto de mensagem temporário para você a partir dos parâmetros no método MqttTopic.publish(byte [] payload, int qos, boolean retained) ao criar uma publicação.

Se a publicação tiver uma qualidade de serviço "pelo menos uma vez" ou "exatamente uma vez", QoS=1 ou QoS=2, o cliente MQTT chamará a interface MqttClientPersistence. Ele chamará MqttClientPersistence para armazenar a mensagem antes de retornar um token de entrega para o aplicativo.

O aplicativo pode escolher bloquear até a mensagem ser entregue ao servidor usando o método MqttDeliveryToken.waitForCompletion. Como alternativa, o aplicativo pode continuar sem bloqueio. Se você deseja verificar se as publicações foram entregues, sem bloqueio, registre uma instância de uma classe de retorno de chamada que implementa MqttCallback com o cliente MQTT. O cliente MQTT chama o método MqttCallback.deliveryComplete assim que a publicação foi entregue. Dependendo da qualidade de serviço, a entrega pode ser quase imediatamente para QoS=0 ou ela pode levar algum tempo para QoS=2.

Use o método MqttDeliveryToken.isComplete para pesquisar se a entrega está concluída. Enquanto o valor de MqttDeliveryToken.isComplete for false, será possível chamar MqttDeliveryToken.getMessage para obter o conteúdo da mensagem. Se o resultado da chamada MqttDeliveryToken.isComplete for true, a mensagem foi descartada e a chamada de MqttDeliveryToken.getMessage lançaria uma exceção nula de ponteiro.

Não há nenhuma sincronização integrada entre o `MqttDeliveryToken.getMessage` e `MqttDeliveryToken.isComplete`.

Se o cliente se desconectar antes de receber todos os tokens de entrega pendentes, uma nova instância do cliente poderá consultar os tokens de entrega pendentes antes de se conectar. Até que o cliente se conecte, nenhuma nova entrega será concluída e segura para chamar `MqttDeliveryToken.getMessage`. Use o método `MqttDeliveryToken.getMessage` para descobrir quais publicações não foram entregues. Os tokens de entrega pendentes serão descartados se você se conectar com `MqttConnectOptions.cleanSession` configurado em seu valor padrão, `true`.

Assinando

Um gerenciador de filas ou IBM MessageSight é responsável por criar publicações para enviar para um assinante do MQTT. O gerenciador de filas verificará se o filtro de tópicos em uma assinatura criada por um cliente MQTT corresponderá à sequência de tópicos em uma publicação. A correspondência pode ser uma correspondência exata ou a correspondência pode incluir caracteres curingas. Antes de a publicação ser encaminhada para o assinante pelo gerenciador de filas, o gerenciador de filas verificará os atributos do tópico associada à publicação. Ele segue o procedimento de procura descrito em Assinatura usando uma sequência de tópicos que contém caracteres curingas para identificar se um objeto de tópico administrativo concederá a autoridade do usuário para assinatura.

Quando o cliente MQTT receber uma publicação com a qualidade de serviço "pelo menos uma vez", ele chamará o método `MqttCallback.messageArrived` para processar a publicação. Se a qualidade de serviço da publicação for "exatamente uma vez", `QoS=2`, o cliente MQTT chamará a interface `MqttClientPersistence` para armazenar a mensagem quando ela for recebida. Ela então chama `MqttCallback.messageArrived`.

Qualidades de serviço fornecidas por um cliente MQTT

Um cliente MQTT fornece três qualidades de serviço para entregar publicações para o WebSphere MQ e para o cliente MQTT : "no máximo uma vez", "pelo menos uma vez" e "exatamente uma vez". Quando um cliente MQTT enviar uma solicitação para o IBM WebSphere MQ para criar uma assinatura, a solicitação será enviada com a qualidade de serviço "pelo menos uma vez".

A qualidade de serviço de uma publicação é um atributo de `MqttMessage`. Ela é configurada pelo método `MqttMessage.setQos`.

O método `MqttClient.subscribe` pode reduzir a qualidade de serviço aplicada às publicações enviadas para um cliente em um tópico. A qualidade de serviço de uma publicação encaminhada para um assinante pode ser diferente da qualidade de serviço da publicação. O menor dos dois valores é usado para encaminhar uma publicação.

No máximo uma vez
`QoS=0`

A mensagem é entregue no máximo uma vez ou não é entregue de modo algum. Sua entrega na rede não é reconhecida.

A mensagem não é armazenada. A mensagem poderá ser perdida se o cliente for desconectado ou se o servidor falhar.

`QoS=0` é o modo de transferência mais rápido. Ele é, às vezes, chamado de "fire and forget".

O protocolo MQTT não requer que servidores encaminhem publicações com `QoS=0` para um cliente. Se o cliente estiver desconectado no momento em que o servidor receber a publicação, a publicação poderá ser descartada, dependendo do servidor. O serviço de telemetria (MQXR) não descarta mensagens enviadas com `QoS=0`. Elas são armazenadas como mensagens não persistentes e só serão descartadas, se o gerenciador de filas for interrompido.

Pelo menos uma vez
`QoS=1`

`QoS=1` é o modo de transferência padrão.

A mensagem é sempre entregue pelo menos uma vez. Se o emissor não receber uma confirmação, a mensagem será enviada novamente com o sinalizador DUP configurado até que uma confirmação seja recebida. Como resultado, o receptor pode receber a mesma mensagem diversas vezes e processá-la diversas vezes.

A mensagem deve ser armazenada localmente no emissor e no receptor até ser processada.

A mensagem será excluída do receptor após ter processado a mensagem. Se o receptor for um broker, a mensagem será publicada para seus assinantes. Se o receptor for um cliente, a mensagem será entregue para o aplicativo de assinante. Após a mensagem ser excluída, o receptor enviará uma confirmação para o emissor.

A mensagem será excluída do emissor após ter recebido uma confirmação do receptor.

Exatamente uma vez

QoS=2

A mensagem é sempre entregue exatamente uma vez.

A mensagem deve ser armazenada localmente no emissor e no receptor até ser processada.

QoS=2 é o mais seguro e o modo de transferência mais lento. Ele levará pelo menos dois pares de transmissões entre o emissor e o receptor antes de a mensagem ser excluída do emissor. A mensagem poderá ser processada no receptor após a primeira transmissão.

No primeiro par de transmissões, o emissor transmite a mensagem e recebe a confirmação do receptor de que ele armazenou a mensagem. Se o emissor não receber uma confirmação, a mensagem será enviada novamente com o sinalizador DUP configurado até que uma confirmação seja recebida.

No segundo par de transmissões, o emissor informa ao receptor que ele pode concluir o processamento da mensagem, "PUBREL" Se o emissor não receber uma confirmação da mensagem "PUBREL" , a mensagem "PUBREL" será enviada novamente até que uma confirmação seja recebida. O emissor exclui a mensagem salva quando recebe o reconhecimento para a mensagem "PUBREL"

O receptor poderá processar a mensagem na primeira ou na segunda fase, contanto que não reprocesse a mensagem. Se o receptor for um broker, ele publicará a mensagem para os assinantes. Se o receptor for um cliente, ele entregará a mensagem para o aplicativo de assinante. O receptor envia uma mensagem de conclusão de volta para o emissor que concluiu o processamento da mensagem.

Publicações Retidas e Clientes MQTT

Se você criar uma assinatura para um tópico que tenha uma publicação retida, a publicação retida mais recente no tópico será encaminhada imediatamente para você.

Use o método `MqttMessage.setRetained` para especificar se uma publicação em um tópico está ou não retida.

Para excluir uma publicação retida no IBM WebSphere MQ, execute o comando MQSC CLEAR TOPICSTR**CLEAR TOPICSTR**.

Se você criar uma publicação com uma carga útil nula, a publicação vazia será encaminhada aos assinantes. Outros brokers do MQTT podem não encaminhar uma publicação vazia aos assinantes.

Se você publicar uma publicação não retida para um tópico que tenha uma publicação retida, a publicação retida não será afetada. Os assinantes atuais recebem a nova publicação. Novos assinantes recebem a publicação retida por primeiro, em seguida, recebem quaisquer novas publicações.

Ao criar ou atualizar uma publicação retida, envie a publicação com um QoS ou 1 ou 2. Se você enviar com um QoS de 0, o IBM WebSphere MQ cria uma publicação retida não persistente. A publicação não será retida se o gerenciador de filas for interrompido.

Use publicações retidas para registrar o valor mais recente de uma medida. Novos assinantes para o tópico retido recebem imediatamente o valor mais recente da medida. Se nenhuma nova medida for adquirida desde que o assinante se inscreveu por último no tópico de publicação e se o assinante se inscrever novamente, o assinante receberá a publicação retida mais recente no tópico novamente.

Assinaturas

Crie assinaturas para registrar um interesse em tópicos de publicação usando um filtro de tópico. Um cliente pode criar diversas assinaturas ou uma assinatura contendo um filtro de tópicos que use curingas para registrar um interesse em diversos tópicos. Publicações em tópicos correspondentes aos filtros são enviadas para o cliente. Assinaturas podem permanecer ativas enquanto um cliente está desconectado. As publicações são enviadas para o cliente quando ele se reconecta.

Crie assinaturas usando os métodos `MqttClient.subscribe`, passando um ou mais filtros de tópicos e parâmetros de qualidade de serviço. O parâmetro de qualidade de serviço configura a qualidade de serviço máxima que o assinante está preparado para usar para receber uma mensagem. Mensagens enviadas para esse cliente não podem ser entregues com qualidade de serviço superior. A qualidade de serviço é configurada para o valor original mais baixo quando a mensagem foi publicada e para o nível especificado para a assinatura. A qualidade de serviço padrão para o recebimento de mensagens é `QoS=1`, pelo menos uma vez.

A solicitação de assinatura em si é enviada com `QoS=1`.

Publicações são recebidas por um assinante quando o cliente MQTT chama o método `MqttCallback.messageArrived`. O método `messageArrived` também passa a sequência de tópicos com a qual a mensagem foi publicada para o assinante.

É possível remover uma assinatura ou um conjunto ou assinaturas usando os métodos `MqttClient.unsubscribe`.

Um comando do WebSphere MQ pode remover uma assinatura.Listar assinaturas usando o WebSphere MQ Explorer ou usando comandos do **runmqsc** ou PCF Todas as assinaturas do cliente MQTT são nomeadas. Eles recebem um nome do formato: *`ClientIdentifier`*:*`Topic name`*

Se você usar o `MqttConnectOptions`padrão ou configurar `MqttConnectOptions.cleanSession` como `true` antes de conectar o cliente, quaisquer assinaturas antigas para o cliente serão removidas quando o cliente se conectar.. Quaisquer novas assinaturas feitas pelo cliente durante a sessão serão removidas quando ele se desconectar.

Se você configurar `MqttConnectOptions.cleanSession` como `false` antes de se conectar, quaisquer assinaturas criadas pelo cliente serão incluídas em todas as assinaturas que existiam para o cliente antes de ele se conectar. Todas as assinaturas permanecem ativas quando o cliente se desconecta.

Outra maneira de entender a maneira como o atributo `cleanSession` afeta assinaturas é pensar nele como um atributo modal. Em seu modo padrão, `cleanSession=true`, o cliente cria assinaturas e recebe publicações apenas dentro do escopo da sessão. No modo alternativo, `cleanSession=false`, assinaturas são duráveis. O cliente pode se conectar e se desconectar e suas assinaturas permanecem ativas. Ao se reconectar, o cliente recebe todas as publicações não entregues. Enquanto estiver conectado, ele pode modificar o conjunto de assinaturas que estão ativas em seu nome.

Deve-se configurar o modo `cleanSession` antes de conectar; o modo dura toda a sessão. Para alterar essa configuração, você deve desconectar e reconectar o cliente. Se você mudar os modos de uso de `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartados

Publicações que correspondem a assinaturas ativas são enviadas para o cliente assim que são publicadas. Se o cliente estiver desconectado, elas serão enviadas para o cliente se reconectar ao mesmo servidor com o mesmo identificador de cliente e se `MqttConnectOptions.cleanSession` estiver configurado para `false`.

Assinaturas para um determinado cliente são identificadas pelo identificador de cliente. É possível reconectar o cliente de um dispositivo de cliente diferente ao mesmo servidor, continuar com as mesmas assinaturas e receber publicações não entregues.

Sequências e filtros de tópicos em clientes MQTT

As sequências de tópicos e os filtros de tópicos são usados para publicar e para assinar. A sintaxe de sequências e filtros de tópicos em clientes MQTT é basicamente a mesma que as sequências de tópicos no IBM WebSphere MQ.

As sequências de tópicos são usadas para enviar publicações para os assinantes. Crie uma sequência de tópicos usando o método, `MqttClient.getTopic(java.lang.String topicString)`

Os filtros de tópicos são usados para assinar tópicos e receber publicações. Os filtros de tópicos podem conter caracteres curingas. Com caracteres curinga, é possível assinar vários tópicos. Crie um filtro de tópico usando um método de subscrição; por exemplo, `MqttClient.subscribe(java.lang.String topicFilter)`

Sequências de tópicos

A sintaxe de uma sequência de tópico IBM WebSphere MQ é descrita em Sequências de tópicos. A sintaxe de sequências de tópicos do MQTT é descrita na classe `MqttClient` na documentação da API para o Cliente de MQTT para Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

A sintaxe de cada tipo de sequência de tópicos é quase idêntica. Há quatro pequenas diferenças:

1. As sequências de tópicos enviadas para IBM WebSphere MQ por clientes MQTT devem seguir a convenção para nomes do gerenciador de filas. Em particular, as sequências de não tópico não podem conter hífens.
2. Os comprimentos máximos são diferentes. As sequências de tópicos do IBM WebSphere MQ são limitadas a 10.240 caracteres. Um cliente MQTT pode criar sequências de tópicos de até 65.535 bytes.
3. Uma sequência de tópicos criada por um cliente MQTT não pode conter um caractere nulo.
4. No WebSphere Message Broker, um nível de tópico nulo, `'...//...'` era inválido. Os níveis de tópicos nulos são suportados pelo IBM WebSphere MQ.

Diferente da publicação/assinatura do IBM WebSphere MQ, o protocolo `mqttv3` não terá um conceito de um objeto do tópico administrativo. Não é possível construir uma sequência de tópicos a partir de um objeto do tópico e uma sequência de tópicos. No entanto, uma sequência de tópicos é mapeada para um tópico administrativo no WebSphere MQ. O controle de acesso associado ao tópico administrativo determina se uma publicação é publicada para o tópico ou descartada. Os atributos aplicados a uma publicação quando for redirecionada para os assinantes serão influenciados pelos atributos do tópico administrativo.

Filtros de tópico

A sintaxe de um filtro tópico IBM WebSphere MQ é descrita em Esquema curinga baseado em tópicos. A sintaxe dos filtros de tópicos que você pode construir com um cliente MQTT são descritas na classe `MqttClient` na documentação da API para o Cliente de MQTT para Java. Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

A sintaxe de cada tipo de filtro de tópico é quase idêntica. A única diferença está na maneira como diferentes brokers do MQTT interpretam um filtro de tópico No WebSphere Message Broker V6, um curinga de vários níveis só poderia ser usado no final de um filtro de tópico No WebSphere MQ, um curinga multinível pode ser usado em qualquer nível na árvore de tópicos; por exemplo `USA/#/Dutchess County`.

Referência de programação do cliente MQTT

Aqui estão os links para o Mobile Messaging and M2M Pacote do Cliente e para a documentação associada da API do cliente.

No Mobile Messaging and M2M Pacote do Cliente, as bibliotecas do cliente MQTT estão em um pacote configurável com sua documentação da API gerada. É possível fazer download do pacote do cliente a partir de downloads da comunidade do sistema de mensagens IBM

É possível ver as cópias on-line da documentação da API mais recente, seguindo esses links para o projeto do Eclipse Paho:

- Cliente MQTT para classes Java
- Biblioteca do cliente MQTT para C
- Biblioteca do cliente MQTT assíncrona para C

Nota:

1. Vincule aplicativos MQTT Java ao pacote `org.eclipse.paho.client.mqttv3` em vez do `com.ibm.micro.client.mqttv3`. package. O pacote `com.ibm.micro.client.mqttv3` é fornecido para suportar aplicativos existentes do MQTT Java.
2. V 7.5.0.1 Vincule aplicativos do cliente MQTT para C para a biblioteca `MQTTAsync` em vez da biblioteca `MQTTClient`. O `MQTTClient` é fornecido para suportar aplicativos MQTT existentes para C.
3. O Cliente de sistema de mensagens do MQTT para JavaScript requer um servidor MQTT que suporta o WebSockets. Por exemplo, IBM WebSphere MQ Version 7.5 e as versões posteriores realizam isso.

Introdução aos servidores MQTT

Os servidores do sistema de mensagens que suportam o protocolo de transporte do MQTT estão disponíveis na IBM e outros. O servidor MQTT mais básico permite que aplicativos e dispositivos móveis, suportados pelas bibliotecas do cliente do MQTT , troquem mensagens IBM WebSphere MQ e IBM MessageSight são MQTT servidores de IBM. Além de agir como servidores MQTT básicos, eles também trocam mensagens entre aplicativos clientes MQTT e aplicativos corporativos. Todos os servidores MQTT do IBM suportam o protocolo MQTT version 3.1 e MQTT sobre WebSocket protocol.

Servidores MQTT atuais da IBM

IBM WebSphere MQ

- O IBM WebSphere MQ fornece o sistema de mensagens de grau corporativo. O componente de telemetria também permite que o IBM WebSphere MQ aja como um servidor MQTT.
- Isso suporta seu dispositivo móvel, machine-to-machine (M2M) e os aplicativos baseados em dispositivo e também permite que eles troquem mensagens com aplicativos do sistema de mensagens corporativos como aplicativos IBM WebSphere MQ e JMS.
- A instalação do IBM WebSphere MQ inclui uma cópia do MQTT SDK de IBM. Esse SDK fornece aplicativos clientes de amostra do MQTT e MQTT bibliotecas clientes que suportam esses aplicativos.

  Nota: Para obter a versão mais atualizada deste SDK, faça o download do Mobile Messaging and M2M Pacote do Cliente Para obter informações adicionais, consulte “Introdução aos clientes MQTT” na página 11.
- O suporte MQTT incluído pela primeira vez no IBM WebSphere MQ Version 7.0.1. Para obter informações completas para cada liberação do IBM WebSphere MQ, consulte a documentação do produto a seguir.:
  - WebSphere MQ Telemetry Versão 7.5
  - WebSphere MQ Telemetry Versão 7.1

  Para obter uma introdução resumida ao IBM WebSphere MQ e as etapas para iniciar com o componente do IBM WebSphere MQ Telemetry, consulte “IBM WebSphere MQ como o servidor MQTT” na página 140.

IBM MessageSight

- IBM MessageSight é um servidor MQTT baseado em dispositivo que pode conectar um grande número de clientes MQTT ao mesmo tempo e entregar o desempenho e a escalabilidade necessários para acomodar o número cada vez maior de dispositivos móveis e sensores. Ele suporta o protocolo MQTT version 3.1 e MQTT sobre o WebSocket protocol.

- Os principais recursos e benefícios do IBM MessageSight como um servidor MQTT são os seguintes:
  - Sistema de mensagens de alto desempenho, confiabilidade e escaláveis.
  - Projetado especificamente para cenários machine-to-machine (M2M) e Internet of Things suportando grandes comunidades para terminais conectados simultaneamente.
  - Facilidade de instalação e uso. Ele pode estar em funcionamento adequado em 30 minutos.
  - Suporte para aplicativos móveis nativos que incluem o Android e iOS.
  - Integração com o IBM WebSphere MQ como o broker de publicação/assinatura.
- Para obter uma introdução rápida ao IBM MessageSight, consulte a introdução MessageSight no YouTube e o anúncio MessageSight. Para obter informações técnicas detalhadas, consulte a MessageSight da documentação do produto

IBM WebSphere MQ Telemetry daemon for devices

- Isso também é conhecido como o IBM WebSphere MQ Telemetry advanced client for C. É um servidor MQTT com uma pequena área de cobertura que geralmente é executado em locais de satélite ou dispositivos próximos à borda da rede; por exemplo, set-top boxes, unidades de telemetria remota ou terminais de ponto de venda.
- Um uso típico para ele é concentrar muitas conexões do cliente MQTT , que são, então, conectadas ao IBM WebSphere MQ pela Internet em uma única conexão MQTT . Por exemplo, você pode instalar um grande número de sensores em um prédio, conectá-los ao IBM WebSphere MQ Telemetry daemon for devices, e conectar o daemon ao IBM WebSphere MQ
- O IBM WebSphere MQ Telemetry daemon for devices é incluído com o IBM WebSphere MQ. Uma licença separada é necessária para conectá-la ao IBM WebSphere MQ Consulte IBM Comunicado de Software dos Estados Unidos 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) é uma versão do IBM WebSphere MQ Telemetry daemon for devices. A principal diferença está no uso. RSMB é um servidor de teste pequeno, disponível no IBM alphaWorks e destinado para uso ao avaliar ou experiências com soluções baseado em MQTT. O RSMB suporta MQTT em várias plataformas Linux , em Windows XP, em Apple Mac OS X Leoparde em Unslung (Linksys NSLU12)

Servidores MQTT anteriores a partir do IBM

WebSphere Message Broker *(agora conhecido como IBM Integration Bus)*

- WebSphere Message Broker Versão 6 fornecia seu próprio servidor MQTT. O suporte foi substituído no WebSphere Message Broker Versão 7 pelo componente de telemetria do IBM WebSphere MQ.

Outros servidores MQTT

MQTT.org mantém uma lista de servidores MQTT e brokers em sua página do Software incluindo servidores de software livre.

Tarefas relacionadas

"Introdução aos clientes MQTT" na página 11
É possível iniciar o desenvolvimento de um dispositivo móvel ou aplicativo do machine-to-machine (M2M) construindo e executando um aplicativo cliente MQTT de amostra que usa uma biblioteca do cliente MQTT. Os apps de amostra e as bibliotecas do cliente associadas estão disponíveis no Mobile Messaging and M2M Pacote do Cliente da IBM. Há versões dos aplicativos e bibliotecas do cliente gravadas em Java, em JavaScripte em C. É possível executar esses apps na maioria das plataformas e dispositivos, incluindo dispositivos e produtos do Android do Apple.

IBM WebSphere MQ como o servidor MQTT

Uma introdução ao uso do servidor MQTT incluído no IBM WebSphere MQ.

Para começar, siga as etapas nos seguintes artigos:

- "Instalando o IBM WebSphere MQ" na página 140
- "Configurando o serviço MQTT a partir da linha de comandos" na página 142
- "Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer" na página 144

Nota: É possível iniciar rapidamente usando o exemplo da interface da linha de comandos. No entanto, se sua configuração for significativamente diferente para o exemplo, será necessário mais conhecimento e habilidade para usar a interface da linha de comandos efetivamente. Use a interface do IBM WebSphere MQ Explorer para iniciar e executar tarefas de configuração padrão facilmente.

Para obter informações conceituais chave sobre o componente do IBM WebSphere MQ Telemetry, consulte os seguintes artigos na documentação do produto IBM WebSphere MQ:

- Conectando Dispositivos de Telemetria a um Gerenciador de Filas
- Serviço de telemetria (MQXR)
- Canais de telemetria

Informações relacionadas

Configurando um gerenciador de filas para telemetria no Linux e no AIX
Configurando um gerenciador de filas para telemetria no Windows
Configurando Enfileiramento Distribuído para Enviar Mensagens para Clientes MQTT
Administrando o WebSphere MQ Telemetry

Instalando o IBM WebSphere MQ

Siga estas instruções para obter e instalar o IBM WebSphere MQ e configure o IBM WebSphere MQ Telemetry no Windows ou Linux.

Antes de começar

Para os sistemas operacionais suportados pelo serviço MQTT em execução no IBM WebSphere MQ, consulte IBM WebSphere MQ Requisitos do sistema de telemetria.

Obtenha uma cópia dos materiais de instalação do IBM WebSphere MQ e uma licença de uma das seguintes maneiras:

1. Peça ao seu administrador do IBM WebSphere MQ os materiais de instalação e confirme que você pode aceitar o contrato de licença.
2. Obtenha uma cópia de avaliação de 90 dias do IBM WebSphere MQ. Consulte o Avaliar: IBM WebSphere MQ.
3. Compre o IBM WebSphere MQ. Consulte o IBM WebSphere MQ página do produto.

Sobre esta tarefa

Instale o IBM WebSphere MQ como `root` no Linux e como um administrador no Windows. No momento da instalação, selecione as opções adicionais `Telemetry Service` e `Telemetry Clients` para instalar o componente IBM WebSphere MQ Telemetry . Crie um ID do usuário para administrar o IBM WebSphere MQ e verifique se o ID do usuário guest está definido. O ID do usuário guest é usado na configuração de serviço do MQTT de amostra para autorizar acesso do MQTT para IBM WebSphere MQ.

Após instalar o IBM WebSphere MQ, inicie o serviço do MQTT executando as etapas em “Configurando o serviço MQTT a partir da linha de comandos” na página 142 ou “Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer” na página 144.

Procedimento

1. Efetue o logon como `root` no Linux ou como um administrador no Windows.
2. Instale o IBM WebSphere MQ.

   Siga as instruções em Instalando o servidor WebSphere MQ no Linux ou Instalando o servidor WebSphere MQ no Windows. Selecione `Serviço de telemetria` e `Clientes de telemetria` para instalar o componente IBM WebSphere MQ Telemetry .

   No Linux, anote a instrução na seção "O que fazer a seguir" para realizar a instalação primária. Mesmo se esta instalação for a única instalação do IBM WebSphere MQ na estação de trabalho, torne-a primária. Consulte Instalação Única do WebSphere MQ Versão 7.1ou posterior, configurada como a instalação primária

   Para seguir as instruções de configuração de amostra exatamente, deve-se fazer a instalação primária.

   Diversas Instalações: Se você desejar trabalhar com uma instalação não primária, execute o comando setmqenv. Ele configura o ambiente do IBM WebSphere MQ em uma janela de comandos na estação de trabalho. Consulte Instalações múltiplas.

   Supondo que você aceitou o local de instalação padrão oferecido pelo programa de instalação, o IBM WebSphere MQ estará instalado nos seguintes diretórios:

   Linux de 64 bits

   ```
   /opt/mqm
   ```

   Windows de 32 bits

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows de 64 bits

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   O diretório de instalação é mostrado como *MQ_INSTALLATION_PATH*
3. Opcional: Inclua o usuário que você vai administrar o IBM WebSphere MQ com o grupo `mqm` nesta estação de trabalho.

   Esta etapa é opcional no Windows porque é possível administrar o IBM WebSphere MQ como um Windows administrador. Consulte Autoridade para administrar WebSphere MQ em sistemas UNIX e Windows.

   Se sua estação de trabalho do Windows for um membro de um domínio, consulte Domínio do Windows 2000 com permissões de segurança não padrão ou domínio do Windows 2003 e Windows Server 2008 com padrão.

   No Linux, o programa de instalação cria um usuário `mqm`, como um membro do grupo `mqm`. Forneça a esse usuário uma senha ou crie outro usuário com `mqm` como seu grupo primário.
4. Opcional: Conecte-se com o usuário que você tornou um membro do grupo `mqm`.

Esta etapa é opcional no Windows porque é possível administrar o IBM WebSphere MQ como um Windows administrador.

5. Verifique se o ID do usuário guest está definido na estação de trabalho.

   O ID do usuário guest é `"guest"` no Windows e `"nobody"` no Linux. O ID do usuário guest não requer nenhuma permissões ou direitos do sistema operacional.

Resultados

Você instalou o IBM WebSphere MQ em sua estação de trabalho como a instalação principal do IBM WebSphere MQ e criou o grupo `mqm`. A instalação concede permissão para administrar o IBM WebSphere MQ para membros do grupo `mqm`. Os membros do grupo de administradores no Windows também têm autoridade para administrar o IBM WebSphere MQ.

Como proceder a seguir

1. Configure o serviço do MQTT a partir da linha de comandos ou IBM WebSphere MQ Explorer; consulte “Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer” na página 144 ou “Configurando o serviço MQTT a partir da linha de comandos” na página 142.
2. Teste seus clientes Android, iOS, WebSockets, Java e "C" MQTT.
3. Ao concluir o teste, remova o gerenciador de filas e o serviço do MQTT executando o comando *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` no Windows e *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` no Linux.

Informações relacionadas

Instalando o WebSphere MQ Telemetry

Instalando o servidor WebSphere MQ no Linux

Instalando o Servidor WebSphere MQ no Windows

Configurando o serviço MQTT a partir da linha de comandos

Seguir estas instruções configuram o IBM WebSphere MQ usando a linha de comandos para executar os aplicativos IBM WebSphere MQ Telemetry de amostra. As etapas mostram como executar um script para criar um serviço do MQTT em um novo gerenciador de filas chamado `MQXR_SAMPLE_QM`.

Antes de começar

Deve-se ter acesso administrativo a um gerenciador de filas do IBM WebSphere MQ para configurar o serviço de MQTT. Você tem diversas maneiras para obter acesso a um gerenciador de filas:

1. Obtenha uma cópia do IBM WebSphere MQ e crie um gerenciador de filas em sua própria estação de trabalho Linux ou Windows. Siga as instruções em “Instalando o IBM WebSphere MQ” na página 140 para obter e instalar o IBM WebSphere MQ. Observe que também é necessário selecionar `Serviço de Telemetria` e `Clientes de Telemetria` na instalação. Também é possível modificar uma instalação existente para incluir essas opções.
2. Entre em contato com um administrador do IBM WebSphere MQ e peça acesso administrativo para um gerenciador de filas em um servidor que tenha o IBM WebSphere MQ Telemetry instalado como uma opção. V 7.5.0.1 Além do nome do gerenciador de filas, você precisa de pelo menos duas portas TCP/IP para o MQTT e para o MQTT sobre WebSockets. Se estiver planejando conectar clientes seguros, você precisa de pelo menos mais duas portas.

Para executar as etapas na tarefa exatamente como são descritas, você deve estar apto a criar um gerenciador de filas chamado `MQXR_SAMPLE_QM` e a porta TCP/IP 1883 não deve ser utilizada.

Sobre esta tarefa

Nesta tarefa, execute um script que cria um gerenciador de filas e, em seguida, configura o serviço do MQTT para atender às conexões do cliente MQTT V3.1 na porta 1883.A configuração fornece a

todos permissão para publicar e assinar qualquer tópico.. A configuração de segurança e o controle de acesso são mínimos e se destinam apenas a um gerenciador de filas que está em uma rede segura com acesso restrito. Para executar o IBM WebSphere MQ e o MQTT em um ambiente inseguro, você deve configurar a segurança. Para configurar a segurança para o IBM WebSphere MQ e o MQTT, consulte os links relacionados no final dessa tarefa.

Procedimento

1. Efetue logon com um ID do usuário que tenha autoridade administrativa para IBM WebSphere MQ.

   Para definir um ID do usuário com autoridade administrativa para IBM WebSphere MQ, consulte a etapa 3 em "Instalando o IBM WebSphere MQ" na página 140.
2. Abra uma janela de comandos e execute o script de comandos de amostra para criar e iniciar o gerenciador de filas de amostra chamado `MQXR_SAMPLE_QM` e o serviço do MQTT.

   O caminho para o script de amostra é `%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat` no Windows e *`MQ_INSTALLATION_PATH`*`/mqxr/samples/SampleMQM.sh` no Linux.

   Digite o seguinte comando para criar e configurar o gerenciador de filas:

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Resultados

A amostra cria um canal do MQTT chamado `PlainText` com estas propriedades no 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
```

As propriedades do canal no Linux são as mesmas que o Windows, exceto `com.ibm.mq.MQXR.UserName=nobody`.

Os clientes MQTT V3.1 que se conectam à porta 1883 acessam o IBM WebSphere MQ com o ID do usuário configurado na variável *`com.ibm.mq.MQXR.UserName`*. O script de amostra autoriza o ID do usuário com os seguintes comandos do 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
```

O primeiro comando fornece a autoridade do usuário para publicar e assinar os tópicos que herdam suas permissões a partir do tópico base. O segundo comando fornece ao usuário a autoridade para colocar mensagens na fila de transmissão `SYSTEM.MQTT.TRANSMIT.QUEUE`. O serviço MQTT envia mensagens no `SYSTEM.MQTT.TRANSMIT.QUEUE` como publicações para assinantes MQTT .

O script inicia o serviço do MQTT no gerenciador de filas para atender às conexões na porta 1883.

Como proceder a seguir

Siga estas etapas para testar a conexão executando o aplicativo MQTT V3.1 Java de amostra.

A origem para o aplicativo Java de amostra está no arquivo `MQTTV3Sample.java`.

Duas janelas de comandos são necessárias para executar a amostra. Execute a amostra como um assinante em uma janela e como publicador no outra.

- Windows Para iniciar o assinante, execute o comando

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
```

  Para publicar, execute o comando:

```
"%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
```

- Linux Para iniciar o assinante, execute o comando

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
```

  Para publicar, execute o comando:

```
MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
```

A saída de gravação do publicador e do assinante para as janelas de comando:

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figura 27. A saída do publicador

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
Topic:          MQTTV3Sample/Java/v3
Message:        Message from MQTTv3 Java client
QoS:            2
```

Figura 28. A saída do assinante

O servidor agora está pronto para você testar seu app MQTT V3.1 .

Tarefas relacionadas

Configurando o serviço MQTT com o WebSphere MQ Explorer

Siga estas instruções para configurar o IBM WebSphere MQ usando o IBM WebSphere MQ Explorer para executar os clientes IBM WebSphere MQ Telemetry de amostra. As etapas mostram como criar um serviço do MQTT executando o assistente de configuração `Define sample`.

Informações relacionadas

WebSphere MQ Telemetry

Desenvolvendo Aplicativos para WebSphere MQ Telemetry

Administrando o WebSphere MQ Telemetry

Segurança do WebSphere MQ Telemetry

Configurando o serviço do MQTT com o IBM WebSphere MQ Explorer

Siga estas instruções para configurar o IBM WebSphere MQ usando o IBM WebSphere MQ Explorer para executar os clientes IBM WebSphere MQ Telemetry de amostra. As etapas mostram como criar um serviço do MQTT executando o assistente de configuração `Define sample`.

Antes de começar

Deve-se ter acesso administrativo a um gerenciador de filas do IBM WebSphere MQ para configurar o serviço de MQTT. Você tem diversas maneiras para obter acesso a um gerenciador de filas:

1. Obtenha uma cópia do IBM WebSphere MQ e crie um gerenciador de filas em sua própria estação de trabalho Linux ou Windows. Siga as instruções em "Instalando o IBM WebSphere MQ" na página 140 para obter e instalar o IBM WebSphere MQ. Observe que também é necessário selecionar `Serviço de Telemetria` e `Clientes de Telemetria` na instalação. Também é possível modificar uma instalação existente para incluir essas opções.
2. Entre em contato com um administrador do IBM WebSphere MQ e peça acesso administrativo para um gerenciador de filas em um servidor que tenha o IBM WebSphere MQ Telemetry instalado como uma opção. V 7.5.0.1 Além do nome do gerenciador de filas, você precisa de pelo menos duas portas TCP/IP para o MQTT e para o MQTT sobre WebSockets. Se estiver planejando conectar clientes seguros, você precisa de pelo menos mais duas portas.

Para executar as etapas na tarefa exatamente como são descritas, você deve estar apto a criar um gerenciador de filas chamado `MQXR_SAMPLE_QM` e a porta TCP/IP 1883 não deve ser utilizada.

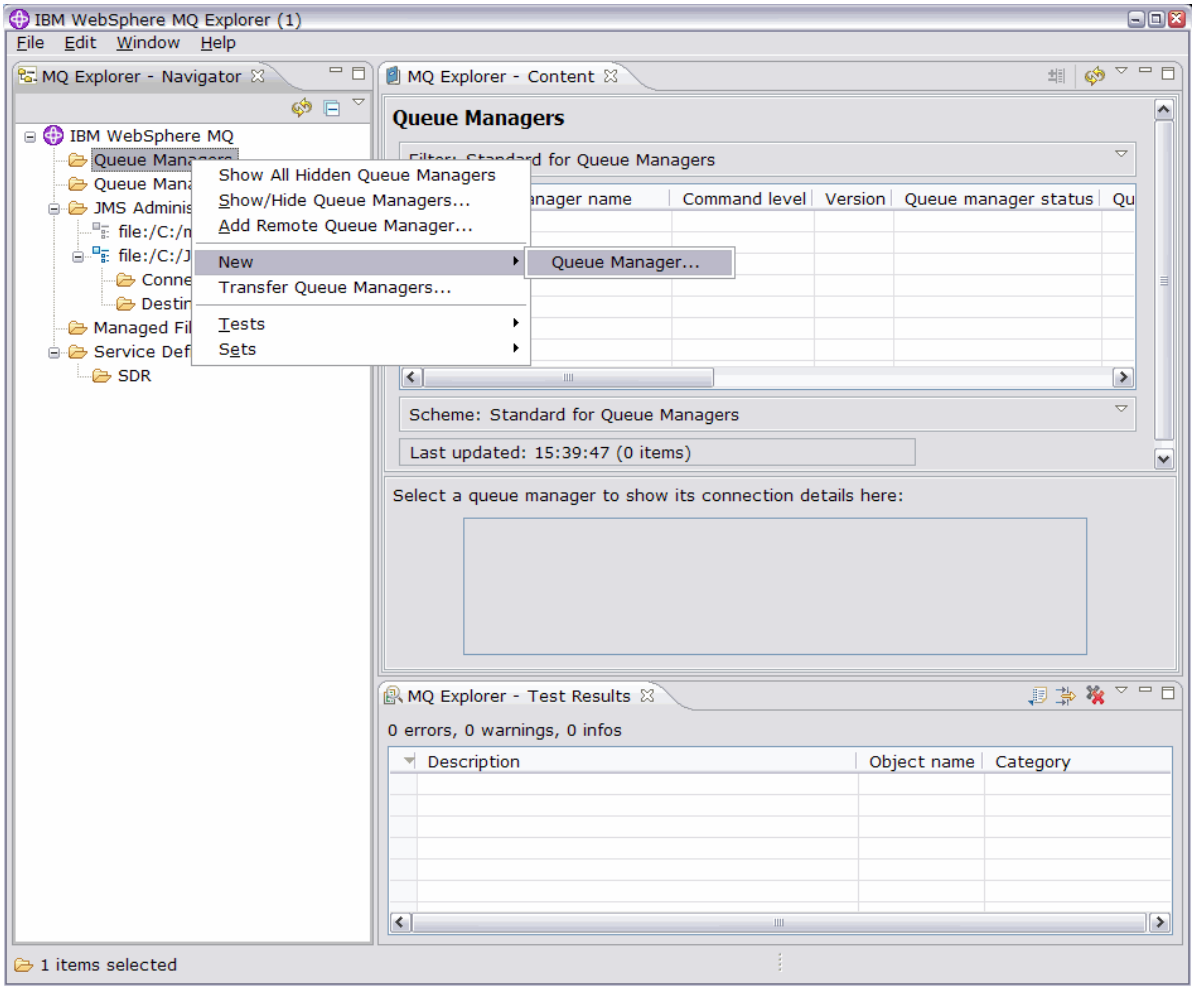
Sobre esta tarefa

Nesta tarefa, execute o assistente de configuração do IBM WebSphere MQ Explorer `Define sample` para criar um serviço MQTT para atender às conexões do cliente MQTT V3.1 na porta 1883. A configuração fornece a todos permissão para publicar e assinar qualquer tópico.. A configuração de segurança e o controle de acesso são mínimos e se destinam apenas a um gerenciador de filas que está em uma rede segura com acesso restrito. Para executar o IBM WebSphere MQ e o MQTT em um ambiente inseguro, você deve configurar a segurança. Para configurar a segurança para o IBM WebSphere MQ e o MQTT, consulte os links relacionados no final dessa tarefa.

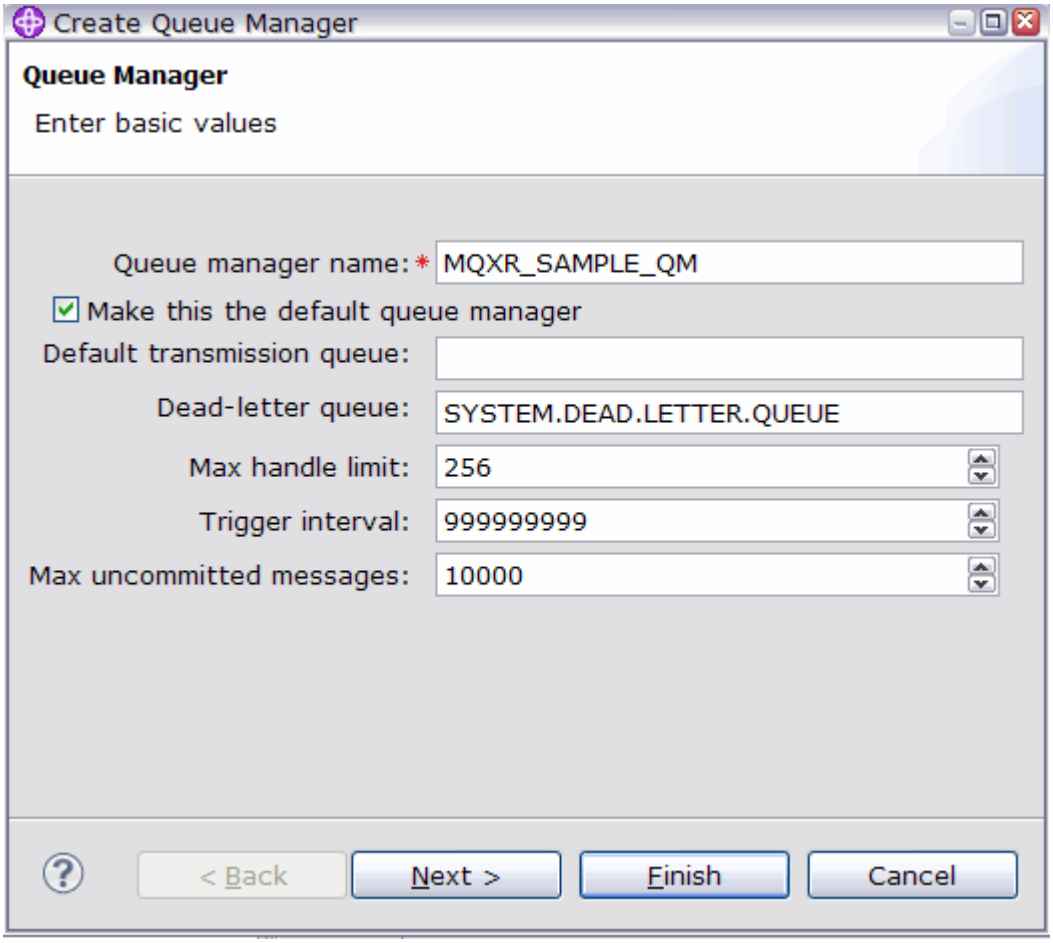
Procedimento

1. Efetue logon com um ID do usuário que tenha autoridade administrativa para IBM WebSphere MQ.

   Para definir um ID do usuário com autoridade administrativa para IBM WebSphere MQ, consulte a etapa 3 em "Instalando o IBM WebSphere MQ" na página 140.
2. Abra uma janela de comandos e execute o comando IBM WebSphere MQ Explorer **strmqcfg** para iniciar o IBM WebSphere MQ Explorer.
3. Crie um gerenciador de filas
   a) Inicie o assistente **Novo gerenciador de filas**

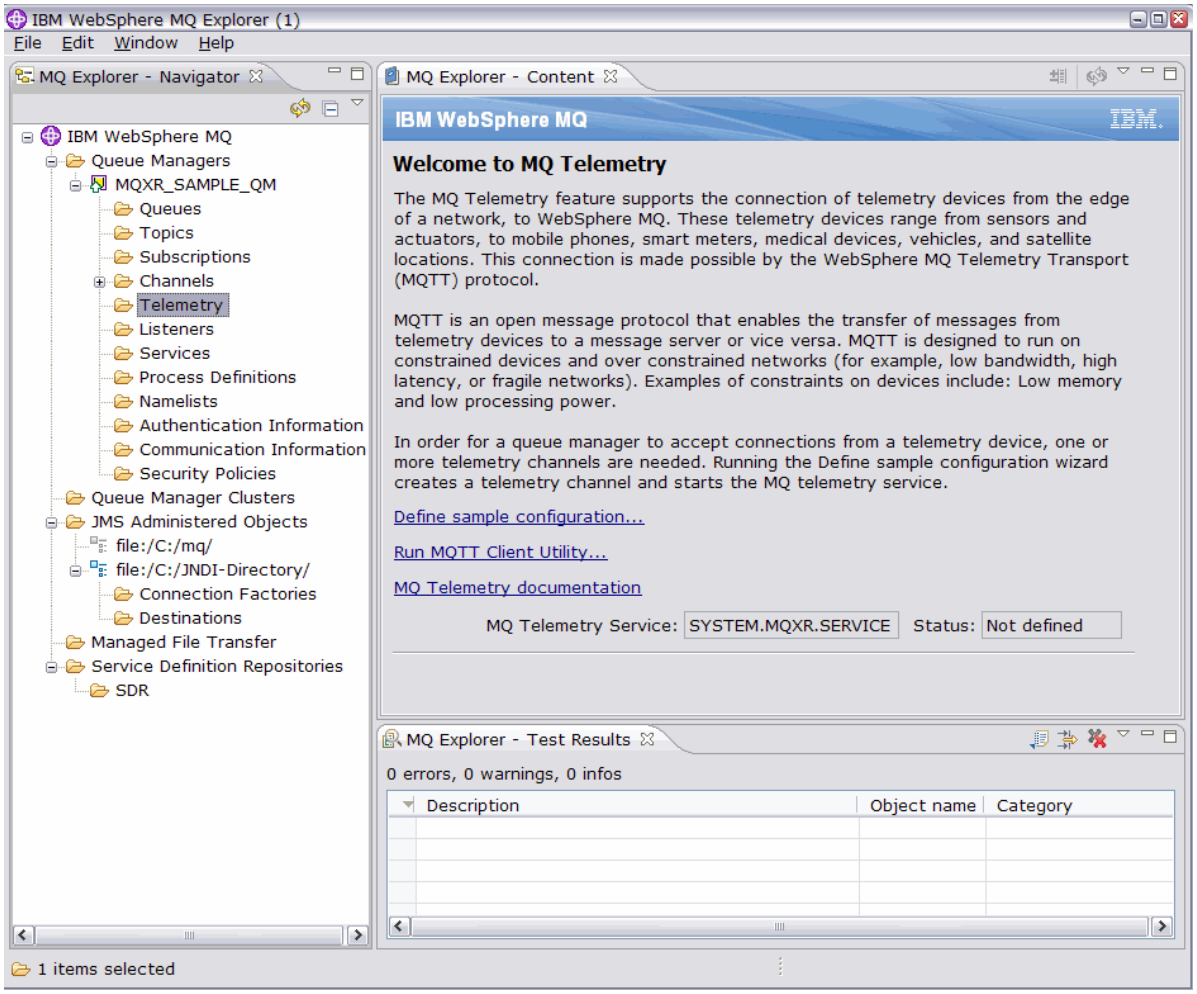


b) Digite um **Nome do gerenciador de filas** e o nome do **Fila de mensagens não entregues**. Por conveniência, faça dele o gerenciador de filas padrão. Clique em **Concluir**.



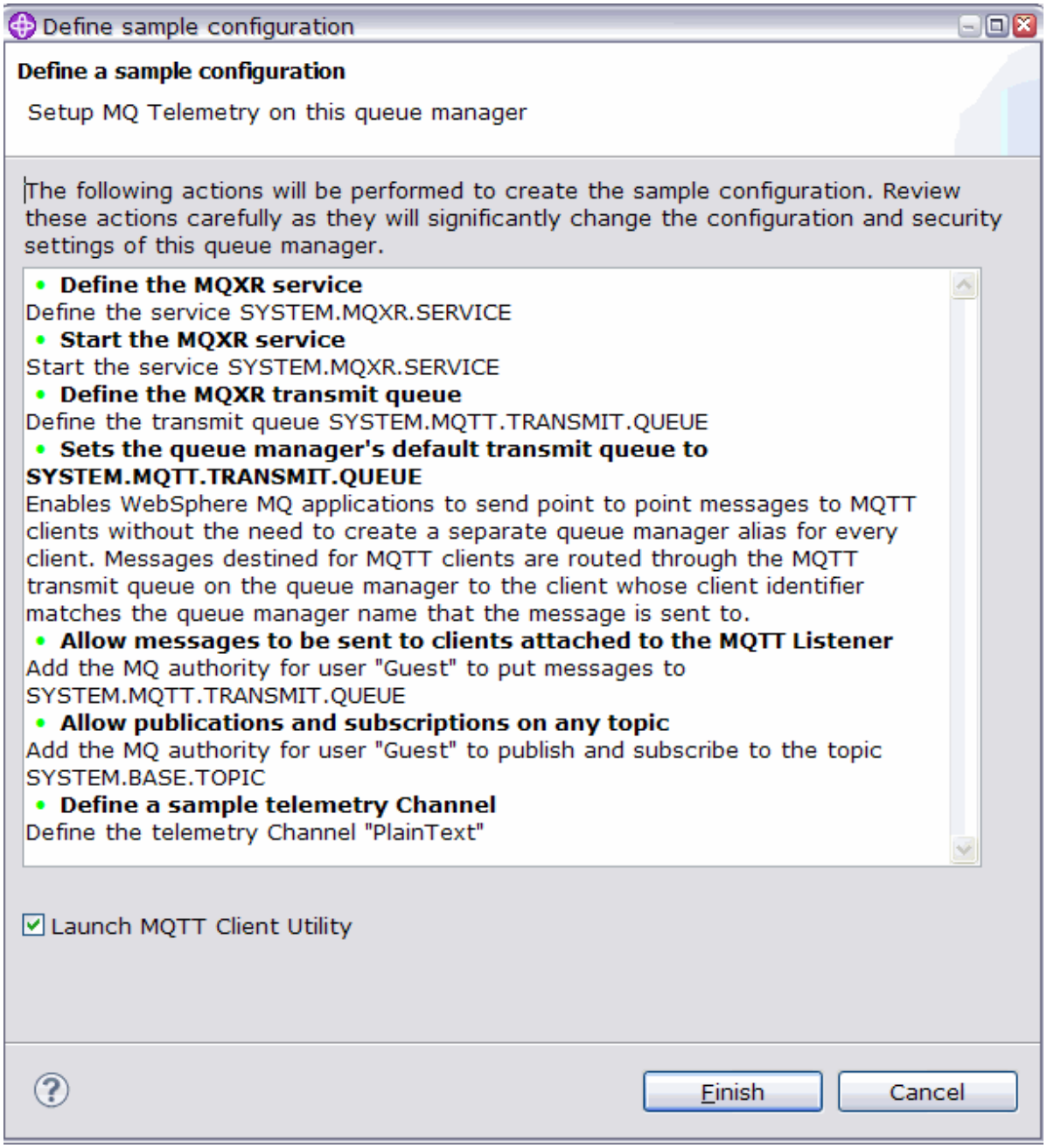
O IBM WebSphere MQ Explorer cria o gerenciador de filas e o inicia.

4. Execute o assistente **Definir configuração de amostra** de Telemetria.
   a) Abra a pasta Telemetria para o gerenciador de filas.



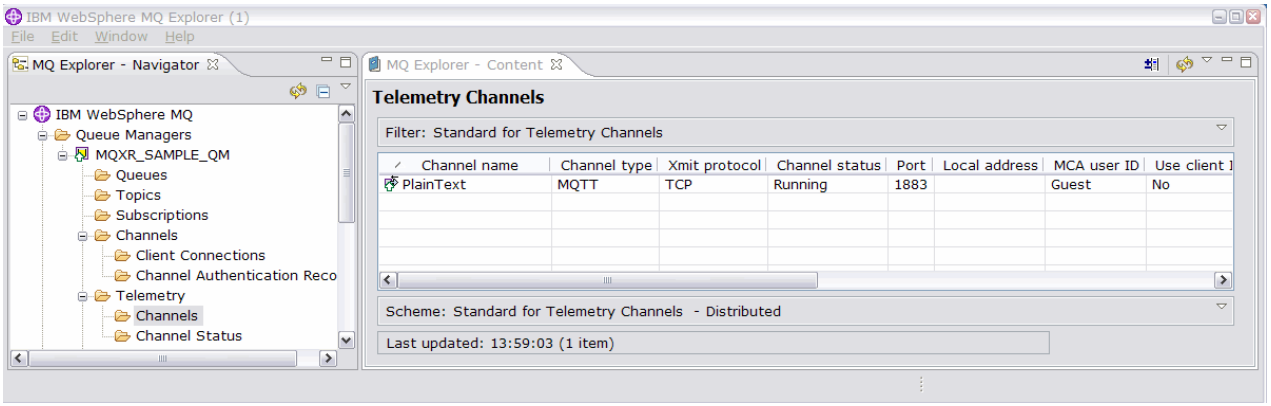
b) Clique em **Definir configuração de amostra** para iniciar o assistente.

c) Clique em **Concluir** para criar o serviço de telemetria e execute o utilitário do cliente MQTT



Resultados

Abra a pasta Canais de telemetria para listar os canais de amostra.



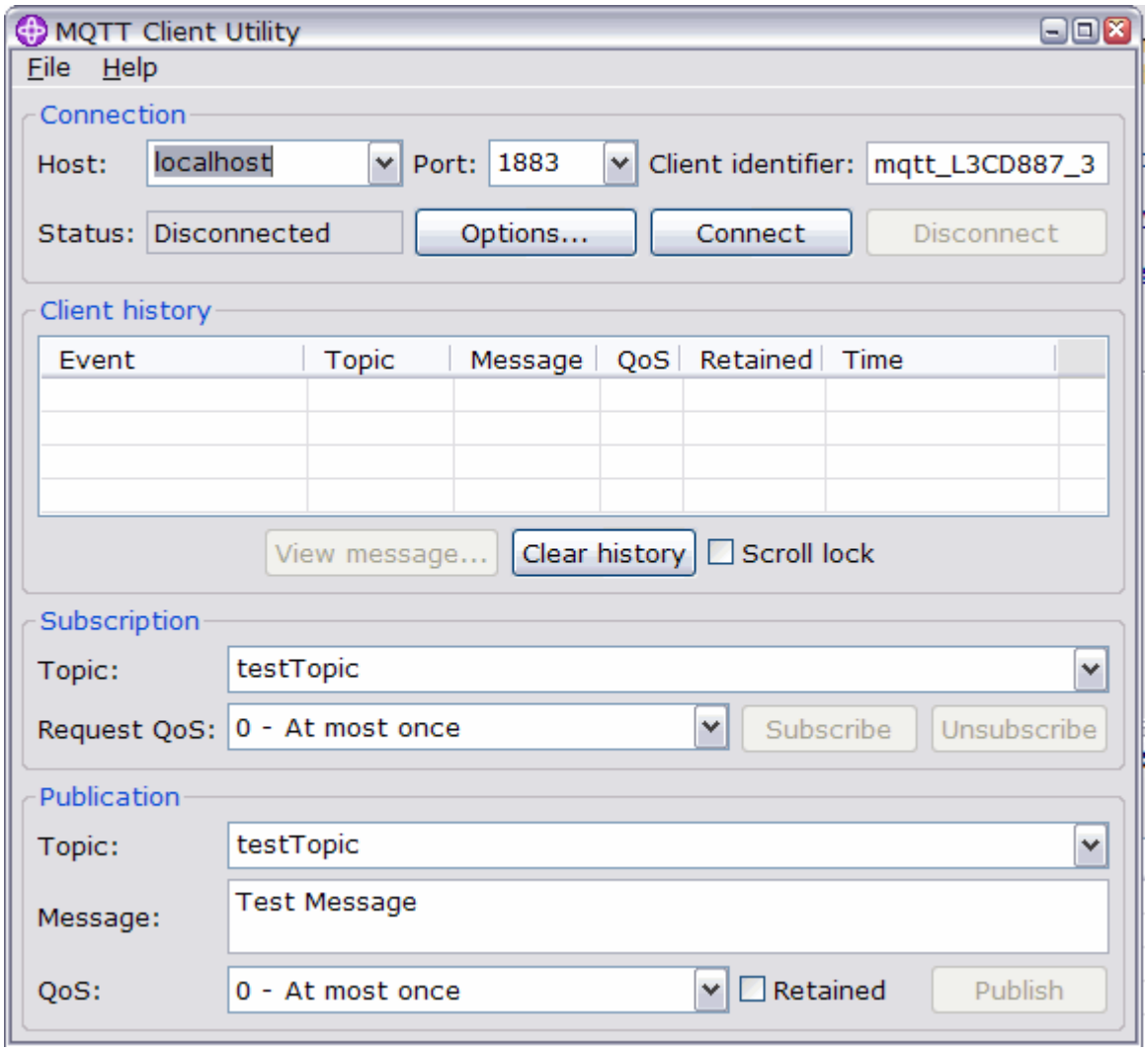
É possível modificar as propriedades deste canal, incluir e excluir canais nessa janela.

Como proceder a seguir

Teste a conexão executando o utilitário do cliente MQTT.

1. Para iniciar o utilitário do cliente, abra a pasta **Telemetria** e clique em **Executar utilitário do cliente MQTT** duas vezes.

   Duas janelas **Utilitário do cliente MQTT** abrem, idênticas mas para identificadores de cliente diferentes.



2. Clique em **Conectar** nas duas janelas.
3. Clique em **Assinar** nas duas janelas.
4. Clique em **Publicar** em qualquer janela. Os resultados são mostrados em Figura 29 na página 151

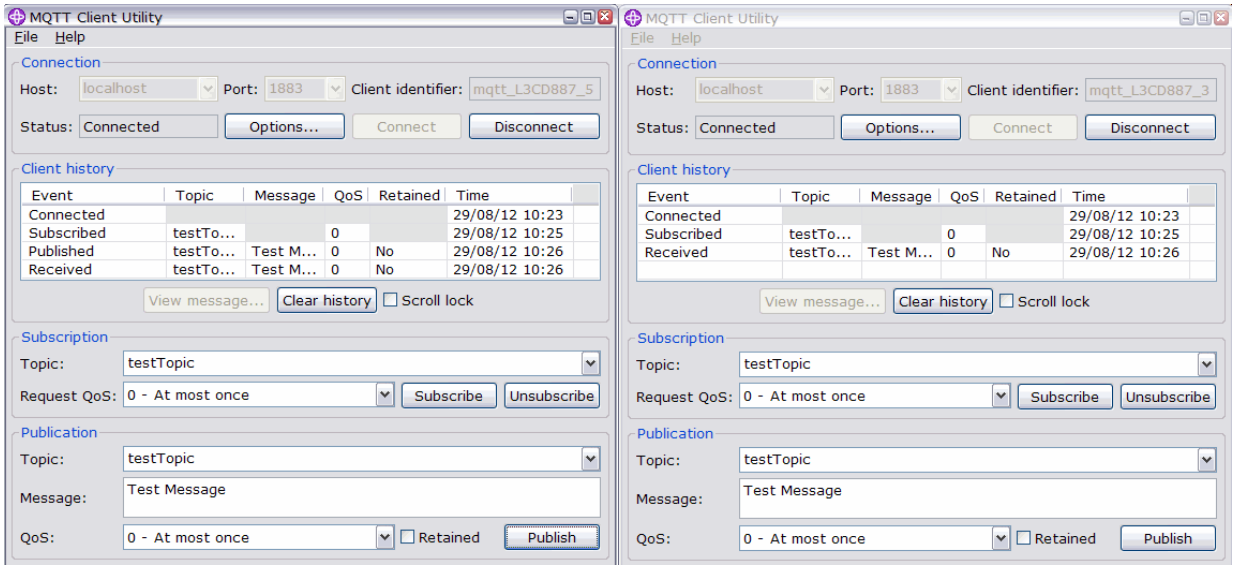


Figura 29. Resultados

5. Clique em **Desconectar** nas duas janelas.

O servidor agora está pronto para você testar seu app MQTT V3.1 .

Tarefas relacionadas

Configurando o serviço MQTT a partir da linha de comandos

Seguir estas instruções configuram o IBM WebSphere MQ usando a linha de comandos para executar os aplicativos IBM WebSphere MQ Telemetry de amostra. As etapas mostram como executar um script para criar um serviço do MQTT em um novo gerenciador de filas chamado `MQXR_SAMPLE_QM`.

Administrando o WebSphere MQ Telemetry

Informações relacionadas

WebSphere MQ Telemetry

Administrando o WebSphere MQ Telemetry com o WebSphere MQ Explorer

Desenvolvendo Aplicativos para WebSphere MQ Telemetry

Segurança

Segurança do WebSphere MQ Telemetry

IBM WebSphere MQ Daemon de telemetria para conceitos de dispositivos

O daemon de telemetria do IBM WebSphere MQ para dispositivos é um aplicativo cliente MQTT V3 avançado Use-o para armazenamento e encaminhamento de mensagens de outros clientes MQTT. Ele se conecta ao IBM WebSphere MQ como um cliente MQTT, mas também é possível conectar outros clientes MQTT a ele..

O daemon é um broker de publicação / assinatura Clientes MQTT V3 se conectam a ele para publicar e assinar tópicos, usando sequências de tópicos para publicar e filtros de tópicos para assinar. A sequência de tópicos é hierárquica com níveis de tópicos divididos por /. Os filtros de tópicos são sequências de tópicos que pode incluir curingas + de nível único e um curinga # de vários níveis como a última parte da sequência de tópicos.

Nota: Os curingas no daemon seguem as regras mais restritivas do WebSphere Message Broker, v6 IBM WebSphere MQ é diferente.. Ele suporta vários curingas de vários níveis; curinga pode representar qualquer número de níveis da hierarquia, em qualquer lugar na sequência de tópicos.

Vários clientes MQTT v3 se conectam ao daemon usando uma porta do listener. A porta do listener padrão é modificável. É possível definir várias portas listener e alocar namespaces diferentes para elas, consulte “Daemon do WebSphere MQ Telemetry para portas do listener de dispositivos” na página 159. O próprio daemon é um cliente MQTT v3. Configure uma conexão de ponte de daemon para conectar o daemon à porta do listener de outro daemon ou a um serviço de Telemetry (MQXR) do WebSphere MQ .

É possível configurar várias pontes do daemon de telemetria do WebSphere MQ para dispositivos. Use as pontes para conectar juntamente a uma rede de daemons que pode trocar publicações.

Cada ponte pode publicar e assinar tópicos em seu daemon local. Também é possível publicar e assinar tópicos em outro daemon, um broker de publicação/assinatura do WebSphere MQ ou qualquer outro broker MQTT v3 ao qual está conectado. Usando um filtro de tópicos, é possível selecionar as publicações para serem propagadas de um broker para outro. É possível propagar as publicações em qualquer direção. É possível propagar publicações do daemon local para cada um de seus brokers remotos conectados ou de qualquer um dos brokers conectados para o daemon local; consulte “IBM WebSphere MQ daemon de telemetria para pontes de dispositivos” na página 152.

IBM WebSphere MQ daemon de telemetria para pontes de dispositivos

Um IBM WebSphere MQ daemon de telemetria para dispositivos ponte conecta dois brokers de publicação / assinatura usando o protocolo MQTT v3 . A ponte propaga publicações de um broker para outro, em qualquer direção. Em uma extremidade há um daemon de telemetria do WebSphere MQ para a conexão de ponte de dispositivos e na outra pode ser um gerenciador de filas ou outro daemon Um gerenciador de filas é conectado à conexão de ponte usando um canal de telemetria. Um daemon está conectado à conexão de ponte usando um listener do daemon.

IBM WebSphere MQ O Daemon de telemetria para dispositivos suporta uma ou mais conexões simultâneas com outros brokers As conexões do daemon são chamadas de pontes e são definidas pelas entradas de conexão no arquivo de configuração do daemon. As conexões com o IBM WebSphere MQ são feitas usando canais de telemetria do IBM WebSphere MQ , conforme mostrado na figura a seguir:

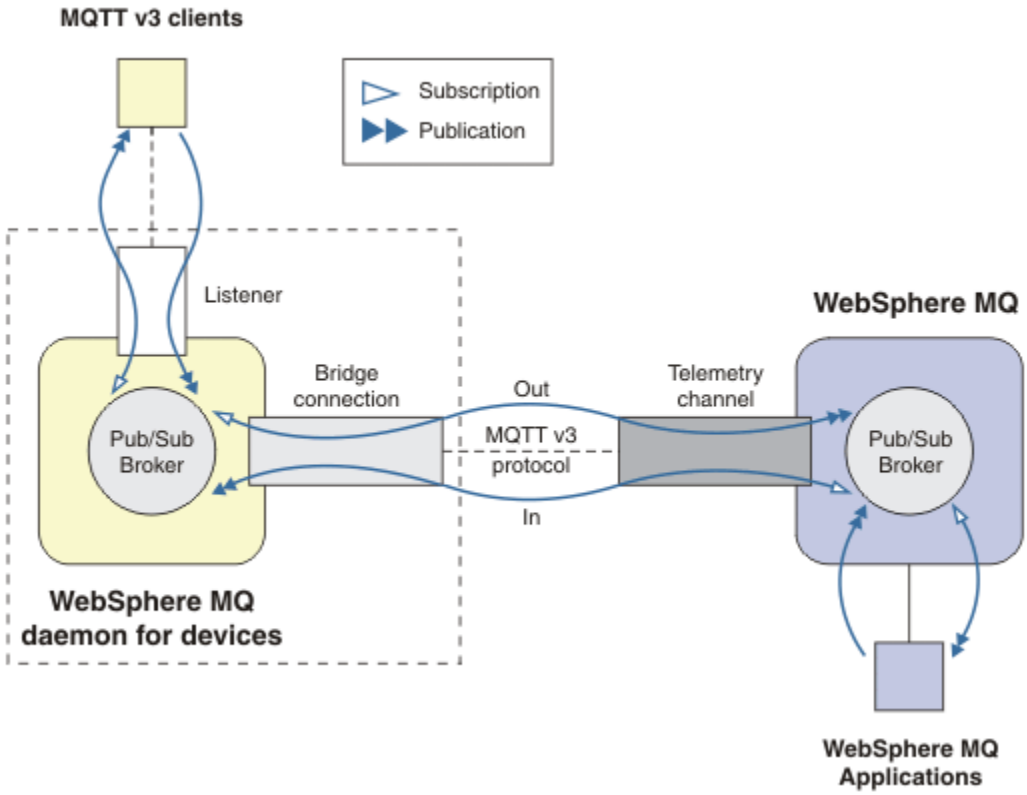


Figura 30. Conectando o IBM WebSphere MQ Telemetry daemon for devices a IBM WebSphere MQ

Uma ponte conecta o daemon em outro broker como um cliente MQTT v3. Os parâmetros bridge espelham os atributos de um cliente MQTT v3 .

Uma ponte é mais de uma conexão. Ela atua como um agente de publicação e assinatura situado entre dois brokers de publicação/assinatura. O broker local é o IBM WebSphere MQ daemon de Telemetria

para dispositivos e o broker remoto é qualquer broker de publicação / assinatura que suporte o protocolo MQTT v3 Geralmente o broker remoto é outro daemon ou IBM WebSphere MQ.

A tarefa da ponte é propagar publicações entre os dois brokers. A ponte é bidirecional. Ela propaga publicações em qualquer direção. O Figura 30 na página 152 ilustra a maneira como a ponte conecta o daemon do IBM WebSphere MQ Telemetry para dispositivos ao IBM WebSphere MQ O “Configurações de tópico de exemplo para a ponte” na página 153 usa exemplos para ilustrar como usar o parâmetro topic para configurar a ponte.

As setas Entrada e Saída em Figura 30 na página 152 indicam a bidirecionalidade da ponte Em uma extremidade da seta, uma assinatura é criada. As publicações que correspondem à assinatura são publicadas para o broker na extremidade oposta da seta. A seta é rotulada de acordo com o fluxo de publicações. Fluxo de publicações In para o daemon e Out a partir do daemon. A importância dos rótulos é que eles são usados na sintaxe de comando. Lembre-se de que In e Out se referem ao lugar em que o fluxo de publicações está e não ao lugar em que a assinatura é enviada.

Outros clientes, aplicativos ou brokers podem estar conectados ao IBM WebSphere MQ ou ao daemon de telemetria do WebSphere MQ para dispositivos Eles publicam e assinam tópicos no broker ao qual eles estão conectados. Se o broker for IBM WebSphere MQ, os tópicos poderão ser armazenados em cluster ou distribuídos e não serão explicitamente definidos no gerenciador de filas locais.

Usos de pontes

Conectar daemons juntos usando conexões de ponte e listeners. Conecte daemons e gerenciadores de filas juntos usando conexões de ponte e canais de telemetria. Ao conectar vários brokers juntos, será possível criar loops. Cuidado: publicações podem circular incessantemente ao redor de um loop de brokers, não detectado.

Algumas das razões para usar daemons vinculados ao IBM WebSphere MQ são as seguintes:

Reduza o número de conexões do cliente MQTT para WebSphere MQ
Usando uma hierarquia de daemons é possível conectar muitos clientes ao WebSphere MQ; mais clientes do que o número que um único gerenciador de filas pode conectar de uma vez.

Armazenar e encaminhar mensagens entre clientes MQTT e WebSphere MQ
Você pode usar armazenamento e encaminhamento para evitar manter conexões contínuas entre clientes e IBM WebSphere MQ, se os clientes não tiverem seu próprio armazenamento. É possível usar diversos tipos de conexões entre o cliente MQTT e o WebSphere MQ. Consulte Conceitos e cenários de telemetria para monitoramento e controle

Filtrar as publicações trocadas entre clientes MQTT e WebSphere MQ
Geralmente, as publicações se dividem em mensagens processadas localmente e mensagens que envolvem outros aplicativos. As publicações locais podem incluir fluxos de controle entre sensores e atuadores e as publicações remotas incluem solicitações para leituras, status e comandos de configuração.

Mude os espaços de tópicos de publicações
Evite sequências de tópicos a partir de clientes conectados às portas de listener diferentes de uma colisão entre si. O exemplo usa o daemon para rotular as leituras do medidor provenientes de diferentes construções; consulte Separando os espaços de tópico de diferentes grupos de clientes

Configurações de tópico de exemplo para a ponte

Publique tudo para o broker remoto - usando padrões

A direção padrão é chamada out e a ponte publica os tópicos para o broker remoto. O parâmetro topic controla quais tópicos são propagados usando filtros de tópicos.

A ponte usa o parâmetro topic no Figura 31 na página 154 para assinar tudo publicado para o daemon local pelos clientes MQTT ou por outros brokers. A ponte publica os tópicos para o broker remoto conectado pela ponte.

```
connection Daemon1
topic #
```

Figura 31. Publique tudo para o broker remoto

Publique tudo para o broker remoto - explícito

A configuração topic no fragmento de código a seguir fornece o mesmo resultado que é usado pelos padrões. A única diferença é que o parâmetro **direction** é explícito. Use a direção `out` para assinar o broker local, o daemon e publicar no broker remoto. As publicações criadas no daemon local que a ponte foi assinada, são publicadas no broker remoto.

```
connection Daemon1
topic # out
```

Figura 32. Publique tudo para o broker remoto - explícito

Publique tudo para o broker local

Em vez de usar a direção `out`, é possível configurar a direção oposta, `in`. O fragmento de código a seguir configura a ponte para assinar tudo publicado no broker remoto conectado pela ponte. A ponte publica os tópicos para o broker local, o daemon.

```
connection Daemon1
topic # in
```

Figura 33. Publique tudo para o broker local

Publique tudo a partir do tópico de exportação no broker local para o tópico de importação no broker remoto

Use dois parâmetros do tópicos adicionais, **local_prefix** e **remote_prefix**, para modificar o filtro de tópicos, *#* nos exemplos anteriores. Um parâmetro é usado para modificar o filtro de tópicos usado na assinatura e o outro parâmetro é usado para modificar o tópico ao qual a publicação é publicada. O efeito é para substituir o início da sequência de tópicos usada em um broker com outra sequência de tópicos no outro broker.

Dependendo da direção do comando de tópicos, o significado de **local_prefix** e **remote_prefix** será revertido. Se a direção for `out`, o padrão, **local_prefix** será usado como parte da assinatura do tópico e **remote_prefix** substituirá a parte da sequência de tópicos **local_prefix** na publicação remota. Se a direção for `in`, **remote_prefix** se tornará parte da assinatura remota e **local_prefix** substituirá a parte da sequência de tópicos **remote_prefix**.

A primeira parte de uma sequência de tópicos é muitas vezes considerada como a definição de um espaço de tópico. Use os parâmetros adicionais para mudar o espaço de tópico ao qual um tópico é publicado. Você pode fazer isto para evitar que o tópico que está sendo propagado colida com outro ao qual o tópico está no broker de destino ou para remover uma sequência de tópicos do ponto de montagem.

Como um exemplo, no fragmento de código a seguir, todas as publicações para a sequência de tópicos `export/#` no daemon são republicadas para `import/#` no broker remoto.

```
topic # out export/ import/
```

Figura 34. Publique tudo a partir do tópico de exportação no broker local para o tópico de importação no broker remoto

Publique tudo para o tópico de importação no broker local do tópico de exportação no broker remoto

O fragmento de código a seguir mostra a configuração revertida; a ponte assina tudo que foi publicado com a sequência de tópicos `export/#` no broker remoto e o publica para `import/#` no broker local.

```
connection Daemon1
topic # in import/ export/
```

Figura 35. Publique tudo para o tópico de importação no broker local do tópico de exportação no broker remoto

Publique tudo a partir do ponto de montagem 1884/ para o broker remoto com a sequência de tópicos original

No fragmento de código a seguir, a ponte assina tudo que foi publicado pelos clientes conectados ao ponto de montagem 1884/ no daemon local. A ponte publica tudo que foi publicado para o ponto de montagem no broker remoto. A sequência do ponto de montagem 1884/ é removida a partir dos tópicos publicados no broker remoto. O *local_prefix* é o mesmo que a sequência do ponto de montagem 1884/ e o *remote_prefix* é uma sequência em branco.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Figura 36. Publique tudo a partir do ponto de montagem 1884/ para o broker remoto com as sequências de tópicos originais.

Separando os espaços de tópicos de clientes diferentes conectados a diferentes daemons

Suponha que um aplicativo é gravado para medidores de energia elétrica para publicar as leituras de medidor para uma construção. As leituras são publicados usando os clientes MQTT em um daemon hospedado na mesma construção. O tópico selecionado para as publicações é `power`. O mesmo aplicativo é implementado em um número de construções em um complexo. Para o site de monitoramento e o armazenamento de dados, as leituras a partir de todas as construção são agregadas usando conexões de ponte. As conexões vinculam os daemons de construção ao WebSphere MQ em um local central..

Um aplicativo cliente idêntico é usado em todas as construções. Este aplicativo publica para o tópico `power`. No entanto, os dados devem ser diferenciados por construção. Isso é realizado pelo daemon para cada construção, que inclui o número de construção como um prefixo para o nome do tópico. A ponte da primeira construção no complexo usa o prefixo `meters/building01/`, a partir da construção dois, o prefixo é `meters/building02/`. As leituras a partir de outras construções seguem o mesmo padrão. WebSphere MQ portanto recebe as leituras com tópicos como `meters/building01/power`.

O arquivo de configuração para cada daemon tem uma instrução de tópico que segue o padrão no fragmento de código a seguir:

```
connection Daemon1
topic power out "" meters/building01/
```

Figura 37. Separe os espaços de tópicos de clientes conectados a diferentes daemons

No fragmento de código anterior, a sequência vazia é um item temporário para o parâmetro local_prefix não usado

Nota: Este exemplo é de alguma forma artificial e destinado apenas como uma ilustração. Na prática, o espaço de tópico que o aplicativo publica deve ser configurável.

Separe os espaços de tópicos de clientes conectados ao mesmo daemon

Suponha que um daemon único é usado para conectar todos os medidores de energia. Supondo que no aplicativo pode ser configurado para se conectar a portas diferentes, você pode distinguir as construções, conectando os medidores a partir de construções diferentes a diferentes portas de listener, como no fragmento de código a seguir. Novamente, o exemplo é complicado; ele ilustra como os pontos de montagem podem ser usados.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Figura 38. Separe os espaços de tópicos de clientes conectados ao mesmo daemon

Remapear diferentes tópicos para publicações fluem em ambas as direções

Na configuração no fragmento de código a seguir, a ponte assina o único tópico b no broker remoto e encaminha publicações sobre b ao daemon local, mudando o tópico para a. A ponte também assina o único tópico x no broker local e encaminha publicações sobre x para o broker remoto, mudando o tópico para y.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Figura 39. Remapear diferentes tópicos para publicações fluem em ambas as direções

Um ponto importante sobre este exemplo é que diferentes tópicos são assinados e publicados em ambos os brokers. Os espaços de tópicos em ambos os brokers são desconectados.

Remapear os mesmos tópicos para publicações fluem em ambas as direções (em loop)

Diferente do exemplo anterior, a configuração no Figura 40 na página 157, em geral, resulta em um loop. Na instrução de tópico `topic "" in a b`, a ponte assina b remotamente e publica a localmente. Na outra instrução de tópico, a ponte assina a localmente e publica b remotamente. A mesma configuração pode ser gravada como mostrado em Figura 41 na página 157.

O resultado geral é que, se um cliente publicar no b remotamente, a publicação será transferida para o daemon local como uma publicação no tópico a No entanto, ao ser publicado pela ponte para o daemon local no tópico a, a publicação corresponde à assinatura feita pela ponte para o tópico local a. A assinatura é `topic "" out a b`.. Como resultado, a publicação é transferida de volta para o broker remoto como uma publicação no tópico b A ponte agora está inscrita no tópico remoto be o ciclo começa novamente.

Alguns brokers implementam a detecção de loop para evitar que o loop aconteça. Mas o mecanismo de detecção do loop deverá funcionar quando diferentes tipos de brokers forem ligados. A detecção de loop não funcionará se o WebSphere MQ estiver vinculado ao daemon do WebSphere MQ Telemetry para dispositivos Ele funcionará se dois daemon de Telemetria do IBM WebSphere MQ para dispositivos forem vinculados juntos. Por padrão, a detecção de loop está ativada. Consulte try_private

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Figura 40. !Remapear os mesmos tópicos para publicações que fluem nas duas direções

```
connection Daemon1
topic "" both a b
```

Figura 41. !Remapeie os mesmos tópicos para publicações fluindo nas duas direções, usando `both`.

A configuração no Figura 39 na página 156 é a mesma que Figura 40 na página 157.

Disponibilidade de conexões de ponte IBM WebSphere MQ Telemetry daemon for devices

Configure diversos endereços de conexão de ponte IBM WebSphere MQ Telemetry daemon for devices para se conectar ao primeiro broker remoto disponível. Se o broker for um gerenciador de filas com várias instâncias, forneça ambos os endereços TCP/IP. Configure uma conexão primária para se conectar ou se reconectar ao servidor principal, quando ele estiver disponível.

O parâmetro de ponte de conexão, addresses é uma lista de endereços de soquete TCP/IP. A ponte tenta se conectar ao endereço de cada vez, até que ele faça uma conexão bem-sucedida. Os parâmetros de conexão round_robin e start_type controlam como os endereços serão usados após uma conexão bem-sucedida ser feita.

Se start_type for auto, manual ou lazy então, se a conexão falhar, a ponte tentará se reconectar. Ele usa cada endereço, por vez, com cerca de 20 segundos de atraso entre cada tentativa de conexão. Se `start_type` for once, então, se a conexão falhar, a ponte não tentará se reconectar automaticamente.

Se round_robin for true, as tentativas de conexão de ponte começarão com o primeiro endereço na lista e tentará cada endereço na lista em ordem. Ele iniciará no primeiro endereço novamente, quando a lista estiver esgotada. Se houver apenas um endereço na lista, ele tentará novamente a cada 20 segundos.

Se round_robin for false, o primeiro endereço na lista, que é chamado de servidor principal, será dado preferência. Se a primeira tentativa de se conectar ao servidor principal falhar, a ponte continuará a tentativa de se reconectar ao servidor principal em segundo plano. Ao mesmo tempo, a ponte tenta a conexão usando outros endereços na lista. Quando o segundo plano tentar se conectar ao servidor principal com sucesso, a ponte se desconectará da conexão atual e alternará para a conexão do servidor principal.

Se uma conexão for desconectada voluntariamente, por exemplo, emitindo um comando **`connection_stop`**, em seguida, se a conexão for reiniciada, ela tentará usar o mesmo endereço novamente. Se a conexão for desconectada devido a uma falha ao se conectar ou para o broker remoto eliminando a conexão, a ponte aguardará 20 segundos. Ela tentará se conectar ao próximo endereço na lista ou ao mesmo endereço, se houver apenas um endereço na lista.

Conectando-se a um gerenciador de filas com várias instâncias

Em uma configuração do gerenciador de filas com várias instâncias, o gerenciador de filas é executado em dois servidores diferentes com endereços IP diferentes. Geralmente, os canais de telemetria estão

configurados sem um endereço IP específico. Eles são configurados apenas com um número de porta. Quando o canal de telemetria for iniciado, por padrão, ele selecionará o primeiro endereço de rede disponível no servidor local.

Configure o parâmetro addresses da conexão de ponte com os dois endereços IP usados pelo gerenciador de filas. Configure round_robin como true..

Se a instância ativa do gerenciador de filas falhar, o gerenciador de filas alternará para a instância em espera. O daemon detecta que a conexão com a instância ativa foi interrompida e tenta se reconectar à instância em espera. Ele usa o outro endereço IP na lista de endereços configurados para a conexão de ponte.

O gerenciador de filas ao qual a ponte se conecta ainda é o mesmo gerenciador de filas. O gerenciador de filas recupera seu próprio estado. Se cleansession for configurado como false, a sessão de conexão de ponte será restaurada para o mesmo estado que antes do failover A conexão será retomada após um atraso. As mensagens com a qualidade de serviço "pelo menos uma vez" ou "no máximo uma vez" não são perdidas e as assinaturas continuam a funcionar.

O tempo de reconexão depende do número de canais e clientes reiniciados ao iniciar a instância em espera e quantas mensagens estavam em andamento. A conexão de ponte poderá tentar se reconectar a ambos os endereços IP um número de vezes antes da conexão ser restabelecida.

Não configure um canal de telemetria do gerenciador de filas com várias instâncias com um endereço IP específico. O endereço IP é válido apenas em um servidor.

Se você estiver usando uma solução de alta disponibilidade alternativa, que gerencia o endereço IP, isso poderá estar correto para configurar um canal de telemetria com um endereço IP específico.

cleansession

Uma conexão de ponte é uma sessão de cliente MQTT v3. É possível controlar se uma conexão inicia uma nova sessão ou se restaura uma sessão existente. Se ela restaurar uma sessão existente, a conexão de ponte preservará as assinaturas e as publicações retidas da sessão anterior.

Não configure cleansession como false se endereços listar vários endereços IP e os endereços IP se conectarem a canais de telemetria hospedados por diferentes gerenciadores de filas, ou a diferentes daemons de telemetria O estado da sessão não é transferido entre os gerenciadores de filas ou os daemons. Ao tentar reiniciar uma sessão existente em um gerenciador de filas ou daemon diferente resultará em uma nova sessão sendo iniciada. Mensagens indeterminadas são perdidas e assinaturas podem não se comportar conforme esperado.

notifications

Um aplicativo pode acompanhar se a conexão de ponte está em execução usando as notificações. Uma notificação é uma publicação que tem o valor `1` conectado ou `0` desconectado. Ela é publicada para *topicString* definido pelo parâmetro `notification_topic`. O valor padrão de *topicString* é $SYS/`broker/connection/` *clientIdentifier*`/state`. O padrão *topicString* contém o prefixo $SYS. Assine os tópicos iniciados com $SYS, definindo um filtro de tópicos iniciados com $SYS. O filtro de tópicos *#*, assina a tudo, não assina os tópicos iniciados com $SYS no daemon. Considere $SYS como a definição de um espaço de tópico do sistema especial distinta do espaço de tópico do aplicativo.

Notificações ativam IBM WebSphere MQ Telemetry daemon for devices para notificar clientes MQTT quando uma ponte está conectada ou desconectada.

keepalive_interval

O parâmetro de conexão de ponte keepalive_interval configura o intervalo entre a ponte enviando um ping do TCP/IP para o servidor remoto. O intervalo padrão é de 60 segundos. O ping impede que a sessão TCP/IP seja fechada pelo servidor remoto ou por um firewall, que detecta um período de inatividade na conexão.

clientid

Uma conexão de ponte é uma sessão do cliente MQTT v3 e possui um `clientIdentifier` configurado pelo parâmetro de conexão de ponte clientid. Se você pretende que as reconexões retomem a uma sessão anterior configurando o parâmetro cleansession como false, o `clientIdentifier` usado em cada sessão deverá ser o mesmo. O valor padrão de clientid é *hostname.connectionName*, que permanece o mesmo.

Instalação, verificação, configuração e controle do daemon do WebSphere MQ Telemetry para dispositivos

A instalação, configuração e o controle do daemon são baseados em arquivos.

Instale o daemon copiando o kit de desenvolvimento de software no dispositivo no qual você pretende executar o daemon.

Como exemplo, execute o utilitário do cliente MQTT e conecte-se ao daemon do WebSphere MQ Telemetry para dispositivos como o broker de publicação / assinação; consulte Use o daemon do WebSphere MQ Telemetry para dispositivos como o broker de publicação / assinatura.

Configure o daemon criando um arquivo de configuração; consulte WebSphere MQ Telemetry daemon para o arquivo de configuração de dispositivos.

Controle um daemon em execução, criando comandos no arquivo, `amqtdd.upd`. A cada 5 segundos, o daemon lê o arquivo, executa os comandos e exclui o arquivo; consulte o WebSphere MQ Telemetry para o arquivo de comando de dispositivos

Daemon do WebSphere MQ Telemetry para portas do listener de dispositivos

Conecte os clientes MQTT V3 ao daemon WebSphere MQ Telemetry para dispositivos que usam portas listener. É possível qualificar uma porta do listener com um ponto de montagem e um número máximo de conexões.

Um porta do listener deve corresponder ao número da porta especificado no método `connect(serverURI)` do cliente MQTT de um cliente conectado a essa porta. Ela é padronizada no cliente e no daemon para 1883.

É possível mudar a porta padrão para o daemon, configurando a definição global `port` no arquivo de configuração do daemon. É possível configurar as portas específicas, incluindo uma definição `listener` para o arquivo de configuração do daemon.

Para cada porta do listener, diferente da porta padrão, é possível especificar um ponto de montagem para isolar os clientes. Os clientes conectados a uma porta com um ponto de montagem são isolados de outros clientes; consulte “Daemon do WebSphere MQ Telemetry para pontos de montagem de dispositivos” na página 160.

É possível limitar o número de clientes que pode se conectar a qualquer porta. Configure a definição global `max_connections` para limitar as conexões com a porta padrão ou qualificar cada porta do listener com `max_connections`.

exemplo

Um exemplo de um arquivo de configuração que muda a porta padrão de 1883 para 1880 e limita as conexões com a porta 1880 para 10000. As conexões à porta 1884 são limitadas a 1000. Os clientes conectados à porta 1884 são isolados dos clientes conectados a outras portas.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Daemon do WebSphere MQ Telemetry para pontos de montagem de dispositivos

É possível associar um ponto de montagem a uma porta listener usada pelos clientes MQTT para se conectar a um daemon do WebSphere MQ Telemetry para dispositivos. Um ponto de montagem isola as publicações e assinaturas trocadas pelos clientes MQTT usando uma porta do listener a partir de clientes MQTT conectados a uma porta do listener diferente.

Os clientes conectados a uma porta do listener com um ponto de montagem nunca podem trocar diretamente os tópicos com clientes conectados a quaisquer outras portas do listener. Os clientes conectados a uma porta do listener sem um ponto de montagem podem publicar ou assinar tópicos de qualquer cliente. Os clientes não estão cientes se eles estão conectados por meio de um ponto de montagem ou não; isso não faz diferença para as sequências de tópicos criadas pelos clientes.

Um ponto de montagem é uma sequência de texto prefixada na sequência de tópicos de publicações e assinaturas. Ela é prefixada para todas as sequências de tópicos criadas por clientes conectados a porta do listener com um ponto de montagem. A sequência de texto é removida de todas as sequências de tópicos enviadas para clientes conectados à porta do listener.

Se uma porta do listener não tiver nenhum ponto de montagem, as sequências de tópicos das publicações e assinaturas criadas e recebidas por clientes conectados à porta não serão alteradas.

Criar sequências de ponto de montagem com um final `/`. Dessa maneira o ponto de montagem é o tópico-pai da árvore de tópicos para o ponto de montagem.

exemplo

Um arquivo de configuração contém as seguintes portas listener:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Um cliente, conectado à porta 1883, cria uma assinatura para `MyTopic`. O daemon registra a assinatura, como `1883/MyTopic`. Outro cliente conectado à porta 1883 publica uma mensagem no tópico, `MyTopic`. O daemon muda a sequência de tópicos para `1883/MyTopic` e procura as assinaturas correspondentes. O assinante na porta 1883 recebe a publicação com a sequência de tópicos original `MyTopic`. O daemon removeu o prefixo do ponto de montagem da sequência de tópicos.

Outro cliente, conectado à porta 1884, também publica no tópico `MyTopic`. Desta vez, a daemon registra o tópico como `1884/MyTopic`. O assinante na porta 1883 não recebe a publicação, porque o ponto de montagem diferente resulta em uma assinatura com uma sequência de tópicos diferente.

Um cliente, conectado à porta 1885, publica no tópico, `1883/MyTopic`. O daemon não muda a sequência de tópicos. O assinante na porta 1883 recebe a publicação para `MyTopic`.

Daemon do WebSphere MQ Telemetry para qualidade de serviço de dispositivos, assinaturas duráveis e publicações retidas..

As configurações de qualidade de serviço se aplicam apenas a um daemon em execução. Se um daemon parar, seja em um modo controlado ou por causa de uma falha, o estado de mensagens em andamento será perdido. A entrega de uma mensagem pelo menos uma vez ou no máximo uma vez, não poderá ser garantida se o daemon parar. O daemon WebSphere MQ Telemetry para dispositivos suporta persistência limitada. Configure o parâmetro de configuração **`retained_persistence`** para salvar as publicações e assinaturas retidas quando o daemon for encerrado.

Diferente do WebSphere MQ, o daemon do WebSphere MQ Telemetry para dispositivos não registra dados persistentes. O estado de sessão, o estado da mensagem e as publicações retidas não serão salvos de forma transacional. Por padrão, o daemon descartará todos os dados, quando ele parar. É possível configurar uma opção para as publicações retidas e assinaturas de ponto de verificação periodicamente.

O status da mensagem sempre será perdido quando o daemon parar. Todas as publicações não retidas são perdidas.

Configure a opção de configuração do daemon, `Retained_persistence` como `true`, para salvar as publicações retidas periodicamente para um arquivo. Quando o daemon for reiniciado, as publicações retidas que foram salvas automaticamente por último serão restabelecidas. Por padrão, as mensagens retidas criadas por clientes não serão restabelecidas quando o daemon for reiniciado.

Configure a opção de configuração do daemon, `Retained_persistence` como `true`, para salvar as assinaturas criadas em uma sessão persistente periodicamente para um arquivo. Se `Retained_persistence` for configurado como `true`, as assinaturas que os clientes criam em uma sessão com `CleanSession` configurado como `false`, uma "sessão persistente", serão restauradas. O daemon restaurará as assinaturas quando for reiniciado, o qual começará a receber publicações. O cliente receberá as publicações quando ele reiniciar com `CleanSession` como `false`. Por padrão, o estado de sessão do cliente não será salvo quando um daemon parar e as assinaturas não forem restauradas, mesmo se o cliente configurar `CleanSession` como `false`.

`Retained_persistence` é um mecanismo de salvamento automático. Ele não pode salvar as publicações ou assinaturas retidas mais recentes. É possível mudar a frequência com que publicações e assinaturas retidas são salvas. Configure o intervalo entre os salvamentos ou o número de mudanças entre os salvamentos, usando as opções de configuração `autosave_on_changes` e `autosave_interval`.

Configuração de exemplo para persistência configuração

```
# 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 do WebSphere MQ Telemetry para segurança de dispositivos

O daemon do WebSphere MQ Telemetry para dispositivos pode autenticar clientes que se conectam a ele, usar credenciais para conectar a outros brokers e controlar o acesso a tópicos. A segurança que o daemon fornece é limitada por ser construído usando o cliente C do WebSphere MQ Telemetry , que não fornece suporte SSL Consequentemente, as conexões para e do daemon não são criptografadas e não podem ser autenticadas usando certificados.

Por padrão, nenhuma segurança está ativada.

Autenticação de clientes

Os clientes MQTT podem configurar uma senha e um nome de usuário usando os métodos `MqttConnectOptions.setUserName` e `MqttConnectOptions.setPassword`.

Autenticar um cliente que se conecta ao daemon, verificando a senha e o nome do usuário fornecidos por um cliente com relação às entradas no arquivo de senha. Para ativar a autenticação, crie um arquivo de senha e configure o parâmetro `password_file` no arquivo de configuração do daemon; consulte password_file.

Configure o parâmetro `allow_anonymous` no arquivo de configuração do daemon para permitir que os clientes se conectem sem as senhas ou nomes dos usuários para se conectarem a um daemon que é a verificação de autenticação; consulte allow_anonymous. Se um cliente fornecer uma senha ou um nome do usuário, ele será sempre verificado com relação ao arquivo de senha, se o parâmetro `password_file` for configurado.

Configure o parâmetro `clientid_prefixes` no arquivo de configuração do daemon para limitar as conexões a clientes específicos. Os clientes devem ter `clientIdentifiers` iniciados com um dos prefixos listados no parâmetro `clientid_prefixes`; consulte clientid_prefixes.

Segurança de conexão de ponte

Cada daemon WebSphere MQ Telemetry para conexão de ponte de dispositivos é um cliente MQTT V3 . É possível configurar a senha e o nome do usuário para cada conexão de ponte como um parâmetro de conexão de ponte no arquivo de configuração do daemon; consulte username e password. Uma ponte pode, então, autenticar-se para um broker.

Controle de acesso dos tópicos

Se os clientes estiverem sendo autenticados, o daemon também poderá fornecer acesso de controle para tópicos para cada usuário. O daemon concede o controle de acesso com base na correspondência do tópico para o qual um cliente é a publicação ou assinatura com uma sequência de tópicos de acesso no arquivo de controle de acesso; consulte acl_file.

A lista de controle de acesso tem duas partes. A primeira parte controla o acesso para todos os clientes, incluindo clientes anônimos. A segunda parte tem uma seção para qualquer usuário no arquivo de senha. Ele lista o controle de acesso específico para cada usuário.

exemplo

Os parâmetros de segurança são mostrados no exemplo a seguir.

```
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. Arquivo de configuração do daemon

```
Fred:Fredpassword
Barney:Barneypassword
```

Figura 43. Password file, 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. Access control file, acl.txt

Clientes de resolução de problemas do MQTT

Procure uma tarefa de resolução de problemas para ajudá-lo a solucionar um problema com a execução de clientes MQTT.

Tarefas relacionadas

“Rastreando e Depurando o Cliente Java MQTT (Paho)” na página 171
O criador de logs padrão usa o recurso de criação de log Java padrão conhecido como `java.util.logging` (JSR47). É possível configurá-lo pelo uso de um arquivo de configuração ou programaticamente.

“Rastreando o cliente MQTT JavaScript” na página 174

É possível usar o cliente JavaScript para coletar rastreio alterando o aplicativo da web cliente para chamar métodos no objeto do cliente conectado.

“Rastreio do serviço de telemetria (MQXR)” na página 167
Siga estas instruções para iniciar um rastreio do serviço de telemetria, configurar os parâmetros que controlam o rastreio e localizar a saída de rastreio.

“Rastreando o Cliente Java MQTT v3” na página 168
Siga estas instruções para criar o rastreio de um cliente MQTT Java e controlar sua saída.

“Rastreando o cliente MQTT para C” na página 170
Configure a variável de ambiente `MQTT_C_CLIENT_TRACE` para rastrear um aplicativo C do cliente MQTT

“Resolução do problema: cliente MQTT não se conecta” na página 181
Resolva o problema de um programa do cliente MQTT com falha ao se conectar ao serviço de telemetria (MQXR).

“Resolução do problema: conexão do cliente MQTT eliminada” na página 183
Descubra o que está fazendo com que um cliente lance exceções inesperadas `ConnectionLost` após uma execução e conexão bem-sucedidas por um tempo curto ou longo.

“Resolução de problemas: mensagens perdidas em um aplicativo MQTT” na página 183
Resolva o problema de perda de uma mensagem. A mensagem não persistente, foi enviada ao lugar errado ou nunca foi enviada? Um programa cliente codificado incorretamente pode perder mensagens.

“Resolução de problema: serviço de telemetria (MQXR) não inicia” na página 185
Resolva o problema do serviço de telemetria (MQXR) que falha ao iniciar. Verifique a instalação do WebSphere MQ Telemetry e se nenhum arquivo está ausente, foi movido ou tem as permissões erradas. Verifique os caminhos usados pelo serviço de telemetria (MQXR) para localizar os programas de serviço de telemetria (MQXR).

“Resolução do problema: o módulo de login JAAS não é chamado pelo serviço de telemetria” na página 187
Descubra se o módulo de login JAAS não está sendo chamado pelo serviço de telemetria (MQXR) e configure o JAAS para corrigir o problema.

“Resolução de problemas: iniciando ou executando o daemon” na página 190
Consulte o daemon do Telemetry do IBM WebSphere MQ para o log do console de dispositivos, ative o rastreio ou use a tabela de sintomas neste tópico para solucionar problemas com o daemon.

“Resolvendo problema: clientes MQTT não se conectam ao daemon” na página 190
Os clientes não estão se conectando ao daemon, o daemon não está se conectando a outros daemons ou a um canal de telemetria do WebSphere MQ.

Referências relacionadas

“Local de logs de telemetria, logs de erro e arquivos de configuração” na página 164
Localize os logs, os logs de erro e os arquivos de configuração usados pelo IBM WebSphere MQ Telemetry.

“Códigos de razão do cliente Java MQTT v3” na página 166
Consulte as causas de códigos de razão em uma exceção do cliente Java MQTT v3 ou lançável.

“Requisitos do sistema para usar conjuntos de cifras SHA-2 com clientes do MQTT” na página 174
Para Java 6 de IBM, SR13 em diante, é possível usar conjuntos de cifras SHA-2 para proteger seus canais e aplicativos clientes do MQTT . No entanto, os conjuntos de cifras SHA-2 não são ativados por padrão até o Java 7 de IBM, SR4 em diante, portanto, em versões anteriores, deve-se especificar o conjunto necessário. Se você estiver executando um cliente MQTT com seu próprio JRE, é preciso garantir que ele suporta os conjuntos de cifras SHA-2. Para os aplicativos clientes usarem os conjuntos de cifras SHA-2, o cliente também deve configurar o contexto de SSL para um valor que suporte o Transport Layer Security (TLS) versão 1.2.

“Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL” na página 175
Há diferenças na capacidade entre navegadores diferentes, em plataformas diferentes. Entender essas diferenças ajuda você a configurar aplicativos, autoridades de certificação (CAs) e certificados de cliente

para se conectar usando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets.

Local de logs de telemetria, logs de erro e arquivos de configuração

Localize os logs, os logs de erro e os arquivos de configuração usados pelo IBM WebSphere MQ Telemetry.

Nota: Os exemplos são codificados para Windows. Altere a sintaxe para executar os exemplos em Linux

Logs do lado do servidor

O assistente de instalação do IBM WebSphere MQ Telemetry grava mensagens em seu log de instalação:

```
WMQ program directory\mqxr
```

O serviço de telemetria (MQXR) grava mensagens no log de erros do gerenciador de filas do WebSphere MQ e nos arquivos FDC no diretório de erro IBM WebSphere MQ :

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

Também grava um log para o serviço de telemetria (MQXR). O log exibe as propriedades com as quais o serviço é iniciado e os erros que localiza ao agir como um proxy para um cliente MQTT. Por exemplo, cancelar uma assinatura que o cliente não criou. O caminho do log é:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

A configuração de amostra de telemetria do IBM WebSphere MQ criada pelo IBM WebSphere MQ Explorer inicia o serviço de telemetria usando o comando **runMQXRService** O **runMQXRService** está em *WMQ Telemetry install directory*\bin Ele grava em:

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Modifique **runMQXRService** para exibir os caminhos configurados para o serviço de telemetria (MQXR) ou para ecoar a inicialização antes de iniciar o serviço de telemetria (MQXR).

Arquivos de configuração do lado do servidor

Canais de telemetria e serviço de telemetria (MQXR)

Restrição: O formato, o local, o conteúdo e a interpretação do arquivo de configuração do canal de telemetria podem mudar em futuras liberações. Deve-se usar o IBM WebSphere MQ Explorer para configurar canais de telemetria.

IBM WebSphere MQ O Explorer salva as configurações de telemetria no arquivo `mqxr_win.properties` no Windowse no arquivo `mqxr_unix.properties` no Linux Os arquivos de propriedades são salvos no diretório de configuração de telemetria:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Figura 45. Diretório de Configuração de Telemetria no Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figura 46. Diretório de configuração de telemetria no Linux

JVM

Configure as propriedades Java que são transmitidas como argumentos para o serviço de telemetria (MQXR) no arquivo, `java.properties`. As propriedades no arquivo são passadas diretamente para a JVM que está executando o serviço de telemetria (MQXR). Elas são transmitidas como

propriedades adicionais da JVM na linha de comando Java Propriedades configuradas na linha de comandos têm precedência sobre propriedades incluídas na linha de comandos a partir do arquivo `java.properties`.

Localize o arquivo `java.properties` na mesma pasta que as configurações de telemetria, consulte Figura 45 na página 164 e Figura 46 na página 164

Modifique `java.properties` especificando cada propriedade como uma linha separada. Formate cada propriedade exatamente como faria para passar a propriedade para a JVM como um argumento; por exemplo:

```
-Xmx1024m
-Xms1024m
```

JAAS

O arquivo de configuração JAAS é descrito em Configuração do canal de telemetria JAAS , que inclui o arquivo de configuração de amostra JAAS , JAAS.config, enviado com o IBM WebSphere MQ Telemetry.

Se configurar o JAAS, muito provavelmente irá escrever uma classe para autenticar usuários para substituir os procedimentos de autenticação JAAS padrão.

Para incluir sua classe `Login` no caminho da classe usado pelo caminho da classe de serviço de telemetria (MQXR), forneça um arquivo de configuração do WebSphere MQ `service.env` .

Configure o caminho de classe para seu `LoginModule` do JAAS em `service.env`. Não é possível usar a variável `%classpath%` em `service.env`. O caminho da classe em `service.env` é incluído no caminho de classe já configurado na definição do serviço de telemetria (MQXR).

Exiba os caminhos de classe que estão sendo usados pelo serviço de telemetria (MQXR), incluindo `echo set classpath` em `runMQXRService.bat`. A saída é enviada para `mqxr.stdout`.

O local padrão para o arquivo `service.env` é:

```
WMQ data directory\service.env
```

Substitua essas configurações por um arquivo `service.env` para cada gerenciador de filas em:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

Figura 47 na página 165 mostra um arquivo `service.env` de amostra para usar a amostra `LoginModule.class`.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Nota: `service.env` não deve conter variáveis. Substitua o valor real de *`WMQ Install Directory`*.

Figura 47. `service.env` de Amostra para Windows

Rastreio

Um engenheiro de serviço da IBM pode solicitar a configuração de um rastreio; consulte “Rastreio do serviço de telemetria (MQXR)” na página 167. Os parâmetros para configuração de rastreio são armazenados em dois arquivos:

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Arquivos de log do lado do cliente

A classe de persistência de arquivo padrão no cliente Java SE MQTT fornecido com o IBM WebSphere MQ Telemetry cria uma pasta com o nome: *clientIdentifier*`-tcp`*hostNameport* ou *clientIdentifier*`-ssl`*hostNameport* no diretório ativo do cliente. O nome da pasta informa `hostName` e `port` usados na tentativa de conexão. A pasta contém mensagens que foram armazenadas pela classe de persistência.. As mensagens são excluídas quando entregues com sucesso.

A pasta é excluída quando um cliente, com uma sessão limpa, termina.

Se o rastreio do cliente estiver ativado, o log não formatado será, por padrão, armazenado no diretório ativo do cliente. O arquivo de rastreio é chamado `mqtt-n.trc`

Arquivos de configuração do lado do cliente

Configure propriedades de rastreio e SSL para o cliente MQTT Java usando arquivos de propriedades Java ou configure as propriedades programaticamente. Passe as propriedades para o cliente MQTT Java usando o comutador JVM `-D` : por exemplo,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Consulte “Rastreando o Cliente Java MQTT v3” na página 168.Para obter links para a documentação da API do cliente para as bibliotecas do cliente MQTT, consulte Referência de programação do cliente MQTT..

Códigos de razão do cliente Java MQTT v3

Consulte as causas de códigos de razão em uma exceção do cliente Java MQTT v3 ou lançável.

Tabela 5. Códigos de razão do cliente Java MQTT v3

Código de razão	Value	Causa
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	O cliente já está conectado.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	O cliente já está desconectado.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Descartado quando uma tentativa para chamar o `MqttClient.disconnect` foi feita de dentro de um método no `MqttCallback`.
REASON_CODE_CLIENT_DISCONNECTING	32102	O cliente está atualmente desconectando e não pode aceitar nenhum novo trabalho.
REASON_CODE_CLIENT_EXCEPTION	0	O cliente encontrou uma exceção.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	O cliente não está conectado ao servidor.
REASON_CODE_CLIENT_TIMEOUT	32000	O cliente atingiu o tempo limite ao esperar por uma resposta do servidor.
REASON_CODE_FAILED_AUTHENTICATION	4	A autenticação com o servidor falhou devido a um nome do usuário ou senha inválido.
REASON_CODE_INVALID_CLIENT_ID	2	O servidor rejeitou o ID do cliente fornecido.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	A versão do protocolo solicitado não é suportada pelo servidor.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Erro interno, causado por nenhum novo ID de mensagens estar disponível.
REASON_CODE_NOT_AUTHORIZED	5	Não autorizado a executar a operação solicitada.

Tabela 5. Códigos de razão do cliente Java MQTT v3 (continuação)		
Código de razão	**Value**	**Causa**
REASON_CODE_SERVER_CONNECT_ERROR	32103	Não é possível conectar-se ao servidor.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	URI do servidor e SocketFactory fornecida não correspondem.
REASON_CODE_SSL_CONFIG_ERROR	32106	Erro de configuração SSL.
REASON_CODE_UNEXPECTED_ERROR	6	Ocorreu um erro inesperado.

Rastreio do serviço de telemetria (MQXR)

Siga estas instruções para iniciar um rastreio do serviço de telemetria, configurar os parâmetros que controlam o rastreio e localizar a saída de rastreio.

Antes de começar

O rastreio é uma função de suporte. Siga estas instruções se um engenheiro de serviço da IBM solicitar que você rastreie o serviço de telemetria (MQXR). A documentação do produto não documenta o formato do arquivo de rastreio nem como usá-lo para depurar um cliente.

Sobre esta tarefa

É possível usar os comandos IBM WebSphere MQ **strmqtrc** e **endmqtrc** para iniciar e parar o rastreio do IBM WebSphere MQ **strmqtrc** captura o rastreio para o serviço de telemetria (MQXR). Ao usar o **strmqtrc**, há um atraso de até dois segundos antes do rastreio de serviço de telemetria ser iniciado. Para obter informações adicionais sobre o rastreio do IBM WebSphere MQ , consulte Usando o rastreio
Com alternativa, é possível rastrear o serviço de telemetria (MQXR) usando o seguinte procedimento:

Procedimento

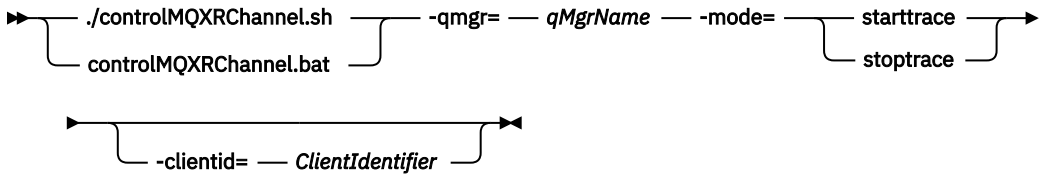
1. Configure as opções de rastreio para controlar a quantia de detalhes e o tamanho do rastreio. As opções se aplicam a um rastreio iniciado com o comando **strmqtrc** ou **controlMQXRChannel**.

   Configure as opções de rastreio nos arquivos a seguir:

   ```
   mqxrtrace.properties
   trace.config
   ```

   Os arquivos estão no diretório:

   - Em Windows, *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr
   - Em Linux, var/mqm/qmgrs/ *qMgrName*/mqxr
2. Abra uma janela de comando no diretório a seguir:
   - Em Windows sistemas, *WebSphere MQ installation directory*\mqxr\bin
   - Em Linux sistemas /opt/mqm/mqxr/bin
3. Execute o comando a seguir para iniciar um rastreio SYSTEM.MQXR.SERVICE:



Parâmetros obrigatórios

qmgr=*qmgrName*
Configure *qmgrName* para o nome do gerenciador de filas

mode=starttrace| stoptrace
Configure starttrace para iniciar o rastreio ou stoptrace para terminar o rastreio

Parâmetros opcionais

clientid=*ClientIdentifier*
Configure *ClientIdentifier* para o `ClientIdentifier` de um cliente. clientid filtra o rastreio para um único cliente. Execute o comando de rastreio diversas vezes para rastrear diversos clientes.

Por exemplo:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

Resultados

Para visualizar a saída de rastreio, acesse o diretório a seguir:

- Em Windows , *WebSphere MQ data directory*\trace
- No Linux, /var/mqm/trace.

Arquivos de rastreio são denominados mqxr_PPPPP.trc, em que PPPPP é o ID do processo.

Referências relacionadas

strmqtrc

Rastreando o Cliente Java MQTT v3

Siga estas instruções para criar o rastreio de um cliente MQTT Java e controlar sua saída.

Antes de começar

Este tópico é aplicável apenas ao IBM WebSphere MQ versão 7.5.0.0 Para obter informações sobre como rastrear o cliente Java para versões posteriores, consulte “Rastreando e Depurando o Cliente Java MQTT (Paho)” na página 171.

O rastreio é uma função de suporte. Siga estas instruções se um engenheiro de serviço IBM solicitar que você rastreie seu cliente MQTT Java. A documentação do produto não documenta o formato do arquivo de rastreio nem como usá-lo para depurar um cliente.

O rastreio funciona apenas para o cliente Java do WebSphere MQ Telemetry

Sobre esta tarefa

Nota: Os exemplos são codificados para o Windows Altere a sintaxe para executar os exemplos em Linux[2].

Procedimento

1. Crie um arquivo de propriedades Java que contenha a configuração de rastreio

   No arquivo de propriedades, especifique as seguintes propriedades opcionais. Se uma chave da propriedade for especificada mais de uma vez, a última ocorrência configurará a propriedade.

   a) com.ibm.micro.client.mqttv3.trace.outputName

[2] O Java usa o delimitador de caminho correto É possível codificar o delimitador em um arquivo de propriedades como '/' ou '\\'; '\' é o caractere de escape

O diretório no qual gravar o arquivo de rastreio. É padronizado para o diretório ativo do cliente. O arquivo de rastreio é chamado `mqtt-n.trc`.

```
java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
```

b) `com.ibm.micro.client.mqttv3.trace.count`

O número de arquivos de rastreio a ser gravado. O padrão é um arquivo de tamanho ilimitado.

```
java com.ibm.micro.client.mqttv3.trace.count=5
```

c) `com.ibm.micro.client.mqttv3.trace.limit`

O tamanho máximo de arquivo a ser gravado; o padrão é 500000. O limite se aplicará apenas se mais de um arquivo de rastreio for solicitado.

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

Ative ou desative o rastreio por cliente. Se *`clientIdentifier`*`=*`, o rastreio será ativado ou desativado para todos os clientes. Por padrão, o rastreio é desativado para todos os clientes.

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. Transmita o arquivo de propriedades de rastreio para a JVM usando uma propriedade de sistema.

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. Execute o cliente.
4. Converta o arquivo de rastreio a partir da codificação binária para texto ou `.html`. Use o comando a seguir:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

em que os argumentos são:

`-?`
Exibe a ajuda

`-i traceFile`
Obrigatório. Passa o arquivo de entrada (por exemplo, `mqtt-0.trc`).

`-o outputFile`
Obrigatório. Define o arquivo de saída (por exemplo, `mqtt-0.trc.html` ou `mqtt-0.trc.txt`).

`-h`
Saída como HTML. A extensão de arquivos de saída deve ser `.html`. Se não especificada, a saída será texto simples.

`-d time`
Indentará uma linha com * se a diferença horária em milissegundos for maior ou igual ao (>=) tempo. Não aplicável à saída HTML.

O exemplo a seguir retirará o arquivo de rastreio no formato HTML

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

O segundo exemplo retirará o arquivo de rastreio como texto simples, com registros de data e hora consecutivos que têm milissegundos com uma diferença de 50 ou maior indentada com um asterisco (*).

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

O exemplo final retirará o arquivo de rastreio como texto simples:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Rastreando o cliente MQTT para C

Configure a variável de ambiente MQTT_C_CLIENT_TRACE para rastrear um aplicativo C do cliente MQTT

Antes de começar

O cliente MQTT para o rastreio C está disponível para o cliente pré-construído do Windows e Linux MQTT para as bibliotecas C e para as bibliotecas do iOS as quais você mesmo constrói.

Sobre esta tarefa

Configure a variável de ambiente MQTT_C_CLIENT_TRACE para um caminho para um arquivo que deve conter a saída de rastreio. A saída de rastreio é gravada no arquivo.

Procedimento

Configure MQTT_C_CLIENT_TRACE=mqttcclient.log antes de executar seu aplicativo C do cliente MQTT.

a) Por exemplo, modifique o script da amostra no "Introdução ao cliente MQTT para C" na página 27:

```
@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) Execute o script a partir do diretório %sdkroot%/sdk/client/c/samples.

Resultados

Os arquivos de saída de rastreio é iniciado com as seguintes linhas:

```
============================================================
                   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
```

Tarefas relacionadas

"Introdução ao cliente MQTT para C" na página 27
Funcionamento adequado com o cliente MQTT de amostra para C em qualquer plataforma na qual é possível compilar a origem C. Verifique se é possível executar o cliente MQTT de amostra para C com o IBM MessageSight ou IBM WebSphere MQ como o MQTT servidor.

Rastreando e Depurando o Cliente Java MQTT (Paho)

O criador de logs padrão usa o recurso de criação de log Java padrão conhecido como `java.util.logging` (JSR47). É possível configurá-lo pelo uso de um arquivo de configuração ou programaticamente.

Sobre esta tarefa

Nota: O cliente Paho Java é aplicável apenas às versões de IBM WebSphere MQ versões 7.5.0.1 e mais recentes Para obter informações que descrevem o rastreio do cliente Java na IBM WebSphere MQ versão 7.5.0.0, consulte "Rastreando o Cliente Java MQTT v3" na página 168

Nota: O rastreio é uma função de suporte. Siga estas instruções se um engenheiro de serviço IBM solicitar que você rastreie seu cliente Java do MQTT A documentação do produto não documenta o formato do arquivo de rastreio nem como usá-lo para depurar um cliente. O rastreio funciona apenas para o cliente Java do Telemetry IBM WebSphere MQ .

O método mais simples para usar um arquivo de configuração é especificar seu nome na propriedade `java.util.logging.config.file`..

Um arquivo de propriedades de trabalho `jsr47min.properties` é fornecido no pacote `org.eclipse.paho.client.mqttv3.logging`

O recurso de criação de log JSR47 pode ser usado de várias maneiras:

- Para coletar mensagens de um conjunto selecionado de pacotes.
- Para coletar mensagens de e abaixo de um nível de log
- Para escolher vários destinos para as mensagens de log
- Fornecendo um criador de logs integrado que grava em um arquivo e controla o tamanho e o número de arquivos usados
- Fornecendo um criador de logs integrado que grava na memória e permite que as mensagens na memória sejam gravadas com base em um acionador
- Se o aplicativo que usa a biblioteca do cliente MQTT também for instrumentada usando JSR47, as mensagens da biblioteca do aplicativo e do cliente serão misturadas

Uma classe de utilitário é fornecida para ajudar a coletar informações de depuração.. Esta classe inclui as mensagens de log e de rastreio que são descritas anteriormente, mas podem coletar informações como propriedades do sistema Java e o valor de variáveis de dentro do cliente Paho.

O recurso de depuração é fornecido na classe pública Debug, que faz parte do pacote `org.eclipse.paho.client.mqttv3.util` Uma instância de Depuração pode ser obtida usando o método getDebug() nos objetos do cliente MQTT assíncronos e síncronos.

Por exemplo:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

O método dumpClientDebug () efetua dump da quantidade máxima de informações de depuração O recurso de log deve ser ativado para capturar as informações de depuração completas, que são gravadas para ele. Para capturar as informações de depuração completas, chame um método de dump quando o problema for conhecido por ocorrer, por exemplo, depois que ocorrer uma exceção específica

Procedimento

1. Crie um arquivo de configuração ou use o jsr47min.properties fornecido.

   Se estiver usando o arquivo de propriedade fornecido, verifique se o acionador de push está configurado para corrigir o nível de erro. Por padrão, isso é configurado para um erro de nível Grave, mas pode ser necessário gravar continuamente o rastreio no arquivo, em vez de reter na memória até que ocorra um erro. Para fazer isso, altere:

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   para

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. Transmita o arquivo de configuração de rastreio para a JVM usando uma propriedade de sistema

   Se você estiver usando o jsr4min.properties , será:

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Execute o cliente.

Resultados

Quando ocorre uma exceção ou um problema, a classe Paho Debug grava o rastreio na memória no destino do arquivo configurado.

O rastreio não é gravado automaticamente no arquivo conforme ele é gerado, isso ocorre apenas quando o acionador push é atingido ou a classe de depuração faz com que o rastreio seja gravado. O último pode requerer mudanças no código do aplicativo

Cada linha é gravada no manipulador de arquivos conforme é criado. É possível controlar o formato no qual as mensagens são gravadas configurando um FileHandler.. Um manipulador de arquivo customizado é fornecido com Paho que grava mais do que o SimpleHandler e menos do que o XMLHandler fornecido com o JRE. Os registros de rastreio que usam o formatador de log do Paho têm o seguinte formato:

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Exemplo

Um arquivo de propriedade de trabalho `jsr47min.properties` é fornecido Esse arquivo contém uma configuração sugerida para coletar rastreio que ajuda a resolver problemas relacionados ao cliente Paho MQTT. Ele configura o rastreio para que seja coletado continuamente na memória com impacto mínimo no desempenho Quando o acionador de push ocorre ou uma solicitação específica é feita para enviar por

push, o rastreio na memória é enviado por push para o manipulador de destino configurado. O acionador de push padrão é uma mensagem de nível Grave, que é uma conexão interrompida. Por padrão, o rastreio coletado na memória é gravado no arquivo especificado neste ponto. Por padrão, esse arquivo é o padrão `java.util.logging.FileHandler` É possível usar a classe de depuração Paho para enviar por push o rastreio de memória para seu destino.

Os detalhes completos do JSR47 podem ser localizados no Javadoc para o pacote 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
```

Como proceder a seguir

Para coletar o rastreio programaticamente, uma classe de utilitário é fornecida para ajudar a coletar informações de depuração Esta classe inclui as mensagens de log e de rastreio que são descritas anteriormente, mas podem coletar informações como propriedades do sistema Java e o valor de variáveis de dentro do cliente Paho.

O recurso de depuração é fornecido na classe pública Debug, que faz parte do pacote `org.eclipse.paho.client.mqttv3.util` Uma instância de Depuração pode ser obtida usando o método getDebug() nos objetos do cliente MQTT assíncronos e síncronos.

Por exemplo:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

O método dumpClientDebug () efetua dump da quantidade máxima de informações de depuração O recurso de log deve ser ativado para capturar as informações de depuração completas, que são gravadas para ele. Para capturar as informações de depuração completas, chame um método de dump quando o problema for conhecido por ocorrer, por exemplo, depois que ocorrer uma exceção específica

Rastreando o cliente MQTT JavaScript

É possível usar o cliente JavaScript para coletar rastreio alterando o aplicativo da web cliente para chamar métodos no objeto do cliente conectado.

Sobre esta tarefa

Para coletar rastreio, é possível usar os métodos a seguir:

- O `client.startTrace()` inicia o rastreio para o cliente
- `client.stopTrace()` para o rastreio do cliente.
- `client.getTraceLog()` retorna o buffer de rastreio atual..

É possível emitir o buffer de rastreio para enviar para o Suporte de Software IBM . Existem várias maneiras de fazer isso. O exemplo mostra o rastreio sendo iniciado e, em seguida, a saída enviada para o console e um endereço de e-mail especificado e, finalmente, o rastreio sendo interrompido.

```
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();
 };
```

Saída de amostra:

```
Client.startTrace, "2013-10-03T10:58:10.531Z", "0.0.0.0",
Client.connect, {"keepAliveInterval":60,"cleanSession":true},, false,
Client._socket_send, {"type":1,"keepAliveInterval":60,"cleanSession":true,
    "clientId":"clientId"},
Client._on_socket_message, {},
Client._on_socket_message, {"type":2,"topicNameCompressionResponse":0,"returnCode":0},
Client.disconnect,Client._socket_send, {"type":14},
Client.getTraceLog, "2013-10-03T10:58:10.548Z",
Client.getTraceLog in flight messages,
```

V 7.5.0.2 Requisitos do sistema para usar conjuntos de cifras SHA-2 com clientes do MQTT

Para Java 6 de IBM, SR13 em diante, é possível usar conjuntos de cifras SHA-2 para proteger seus canais e aplicativos clientes do MQTT . No entanto, os conjuntos de cifras SHA-2 não são ativados por padrão até o Java 7 de IBM, SR4 em diante, portanto, em versões anteriores, deve-se especificar o conjunto necessário. Se você estiver executando um cliente MQTT com seu próprio JRE, é preciso garantir que ele suporta os conjuntos de cifras SHA-2. Para os aplicativos clientes usarem os conjuntos de cifras SHA-2, o cliente também deve configurar o contexto de SSL para um valor que suporte o Transport Layer Security (TLS) versão 1.2.

Para Java 7 de IBM, SR4 em diante, os conjuntos de cifras SHA-2 são ativados por padrão. Para Java 6 de IBM, SR13 e liberações de serviço mais recentes, se você definir um canal do MQTT sem especificar um conjunto de cifras, o canal não aceitará conexões de um cliente usando um conjunto de cifras SHA-2 . Para usar conjuntos de cifras SHA-2, você deve especificar o conjunto requerido na definição de canal.

Isso faz o servidor MQTT ativar o conjunto antes de fazer conexões. Isso também significa que apenas aplicativos cliente que utilizam o conjunto especificado podem se conectar a este canal.

Existe uma limitação similar para o Cliente de MQTT para Java. Se o código do cliente estiver em execução em um Java 1.6 JRE de IBM, os conjuntos de cifras SHA-2 necessários deverão ser ativados explicitamente. Para usar esses conjuntos de cifras, o cliente também deve configurar o contexto de SSL para um valor que suporta a Versão 1.2 do protocolo TLS (Transport Layer Security). Por exemplo:

```
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
java.util.Properties sslClientProps = new java.util.Properties();
sslClientProps.setProperty("com.ibm.ssl.keyStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.keyStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.trustStore", sslKeys.clientKeyStore);
sslClientProps.setProperty("com.ibm.ssl.trustStorePassword", sslKeys.clientStorePassword);
sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
sslClientProps.setProperty("com.ibm.ssl.enabledCipherSuites",
"SSL_RSA_WITH_AES_256_CBC_SHA256" );
mqttConnectOptions.setSSLProperties(sslClientProps);
```

Como em junho de 2013, o Internet Explorer 10 é o único navegador que funciona com o Cliente de sistema de mensagens do MQTT para JavaScript e também suporta o protocolo TLS 1.2 , portanto, é o único navegador que você pode usar se desejar fazer conexões SHA-2 com o cliente JavaScript .

Para uma lista dos conjuntos de cifras que são suportados atualmente, consulte os links relacionados.

Conceitos relacionados

“Configuração do cliente MQTT para autenticação de cliente usando o SSL” na página 105
Para autenticar o cliente MQTT usando SSL, o cliente se conecta a um canal de telemetria usando SSL. Ele deve especificar uma porta TCP que corresponda a um canal de telemetria configurado para autenticar clientes SSL.

“Configuração do cliente MQTT para autenticação de canal usando o SSL” na página 108
Para autenticar o canal de telemetria usando SSL, o cliente deve se conectar ao canal de telemetria usando SSL. Ele deve especificar uma porta que corresponda a um canal de telemetria configurado para SSL. A configuração deve incluir um keystore protegido por passphrase contendo o certificado digital assinado em particular do servidor.

V 7.5.0.1 Restrições no suporte do navegador para aplicativos da web do sistema de mensagens móveis através do SSL

Há diferenças na capacidade entre navegadores diferentes, em plataformas diferentes. Entender essas diferenças ajuda você a configurar aplicativos, autoridades de certificação (CAs) e certificados de cliente para se conectar usando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets.

O sistema de mensagens móveis usando o JavaScript através do SSL é consideravelmente novo, portanto, não é de se surpreender que o navegador diferente e as combinações de plataforma implementaram o recurso de formas um pouco diferentes e para diferentes extensões. A tabela a seguir fornece uma visão geral do que atualmente funciona e não funciona para cada combinação de navegador (Firefox, Chrome, Internet Explorer e Safari) e plataforma (Windows, Linux, Mac, iOS e Android).

Tabela 6. Suporte para SSL pela plataforma e navegador. Para cada navegador e combinação de plataforma, a tabela especifica se as conexões SSL anônimas e não anônimas são suportadas e a extensão na qual o navegador funciona com todas as autoridades de certificação (CA) e certificados de cliente.

da Web	Suporte SSL (S/N)	O SSL funciona com qualquer CA (S/N)	Informações adicionais
Desktop Firefox.	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Sim	Inclua a CA e o certificado de cliente no navegador. Firefox usa seu próprio armazenamento de certificados. Para importar um certificado de autoridade de certificação, clique em **Ferramentas** > **Opções** > **Avançadas** > **Criptografia** > **Visualizar certificados** > **Autoridades** > **Importar** Para importar um certificado de cliente, clique em **Ferramentas** > **Opções** > **Avançadas** > **Criptografia** > **Visualizar certificados** > **Seus certificados** > **Importar** Para ativar uma conexão segura, especifique `https://` na URL. Firefox fornece a opção de selecionar um certificado automaticamente ou solicitar a você toda vez. Firefox também fornece a opção de usar o SSL 3.0 ou TLS 1.0; certifique-se de que ambos estejam selecionados.

Tabela 6. Suporte para SSL pela plataforma e navegador. Para cada navegador e combinação de plataforma, a tabela especifica se as conexões SSL anônimas e não anônimas são suportadas e a extensão na qual o navegador funciona com todas as autoridades de certificação (CA) e certificados de cliente. *(continuação)*

da Web	Suporte SSL (S/N)	O SSL funciona com qualquer CA (S/N)	Informações adicionais
Desktop Chrome.	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Sim	Use o navegador para incluir a CA e o certificado de cliente no armazenamento de certificados do sistema operacional, que é compartilhado com outro software. Para importar um certificado de autoridade de certificação, clique em **Configurações** > **Mostrar configurações avançadas** > **Gerenciar certificados** > **Autoridades de certificação raiz confiáveis** > **Importar** Para importar um certificado de cliente, clique em **Configurações** > **Mostrar Configurações Avançadas** > **Gerenciar Certificados** > **Pessoal** > **Importar** Para ativar uma conexão segura, especifique `https://` na URL. O Chrome fornece a você várias opções; selecione a correta, dependendo se você estiver configurando uma conexão anônima ou não anônima.

Tabela 6. Suporte para SSL pela plataforma e navegador. Para cada navegador e combinação de plataforma, a tabela especifica se as conexões SSL anônimas e não anônimas são suportadas e a extensão na qual o navegador funciona com todas as autoridades de certificação (CA) e certificados de cliente. *(continuação)*

da Web	Suporte SSL (S/N)	O SSL funciona com qualquer CA (S/N)	Informações adicionais
Internet Explorer.	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Sim	Quando você fizer uma conexão SSL não anônima, você será solicitado a escolher o certificado de cliente correto. Internet Explorer usa o armazenamento de certificados do Windows, que é compartilhado com outro software. Para importar um certificado de CA, clique em **Ferramentas** > **Opções da Internet** > **Conteúdo** > **Certificados** > **Autoridades de Certificação de Raiz Confiável** > **Importação** Para importar um certificado de cliente, clique em **Ferramentas** > **Opções da Internet** > **Conteúdo** > **Certificados** > **Pessoal** > **Importar**
Desktop Safari.	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Sim	Use o navegador para incluir a CA e o certificado de cliente no armazenamento de certificados do sistema operacional, que é compartilhado com outro software.

Tabela 6. Suporte para SSL pela plataforma e navegador. Para cada navegador e combinação de plataforma, a tabela especifica se as conexões SSL anônimas e não anônimas são suportadas e a extensão na qual o navegador funciona com todas as autoridades de certificação (CA) e certificados de cliente. *(continuação)*

da Web	Suporte SSL (S/N)	O SSL funciona com qualquer CA (S/N)	Informações adicionais
Firefox em Android	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Não	Não anônima: Os certificados cliente não funcionarão, pois não é possível atender ao requisito para incluir sua CA à lista em Firefox. Para importar um certificado de cliente, Clique em **Configurações** > **Segurança** > **Armazenamento de Credencial**. Se seu certificado for assinado por uma CA confiável na lista, será possível fazer uma conexão segura.
Chrome em Android	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Não	Não anônima: Os certificados cliente não funcionarão, pois não é possível atender ao requisito para incluir sua CA à lista em Chrome. **Nota:** Google planeja suportar isso na versão 27 do Chrome. Este foi um defeito aberto desde a versão 18. Para importar um certificado de cliente, Clique em **Configurações** > **Segurança** > **Armazenamento de Credencial**. Se seu certificado for assinado por uma CA confiável na lista, será possível fazer uma conexão segura.

Tabela 6. Suporte para SSL pela plataforma e navegador. Para cada navegador e combinação de plataforma, a tabela especifica se as conexões SSL anônimas e não anônimas são suportadas e a extensão na qual o navegador funciona com todas as autoridades de certificação (CA) e certificados de cliente. *(continuação)*

da Web	Suporte SSL (S/N)	O SSL funciona com qualquer CA (S/N)	Informações adicionais
Safari em iOS	SSL anônimo - Sim SSL não anônima - Sim	SSL anônimo - Sim SSL não anônima - Não	Não anônima: O dispositivo não confiará no certificado de cliente, mesmo quando o certificado de CA for instalado ao mesmo tempo. Safari usa o armazenamento de certificados do dispositivo. Para importar nesse armazenamento, clique em **Configurações** > **Geral** > **Perfil**e atenda o certificado de autoridade de certificação ou de cliente de uma página da web ou envie por e-mail para você.
Chrome em iOS	SSL anônimo - Sim SSL não anônima - Não	SSL anônima - Não SSL não anônima - Não	Anônimo: Apenas os aplicativos da Apple podem acessar o armazenamento da raiz do sistema do iOS. Portanto, o Chrome deve usar sua própria lista de CA, a qual não é possível ser incluída. Não anônima: Os certificados cliente não funcionarão, pois não é possível atender ao requisito para incluir sua CA à lista.

Tarefas relacionadas

“Conectando o Cliente de sistema de mensagens do MQTT para JavaScript através do SSL e WebSockets” na página 78

Conecte o aplicativo da web com segurança aoIBM WebSphere MQ usando as páginas HTML de amostra do Cliente de sistema de mensagens do MQTT para JavaScript com SSL e o WebSocket protocol.

Informações relacionadas

Mozilla: (SSL) O Firefox usa o armazenamento de CA Android ou seu próprio?

Chromium: emita 134418 - Implementa suporte ao certificado cliente
Não é possível abrir site https sem certificado confiável em ie10

Resolução do problema: cliente MQTT não se conecta

Resolva o problema de um programa do cliente MQTT com falha ao se conectar ao serviço de telemetria (MQXR).

Antes de começar

O problema está no servidor, no cliente ou com a conexão? Você gravou seu próprio cliente de manipulação de protocolo MQTT v3 ou um aplicativo cliente MQTT usando os clientes C ou Java WebSphere MQTT?

Execute o aplicativo de verificação fornecido com o WebSphere MQ Telemetry no servidor e verifique se o canal de telemetria e o serviço de telemetria (MQXR) estão sendo executados corretamente. Em seguida, transfira o aplicativo de verificação ao cliente e execute o aplicativo de verificação.

Sobre esta tarefa

Há várias razões pelas quais um cliente MQTT pode não se conectar ou você pode concluir que ele não foi conectado ao servidor de telemetria.

Procedimento

1. Considere quais inferências podem ser extraídas do código de razão que o serviço de telemetria (MQXR) retornou para `MqttClient.Connect`. Qual é o tipo de falha de conexão?

Opção	Descrição
REASON_CODE_INVALID_PROTOCOL_VERSION	Certifique-se de que o endereço de soquete corresponde a um canal de telemetria e você não usou o mesmo endereço de soquete para outro broker.
REASON_CODE_INVALID_CLIENT_ID	Verifique se o identificador de cliente não tem mais de 23 bytes e contém apenas caracteres do intervalo: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	Verifique se o identificador do cliente não é o mesmo que o nome do gerenciador de filas.
REASON_CODE_SERVER_CONNECT_ERROR	Verifique se o serviço de telemetria (MQXR) e o gerenciador de filas estão em execução normalmente. Use **netstat** para verificar se o endereço de soquete não está alocado para outro aplicativo.

   Se você tiver gravado uma biblioteca do cliente MQTT em vez de usar uma das bibliotecas fornecidas pelo IBM WebSphere MQ Telemetry, consulte o código de retorno `CONNACK` .

   Desses três erros, é possível inferir que o cliente se conectou ao serviço de telemetria (MQXR), mas o serviço localizou um erro.
2. Considere quais inferências podem ser obtidas a partir dos códigos de razão que o cliente produz quando o serviço de telemetria (MQXR) não responder:

Opção	Descrição
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	Procure um arquivo FDC no servidor; consulte "Logs do lado do servidor" na página 164. Quando o serviço de telemetria (MQXR) detectar que o cliente atingiu o tempo limite, ele gravará um arquivo de first-failure data capture (FDC). Ele grava um arquivo FDC sempre que a conexão for inesperadamente interrompida.

O serviço de telemetria (MQXR) pode não ter respondido ao cliente e o tempo limite no cliente expirou. O cliente Java do WebSphere MQ Telemetry somente será interrompido se o aplicativo tiver configurado um tempo limite indefinido. O cliente lançará uma dessas exceções após o tempo limite configurado para `MqttClient.Connect` expirar com um problema de conexão não diagnosticado.

A menos que localize um arquivo FDC correlacionado à falha na conexão, não será possível inferir se o cliente tentou se conectar ao servidor:

a) Confirme se o cliente enviou uma solicitação de conexão.

   Verifique a solicitação TCPIP com uma ferramenta como **tcpmon**, disponível em https://java.net/projects/tcpmon

b) O endereço de soquete remoto usado pelo cliente corresponde ao endereço de soquete definido para o canal de telemetria?

   A classe de persistência de arquivo padrão no cliente Java SE MQTT fornecido com o IBM WebSphere MQ Telemetry cria uma pasta com o nome: *clientIdentifier*`-tcp`*hostNameport* ou *clientIdentifier*`-ssl`*hostNameport* no diretório ativo do cliente. O nome da pasta informa `hostName` e `port` usados na tentativa de conexão; consulte "Arquivos de log do lado do cliente" na página 165.

c) É possível executar ping do endereço do servidor remoto?

d) O **netstat** no servidor mostra que o canal de telemetria é executado na porta que o cliente está se conectando também?

3. Verifique se o serviço de telemetria (MQXR) localizou um problema na solicitação do cliente.

   O serviço de telemetria (MQXR) grava os erros que ele detecta em `mqxr.log` e o gerenciador de filas grava os erros em `AMQERR01.LOG`; consulte

4. Tente isolar o problema executando outro cliente.

   - Execute o aplicativo de amostra MQTT usando o mesmo canal de telemetria.
   - Execute o cliente GUI **wmqttSample** para verificar a conexão. Obtenha **wmqttSample** fazendo o download do SupportPac IA92

     Nota: Versões mais antigas do IA92 não incluem a biblioteca do cliente Java MQTT v3 .

   Execute os programas de amostra na plataforma do servidor para eliminar as incertezas sobre a conexão de rede, em seguida, execute as amostras na plataforma do cliente.

5. Outras coisas para serem verificadas:

   a) Dezenas de milhares de clientes MQTT estão tentando se conectar ao mesmo tempo?

      Os canais de telemetria têm uma fila para armazenar em buffer uma lista não processada de conexões de entrada. As conexões são processadas acima de 10.000 por segundo. O tamanho do buffer da lista não processada é configurável usando o assistente de canal de telemetria no IBM WebSphere MQ Explorer. Seu tamanho padrão é 4096. Verifique se a lista não processada não foi configurada para um valor baixo.

   b) O serviço de telemetria (MQXR) e o gerenciador de filas ainda estão em execução?

   c) O cliente se conectou a um gerenciador de filas de alta disponibilidade que alternou seu endereço TCPIP?

   d) Um firewall está seletivamente filtrando pacotes de dados de saída ou retorno?

Resolução do problema: conexão do cliente MQTT eliminada

Descubra o que está fazendo com que um cliente lance exceções inesperadas `ConnectionLost` após uma execução e conexão bem-sucedidas por um tempo curto ou longo.

Antes de começar

O cliente MQTT se conectou com sucesso. O cliente pode estar ativo por muito tempo. Se os clientes estiverem começando com apenas um curto intervalo entre eles, o tempo entre a conexão com sucesso e a conexão que está sendo descartada pode ser curto.

Não é difícil distinguir uma conexão descartada de uma conexão que foi feita com sucesso e, em seguida, descartada. Uma conexão eliminada é definida pelo cliente MQTT chamando o método `MqttCallback.ConnectionLost` . O método será chamado somente após a conexão ter sido estabelecida com sucesso. O sintoma é diferente para quando `MqttClient.Connect` lançar uma exceção após receber uma confirmação negativa ou atingir um tempo limite.

Se o aplicativo do cliente MQTT não estiver usando as bibliotecas do cliente MQTT fornecidas pelo IBM WebSphere MQ, o sintoma dependerá do cliente No protocolo do MQTT v3, o sintoma é uma falta de resposta oportuna para uma solicitação ao servidor ou a falha da conexão TCP/IP.

Sobre esta tarefa

O cliente MQTT chama `MqttCallback.ConnectionLost` com uma exceção lançada em resposta a quaisquer problemas de lado do servidor encontradas após receber uma confirmação de conexão positiva. Quando um cliente MQTT retornar a partir de `MqttTopic.publish` e `MqttClient.subscribe`, a solicitação será transferida para um encadeamento do cliente MQTT responsável por enviar e receber mensagens. Erros do lado do servidor são relatados de maneira assíncrona, transmitindo uma exceção lançada ao método de retorno de chamada `ConnectionLost`.

O serviço de telemetria (MQXR) sempre gravará um arquivo de first-failure data capture se ele descartar a conexão.

Procedimento

1. Outro cliente iniciado usou o mesmo `ClientIdentifier`?

   Se um segundo cliente for iniciado ou o mesmo cliente for reiniciado usando o mesmo `ClientIdentifier`, a primeira conexão com o primeiro cliente será descartada.
2. Será que o cliente acessou um tópico que ele não está autorizado a publicar ou assinar?

   Todas as ações que o serviço de telemetria obtém em nome de um cliente que retorna `MQCC_FAIL` resultará no descarte da conexão do cliente através do serviço.

   O código de razão não é retornado ao cliente.

   - Procure mensagens de log nos arquivos `mqxr.log` e `AMQERR01.LOG` para o gerenciador de filas ao qual o cliente está conectado; consulte “Logs do lado do servidor” na página 164.
3. A conexão TCP/IP foi descartada?

   Um firewall pode ter uma configuração de tempo limite baixa para marcação de uma conexão TCPIP como inativa e descartou a conexão.

   - Diminua o tempo inativo de conexão TCPIP usando `MqttConnectOptions.setKeepAliveInterval`.

Resolução de problemas: mensagens perdidas em um aplicativo MQTT

Resolva o problema de perda de uma mensagem. A mensagem não persistente, foi enviada ao lugar errado ou nunca foi enviada? Um programa cliente codificado incorretamente pode perder mensagens.

Antes de começar

Quão certo você está de que a mensagem que você enviou, foi perdida? É possível inferir que uma mensagem está perdida porque a mensagem não foi recebida? Se mensagem for uma publicação, qual mensagem está perdida: a mensagem enviada pelo publicador ou a mensagem enviada para o assinante? Ou fez com que a assinatura se perdesse e o broker não está enviando publicações para essa assinatura ao assinante?

Se a solução envolver a publicação/assinatura distribuída, usando clusters ou hierarquias de publicação/assinatura, há várias questões de configuração que poderão resultar no aparecimento de uma mensagem perdida.

Se você enviou uma mensagem com a qualidade de serviço "Pelo menos uma vez" ou "No máximo uma vez", será provável que a mensagem que você acha que está perdida não tenha sido entregue da forma esperada. É improvável que a mensagem tenha sido excluída do sistema de forma equivocada. Ele pode ter falhado ao criar a publicação ou a assinatura que você esperava.

O passo mais importante a se tomar para a determinação de problemas de mensagens perdidas é confirmar que a mensagem está perdida. Recrie o cenário e perca mais mensagens. Use a qualidade de serviço "Pelo menos uma vez" ou "No máximo uma vez" para eliminar todos os casos de descartes de mensagens do sistema.

Sobre esta tarefa

Há quatro pernas para diagnosticar uma mensagem perdida.

1. Mensagens "Fire and forget" funcionando como projetadas. Mensagens "Fire and forget" às vezes são descartadas pelo sistema.
2. Configuração: configuração de publicação/assinatura com as autoridades corretas em um ambiente distribuído não é simples.
3. Erros de programação do cliente: a responsabilidade pela entrega de mensagens não é responsabilidade exclusiva do código gravado pela IBM.
4. Se você tiver esgotado todas estas possibilidades, você poderá decidir envolver o serviço IBM.

Procedimento

1. Se a mensagem perdida tinha a qualidade de serviço "Fire and forget", configure a qualidade de serviço "Pelo menos uma vez" ou "No máximo uma vez". Tentativa de perder a mensagem novamente.
   - As mensagens enviadas com a qualidade de serviço "Fire and forget" são imediatamente descartadas pelo IBM WebSphere MQ em várias circunstâncias:
     - Perda de comunicações e canal interrompido.
     - Gerenciador de filas encerrado.
     - Número excessivo de mensagens.
   - A entrega das mensagens "Fire and forget" depende da confiabilidade do TCP/IP. TCP/IP continua a enviar pacotes de dados novamente até que sua entrega seja reconhecida. Se a sessão TCP/IP for interrompida, as mensagens com a qualidade de serviço "Fire and forget" serão perdidas. A sessão pode ser interrompida pelo fechamento do cliente ou do servidor, um problema de comunicações ou uma desconexão de sessão pelo firewall.
2. Verifique se o cliente está reiniciando a sessão anterior, para enviar mensagens não entregues com a qualidade de serviço "Pelo menos uma vez" ou "No máximo uma vez" novamente.
   a) Se o aplicativo cliente estiver usando o cliente MQTT do Java SE, verifique se ele configura `MqttClient.CleanSession` como `false`
   b) Se você estiver usando as bibliotecas do cliente diferentes, verifique se uma sessão estará sendo reiniciada corretamente.
3. Verifique se o aplicativo cliente está reiniciando a mesma sessão e não uma sessão diferente por engano.

Para iniciar a mesma sessão novamente, `cleanSession = false` e o `Mqttclient.clientIdentifier` e o `MqttClient.serverURI` devem ser os mesmos que a sessão anterior.

4. Se uma sessão for fechada prematuramente, verifique se a mensagem estará disponível no armazenamento de persistência no cliente para enviar novamente.
   a) Se o aplicativo cliente estiver usando o cliente MQTT do Java SE, verifique se a mensagem está sendo salva na pasta de persistência; consulte “Arquivos de log do lado do cliente” na página 165
   b) Se você estiver usando as bibliotecas do cliente diferente ou tiver implementado seu próprio mecanismo de persistência, verifique se ele estará funcionando corretamente.
5. Verifique se ninguém tiver excluído a mensagem antes de ela ser entregue.

   Mensagens não entregues aguardando a entrega para clientes MQTT são armazenadas em `SYSTEM.MQTT.TRANSMIT.QUEUE`. As mensagens esperando a entrega para o servidor de telemetria são armazenadas pelo mecanismo de persistência do cliente. Consulte Persistência de mensagem em clientes do MQTT.
6. Verifique se o cliente tem uma assinatura para a publicação que ele espera receber.

   Listar assinaturas usando o WebSphere MQ Explorer ou usando comandos do **runmqsc** ou PCF Todas as assinaturas do cliente MQTT são nomeadas. Eles recebem um nome do formato: *ClientIdentifier*:*Topic name*
7. Verifique se o publicador tem autoridade para publicar e o assinante para assinar o tópico de publicação.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   Em um sistema de publicação/assinatura em cluster, o assinante deve estar autorizado para o tópico no gerenciador de filas ao qual o assinante está conectado. Não é necessário que o assinante seja autorizado a assinar o tópico no gerenciador de filas no qual a publicação é publicada. Os canais entre os gerenciadores de filas devem estar corretamente autorizados para transmitir a assinatura de proxy e encaminhar a publicação.

   Crie a mesma assinatura e publicação para ela usando o IBM WebSphere MQ Explorer. Simule sua publicação e assinatura do aplicativo cliente usando o utilitário do cliente. Inicie o utilitário a partir do IBM WebSphere MQ Explorer e altere seu ID do usuário para corresponder ao adotado pelo aplicativo cliente.
8. Verifique se o assinante tem permissão para colocar a publicação no `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```
9. Verifique se o aplicativo ponto a ponto IBM WebSphere MQ tem autoridade para colocar sua mensagem no `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

   Consulte "Enviando uma mensagem para um cliente diretamente" em Configurar enfileiramento distribuído para enviar mensagens a clientes MQTT.

Resolução de problema: serviço de telemetria (MQXR) não inicia

Resolva o problema do serviço de telemetria (MQXR) que falha ao iniciar. Verifique a instalação do WebSphere MQ Telemetry e se nenhum arquivo está ausente, foi movido ou tem as permissões erradas. Verifique os caminhos usados pelo serviço de telemetria (MQXR) para localizar os programas de serviço de telemetria (MQXR).

Antes de começar

O recurso do WebSphere MQ Telemetry está instalado. O IBM WebSphere MQ Explorer tem uma pasta Telemetry no **IBM WebSphere MQ > Gerenciadores de Filas> *qMgrNome* > Telemetry**. Se a pasta não existir, a instalação falhou.

O serviço de Telemetria (MQXR) deve ter sido criado para ser iniciado. Se o serviço de telemetria (MQXR) não tiver sido criado, em seguida, execute o assistente **Definir configuração de amostra ...** na pasta `Telemetry`.

Se o serviço de telemetria (MQXR) tiver sido iniciado antes, então as pastas adicionais **Canais** e **Status do canal** serão criadas sob a pasta `Telemetry`. O serviço de telemetria, `SYSTEM.MQXR.SERVICE`, está na pasta **Serviços**. Ela está visível se o botão de opções Explorer para mostrar Objetos do sistema for clicado.

Clique com o botão direito em `SYSTEM.MQXR.SERVICE` para iniciar e parar o serviço, mostrar seu status e exibir se seu ID do usuário tem autoridade para iniciar o serviço.

Sobre esta tarefa

O serviço de telemetria `SYSTEM.MQXR.SERVICE` (MQXR) falha ao iniciar. Uma falha ao iniciar se manifesta de duas maneiras diferentes:

1. O comando inicial falha imediatamente.
2. O comando inicial é bem-sucedido e é seguido imediatamente pela parada do serviço.

Procedimento

1. Iniciar o serviço

   Resultado

   O serviço para imediatamente. Uma janela exibe uma mensagem de erro; por exemplo:

   ```
   WebSphere MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Motivo

   Arquivos estão ausentes na instalação ou as permissões em arquivos instalados estão configuradas incorretamente.

   O recurso Telemetry IBM WebSphere MQ é instalado apenas em um par de gerenciadores de filas altamente disponíveis. Se a instância do gerenciador de filas alternar para uma em espera, ela tenta iniciar o `SYSTEM.MQXR.SERVICE`. O comando para iniciar o serviço falha porque o serviço de telemetria (MQXR) não está instalado na instância em espera.

   Investigação

   Consulte os logs de erro; consulte “Logs do lado do servidor” na página 164.

   Ações

   Instale ou desinstale e reinstale o recurso doWebSphere MQ Telemetry.

2. Inicie o serviço; espere 30 segundos; atualize o Explorer e verifique o status de serviço.

   Resultado

   O serviço é iniciado e, em seguida, para.

   Motivo

   `SYSTEM.MQXR.SERVICE` iniciou o comando **`runMQXRService`**, mas o comando falhou.

   Investigação

   Consulte os logs de erro; consulte “Logs do lado do servidor” na página 164.

   Veja se o problema ocorre somente com o canal de amostra definido. Faça backup e limpe o conteúdo do diretório *`WMQ data directory`*`\Qmgrs\`*`qMgrName`*`\mqxr\`. Execute o assistente de configuração de amostra e tente iniciar o serviço.

Ações

Procure problemas de permissão e de caminho.

Resolução do problema: o módulo de login JAAS não é chamado pelo serviço de telemetria

Descubra se o módulo de login JAAS não está sendo chamado pelo serviço de telemetria (MQXR) e configure o JAAS para corrigir o problema.

Antes de começar

Você modificou *WMQ installation directory*\mqxr\samples\LoginModule.java para criar sua própria classe de autenticação *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. Como alternativa, você gravou suas próprias classes de autenticação JAAS e as colocou em um diretório de sua escolha. Após algum teste inicial com o serviço de telemetria (MQXR), você suspeita que a classe de autenticação não está sendo chamada pelo serviço de telemetria (MQXR).

Nota: Proteja-se com relação a possibilidade de que sua classe de autenticação possa ser sobrescrita pela manutenção que está sendo aplicada ao WebSphere MQ. Use seu próprio caminho para as classes de autenticação, em vez de um caminho na árvore de diretórios do WebSphere MQ.

Sobre esta tarefa

A tarefa usa um cenário para ilustrar como resolver o problema. No cenário, um pacote chamado `security.jaas` contém uma classe de autenticação JAAS chamada `JAASLogin.class`. Ele é armazenado no caminho `C:\WMQTelemetryApps\security\jaas`. Consulte Configuração do Canal de Telemetria JAAS para obter ajuda na configuração do JAAS para IBM WebSphere MQ Telemetria. O exemplo, “Exemplo de configuração JAAS” na página 188 é uma configuração de amostra.

Procedimento

1. Consulte em `mqxr.log` para uma exceção lançada por `javax.security.auth.login.LoginException`.

   Consulte “Logs do lado do servidor” na página 164 para obter o caminho para `mqxr.log` e Figura 54 na página 189 para um exemplo da exceção listada no log.
2. Corrija a configuração JAAS, comparando-a com o exemplo trabalhado no “Exemplo de configuração JAAS” na página 188.
3. Substitua a classe de login pela amostra `JAASLoginModule`, após sua refatoração em seu pacote de autenticação e implemente-a usando o mesmo caminho. Alterne o valor de `loggedIn` entre `true` e `false`.

   Se o problema desaparecer quando o `loggedIn` for `true` e parecer o mesmo quando o `loggedIn` for `false`, o problema residirá em sua classe de login.
4. Verifique se o problema está com a autorização, em vez da autenticação.
   a) Mude a definição do canal de telemetria para executar a verificação de autorização usando um ID do usuário fixo. Selecione um ID de usuário que seja membro do grupo `mqm`.
   b) Execute novamente o aplicativo cliente.

      Se o problema desaparecer, a solução residirá com o ID do usuário que está sendo transmitido para autorização. Qual é o nome do usuário que está sendo transmitido? Imprima-o para o arquivo de seu módulo de login. Verifique suas permissões de acesso usando o IBM WebSphere MQ Explorer ou **dspmqauth**.

Exemplo de configuração JAAS

Use o assistente **Novo canal de telemetria** , no WebSphere MQ Explorer, para configurar um canal de telemetria. O cliente se conecta na porta 1884 e se conecta ao canal de telemetria `JAASMCAUser`. Figura 48 na página 188 mostra um exemplo do arquivo de propriedades de telemetria criado pelo assistente de telemetria. Não edite esse arquivo diretamente. O canal é autenticado usando JAAS, usando a configuração chamada `JAASConfig`. Depois que o cliente for autenticado, ele usará o ID do usuário `Admin` para autorizar seu acesso aos objetos do 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

O arquivo de configuração do JAAS possui uma sub-rotina denominada `JAASConfig` que nomeia a classe Java `security.jaas.JAASLogin`, que JAAS deve ser usada para autenticar clientes

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Figura 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Quando `SYSTEM.MQTT.SERVICE` iniciar, ele incluirá o caminho em Figura 50 na página 188 para seu caminho de classe.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figura 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

Figura 51 na página 188 mostra o caminho adicional no Figura 50 na página 188 incluído no caminho de classe configurado para o serviço de 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. Saída do caminho de classe a partir de runMQXRService.bat

A saída em Figura 52 na página 189 mostra que o serviço de telemetria (MQXR) foi iniciado com a definição de canal mostrada em Figura 48 na página 188.

```
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 o aplicativo cliente se conectar ao canal do JAAS, se `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` não corresponder ao nome de uma sub-rotina do JAAS no arquivo `jaas.config`, a conexão falhará e o cliente lançará uma exceção com um código de retorno de `0`; consulte Figura 53 na página 189. A segunda exceção, `Client is not connected (32104)`, foi lançada porque o cliente tentou se desconectar quando ele não estava conectado.

```
C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
Userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
        at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
        at java.io.DataInputStream.readByte(DataInputStream.java:269)
        at
com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.ja
va:56)
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:90)
        ... 1 more
Client is not connected (32104)
        at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:117)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
        at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
        at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)
```

Figura 53. Exceção lançada ao se conectar ao com.ibm.mq.id.PubAsyncRestartable

`mqxr.log` contém a saída adicional mostrada em Figura 53 na página 189.

O erro é detectado pelo JAAS que lança `javax.security.auth.login.LoginException` com a causa `No LoginModules configured for JAAS`. Isso pôde ser causado, conforme descrito em Figura 54 na página 189, por um nome de configuração inválido. Ele também pode ser o resultado de outros problemas JAAS que foram encontrados ao carregar a configuração de JAAS.

Se nenhuma exceção for relatada pelo JAAS, o JAAS carregou com êxito a classe `security.jaas.JAASLogin` denominada na sub-rotina `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 - erro ao carregar a configuração de JAAS

Resolução de problemas: iniciando ou executando o daemon

Consulte o daemon do Telemetry do IBM WebSphere MQ para o log do console de dispositivos, ative o rastreio ou use a tabela de sintomas neste tópico para solucionar problemas com o daemon.

Procedimento

1. Verifique o log do console.

   Se o daemon estiver em execução no primeiro plano, as mensagens do console serão gravadas na janela do terminal. Se o daemon foi iniciado em segundo plano, o console está no local para o qual você redirecionou o `stdout`.
2. Reinicie o daemon.

   Mudanças no arquivo de configuração não são ativadas até que o daemon seja reinicializado.
3. Consulte Tabela 7 na página 190:

Tabela 7. Tabela Sintoma

Problema	**Solução sugerida**

A mensagem a seguir será exibida ao iniciar o daemon no Windows:
`O sistema não pode executar o programa especificado`
ou
`O aplicativo falhou ao iniciar pois sua configuração lado a lado está incorreta.`
	Instale o Microsoft Visual C++ 2008 Redistributable Package.

Dois ou mais daemons ou servidores com capacidade de MQTT são interconectados por uma ponte ou pontes e o processador está mostrando o carregamento excessivo.	Existe possivelmente um loop de mensagem, com uma ou mais mensagens sendo repetidamente transmitidas de um servidor para outro. Examine os parâmetros de tópico nos arquivos de configuração. Use os tópicos mais específicos quando possível. Caracteres curinga amplos em ambas as direções são a causa mais comum de loops de conexão.
A ponte não pode se conectar a um servidor com capacidade de MQTT remoto ao qual outros clientes MQTT podem se conectar.	O servidor remoto pode ser incompatível com tentativas de determinar se o servidor remoto também é o daemon do WebSphere MQ Telemetry para dispositivos. Tente configurar **`try_private`** como `off` para desativar o processamento especial para eliminar loops de mensagem.
Esta mensagem é impressa quando uma ponte está configurada: `Aviso: a conexão não foi o primeiro pacote no soquete 1888, obtido CONNACK.`	Você provavelmente terá configurado uma ponte para loop de volta para o daemon local. O loopback não é suportado.

Resolvendo problema: clientes MQTT não se conectam ao daemon

Os clientes não estão se conectando ao daemon, o daemon não está se conectando a outros daemons ou a um canal de telemetria do WebSphere MQ.

Sobre esta tarefa

Rastreie cada pacote MQTT enviado e recebido pelo daemon.

Procedimento

Configure o parâmetro **`trace_output`** como `protocol` no arquivo de configuração do daemon ou envie um comando para o daemon usando o arquivo `amqtdd.upd` .

Consulte Transferir mensagens entre o IBM WebSphere MQ daemon de telemetria para dispositivos e IBM WebSphere MQ , para obter um exemplo de uso do arquivo `amqtdd.upd` .

Usando a configuração de protocolo, o daemon imprimirá uma mensagem para o console que descreve cada pacote MQTT que ele envia e recebe.

Avisos

Estas informações foram desenvolvidas para produtos e serviços oferecidos nos Estados Unidos.

É possível que a IBM não ofereça os produtos, serviços ou recursos discutidos nesta publicação em outros países. Consulte seu representante local do IBM para obter informações sobre produtos e serviços disponíveis atualmente em sua área. Qualquer referência a um IBM produto, programa ou serviço não se destina a estado ou significa que apenas esse produto IBM, programas ou serviços possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM poderá ser utilizado em substituição. Entretanto, a avaliação e verificação da operação de qualquer produto, programa ou serviço não IBM são de responsabilidade do Cliente.

A IBM pode ter patentes ou aplicativos de patentes pendentes relativas aos assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum sobre tais patentes. é possível enviar pedidos de licença, por escrito, para:

Relações Comerciais e Industriais da IBM
Av. Pasteur, 138-146
Botafogo
Rio, RJ 10504-1785
U.S.A.

Para pedidos de licença relacionados a informações de DBCS (Conjunto de Caracteres de Byte Duplo), entre em contato com o Departamento de Propriedade Intelectual da IBM em seu país ou envie pedidos de licença, por escrito, para:

licença de propriedade intelectual
IBM World Trade Asia Corporation Licensing
IBM Japan, Ltd.
Minato-ku
Tóquio 103-8510, Japão

O parágrafo a seguir não se aplica a nenhum país em que tais disposições não estejam de acordo com a legislação local: A INTERNATIONAL BUSINESS MACHINES CORPORATION FORNECE ESTA PUBLICAÇÃO "NO ESTADO EM QUE SE ENCONTRA", SEM GARANTIA DE NENHUM TIPO, SEJA EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS A ELAS NÃO SE LIMITANDO, AS GARANTIAS IMPLÍCITAS DE NÃO INFRAÇÃO, COMERCIALIZAÇÃO OU ADEQUAÇÃO A UM DETERMINADO PROPÓSITO. Alguns países não permitem a exclusão de garantias expressas ou implícitas em certas transações; portanto, essa disposição pode não se aplicar ao Cliente.

Essas informações podem conter imprecisões técnicas ou erros tipográficos. Periodicamente, são feitas nas informações aqui contidas; essas alterações serão incorporadas em futuras edições desta publicação. IBM pode aperfeiçoar e/ou alterar no produto(s) e/ou programa(s) descritos nesta publicação a qualquer momento sem aviso prévio.

Referências nestas informações a websites não IBM são fornecidas apenas por conveniência e não representam de forma alguma um endosso a esses websites. Os materiais contidos nesses websites não fazem parte dos materiais desse produto IBM e a utilização desses websites é de inteira responsabilidade do Cliente.

A IBM pode utilizar ou distribuir as informações fornecidas da forma que julgar apropriada sem incorrer em qualquer obrigação para com o Cliente.

Licenciados deste programa que desejam obter informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem entrar em contato com:

Av. Pasteur, 138-146
Av. Pasteur, 138-146

Botafogo
Rio de Janeiro, RJ
U.S.A.

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriadas, incluindo em alguns casos o pagamento de uma taxa.

O programa licenciado descrito nesta publicação e todo o material licenciado disponível para ele são fornecidos pela IBM sob os termos do IBM Customer Agreement, IBM Contrato de Licença do Programa Internacional ou qualquer contrato equivalente entre as partes.

Todos os dados de desempenho aqui contidos foram determinados em um ambiente controlado. Portanto, os resultados obtidos em outros ambientes operacionais podem variar significativamente. Algumas medidas podem ter sido tomadas em sistemas em nível de desenvolvimento e não há garantia de que estas medidas serão iguais em sistemas geralmente disponíveis. Além disto, algumas medidas podem ter sido estimadas através de extrapolação. Os resultados reais podem variar. usuários deste documento devem verificar os dados aplicáveis para seu ambiente específico.

As informações relativas a produtos não IBM foram obtidas junto aos fornecedores dos respectivos produtos, de seus anúncios publicados ou de outras fontes disponíveis publicamente. A IBM não testou estes produtos e não pode confirmar a precisão de seu desempenho, compatibilidade nem qualquer outra reivindicação relacionada a produtos não IBM. Dúvidas sobre os recursos de produtos não IBM devem ser encaminhadas diretamente a seus fornecedores.

Todas as declarações relacionadas aos objetivos e intenções futuras da IBM estão sujeitas a alterações ou cancelamento sem aviso prévio e representam somente metas e objetivos.

Essas informações contêm exemplos de dados e relatórios utilizados em operações diárias de negócios. Para ilustrá-los da forma mais completa possível, os exemplos incluem nomes de indivíduos, empresas, marcas e produtos. Todos estes nomes são fictícios e qualquer semelhança com os nomes e endereços utilizados por uma empresa real é mera coincidência.

LICENÇA DE COPYRIGHT :

Estas informações contêm programas de aplicativos de amostra na linguagem fonte, ilustrando as técnicas de programação em diversas plataformas operacionais. O Cliente pode copiar, modificar e distribuir estes programas de amostra sem a necessidade de pagar à IBM, com objetivos de desenvolvimento, uso, marketing ou distribuição de programas aplicativos em conformidade com a interface de programação de aplicativo para a plataforma operacional para a qual os programas de amostra são criados. Esses exemplos não foram testados completamente em todas as condições. Portanto, a IBM não pode garantir ou implicar a confiabilidade, manutenção ou função destes programas.

Se estiver visualizando estas informações em formato eletrônico, as fotografias e ilustrações coloridas poderão não aparecer.

Informações sobre a Interface de Programação

As informações da interface de programação, se fornecidas, destinam-se a ajudá-lo a criar software aplicativo para uso com este programa.

Este manual contém informações sobre interfaces de programação desejadas que permitem que o cliente grave programas para obter os serviços do IBM WebSphere MQ.

No entanto, estas informações também podem conter informações sobre diagnósticos, modificações e ajustes. As informações sobre diagnósticos, modificações e ajustes são fornecidas para ajudá-lo a depurar seu software aplicativo.

Importante: Não use essas informações de diagnóstico, modificação e ajuste como uma interface de programação, pois elas estão sujeitas a mudanças

Marcas comerciais

IBM, o logotipo IBM , ibm.com, são marcas registradas da IBM Corporation, registradas em várias jurisdições no mundo todo Uma lista atual de marcas registradas da IBM está disponível na Web em "Informações de copyright e marca registrada"www.ibm.com/legal/copytrade.shtml. Outros nomes de produtos e serviços podem ser marcas comerciais da IBM ou de outras empresas.

Microsoft e Windows são marcas comerciais da Microsoft Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada do The Open Group nos Estados Unidos e em outros países.

Linux é uma marca registrada de Linus Torvalds nos Estados Unidos e/ou em outros países.

Este produto inclui software desenvolvido pelo Projeto Eclipse (http://www.eclipse.org/).

Java e todas as marcas comerciais e logotipos baseados em Java são marcas comerciais ou marcas registradas da Oracle e/ou de suas afiliadas.

Part Number:

(1P) P/N: