7.5

Mobile Messaging et M2M

Remarque

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section «Remarques», à la page 193.

Cette édition s'applique à la version 7 édition 5 d' IBM® WebSphere MQ et à toutes les éditions et modifications ultérieures, sauf indication contraire dans les nouvelles éditions.

Table des matières

Introduction à MQTT

Découvrez comment envoyer des messages entre des applications mobiles à l'aide de MQ Telemetry Transport (MQTT). Le protocole est destiné à être utilisé sur les réseaux sans fil et à faible bande passante. Une application mobile qui utilise MQTT envoie et reçoit des messages en appelant une bibliothèque MQTT. Les messages sont échangés par l'intermédiaire d'un serveur de messagerie MQTT. Le client et le serveur MQTT gèrent la complexité et la fiabilité de la distribution des messages pour l'app mobile, tout en maintenant un faible coût de gestion du réseau.

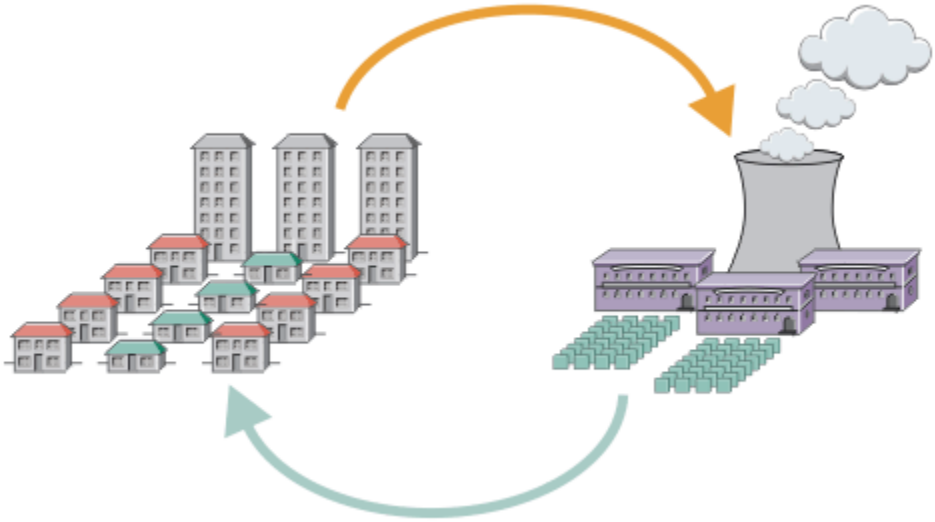
Les applications MQTT fonctionnent sur des appareils mobiles, tels que des téléphones intelligents et des tablettes. MQTT est aussi utilisé dans le cadre de la télémétrie pour recevoir des données des détecteurs et les contrôler à distance. Pour les appareils mobiles et les détecteurs, MQTT constitue un protocole de type publication/abonnement hautement évolutif qui garantit la distribution. Pour envoyer et recevoir des messages MQTT, ajoutez une bibliothèque client MQTT à votre application.

La bibliothèque client MQTT est petite. La bibliothèque agit comme une boîte aux lettres, envoyant et recevant des messages avec d'autres applications MQTT qui sont connectées à un serveur MQTT. En envoyant des messages au lieu de rester connecté à un serveur qui attend une réponse, les MQTT conservent la durée de vie de la batterie. La bibliothèque envoie des messages aux autres appareils par l'intermédiaire d'un serveur MQTT qui exécute le protocole MQTT version 3.1. Vous pouvez envoyer des messages à un client spécifique, ou utiliser la messagerie de type publication/abonnement pour se connecter à un grand nombre d'appareils.

Les bibliothèques clientMQTT connectent les applications des appareils mobiles et des détecteurs à un serveur MQTT à l'aide du protocole MQTT.

IBM MessageSight et IBM WebSphere MQ sont des serveurs MQTT. Ils peuvent connecter d'importants volumes d'applications client MQTT et connecter des réseaux MQTT et IBM WebSphere MQ. Voir «Initiation aux serveurs MQTT», à la page 138. IBM WebSphere MQ et IBM MessageSight peuvent tous deux constituer un pont entre les applications Web externes exécutées sur des périphériques mobiles et des détecteurs, ainsi que d'autres types d'applications de messagerie et de publication/abonnement exécutées au sein de l'entreprise. Le pont facilite la mise en oeuvre de "solutions intelligentes" qui incorporent les appareils mobiles et les détecteurs.

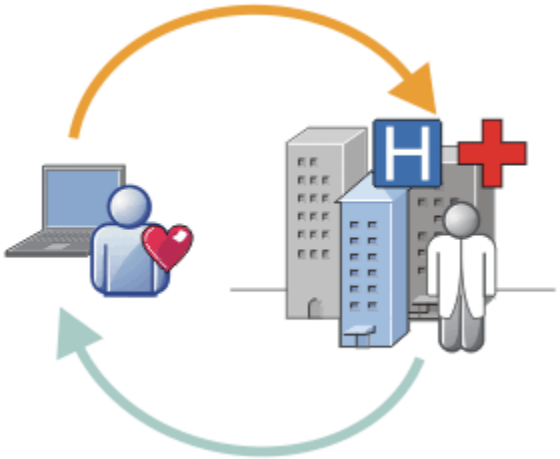
Ces solutions permettent d'utiliser l'abondance d'informations disponibles sur Internet dans les applications fonctionnant sur des appareils mobiles et des détecteurs. L'électricité intelligente et les services de santé intelligents sont deux exemples d'application intelligente basée sur la télémétrie.



- Message MQTT contenant les données relatives à la consommation d'énergie envoyées au fournisseur de services.
- Une application de télémétrie envoie des commandes de contrôle basées sur l'analyse des données relatives à la consommation d'énergie.
- Pour plus d'informations, voir Scénario de télémétrie : surveillance et contrôle de l'énergie à domicile.

Figure 1. Compteur électrique intelligent

Figure 2. Moniteur de santé intelligent



- Une application de télémétrie envoie à votre hôpital et à votre docteur les données relatives à votre santé.
- Des alertes de message MQTT ou des commentaires peuvent être envoyés en fonction de l'analyse des données relatives à votre santé.
- Pour plus d'informations, voir Scénario de télémétrie : surveillance de patient à domicile.

Vous pouvez intégrer MQTT à de petits appareils en écrivant vous-même une app pour le MQTT protocol. Pour vous faciliter la tâche, IBM fournit des bibliothèques client qui prennent en charge les apps qui fonctionnent sur MQTT. Voir «Initiation aux client MQTT», à la page 11. IBM fournit des bibliothèques client pour les applications iOS et pour les Android applications V 7.5.0.1, ainsi qu'un JavaScript client de navigation pour les applications Web indépendantes de la plateforme. V 7.5.0.1 Les pages JavaScript du client se connectent à IBM MessageSight et IBM WebSphere MQ avec le protocole MQTT sur WebSockets. IBM fournit également des modèles d'app MQTT pour C et Java sous Linux® et Windows.

Les bibliothèques C et Java fonctionnent sous iOS, Android, Windows et un certain nombre de plateformes UNIX and Linux. Vous pouvez porter le code source C de la bibliothèque client MQTT sur d'autres plateformes. Les bibliothèques client MQTT pour C et Java sont disponibles avec une licence open source du projet Eclipse Paho. Voir Eclipse Paho. La spécification du MQTT protocol est libre et disponible sur le site MQTT.org.

MQTT protocol

Le MQTT protocol peut être qualifié de léger dans la mesure où les clients sont petits et qu'il utilise de façon efficace la bande passante réseau. Le protocole MQTT garantit la distribution des messages et des transferts "autonomes après diffusion". Dans le protocole, la distribution des messages est découplée de l'application. L'étendue du découplage dans une application dépend de la façon dont sont écrits le client MQTT et le serveur MQTT. La distribution découplée libère l'application des connexions serveur et de l'attente des messages. Le modèle d'interaction est semblable à celui de la messagerie électronique, mais il est optimisé pour la programmation d'application.

Le protocole MQTT V3.1 est publié. Voir MQTT V3.1 Protocol Specification. La spécification identifie un certain nombre de fonctions distinctives du protocole :

- Il s'agit d'un protocole de type publication/abonnement.

  Outre la distribution de type "un à plusieurs" des messages, la fonction de publication/abonnement découple les applications. Ces deux fonctions sont importantes dans les applications qui ont beaucoup de clients.
- Il ne dépend aucunement du contenu des messages.
- Il fonctionne sur TCP/IP, qui fournit la connectivité réseau de base.
- Il propose trois qualités de service pour la distribution des messages :

  "Au plus une fois"
  Le réseau IP sous-jacent fait son maximum pour distribuer les messages. Des messages peuvent être perdus.

  Cette qualité de service est par exemple suffisante pour la communication des données d'un détecteur d'ambiance. La perte d'un relevé individuel est sans importance s'il est rapidement suivi d'un autre.

  "Au moins une fois"
  Les messages arrivent de façon certaine, mais ils peuvent arriver en double.

  "Une seule fois"
  Les messages arrivent de façon certaine, une seule fois.

  Cette qualité de service est par exemple nécessaire aux systèmes de facturation. Les doublons ou les messages perdus peuvent causer des problèmes et générer des demandes de paiement erronées.
- Il gère le flux des messages sur le réseau de manière économique. Par exemple, l'en-tête de longueur fixe ne prend que deux octets, et les échanges sont réduits pour alléger le trafic du réseau.
- Il dispose de la fonction "Dernières volontés et testament" qui notifie les abonnés de la déconnexion anormale d'un client du serveur MQTT. Voir «Publication Last will and testament (dernières volontés et testament)», à la page 131.

MQTT version 3.1 est pris en charge par IBM IBM WebSphere MQ et IBM MessageSight. MQTT est implémenté sur TCP/IP. Une autre version du protocole, MQTT-S, est disponible pour les réseaux non TCP/IP. Voir MQTT-S version 1.2 specification.

Communautés MQTT

IBM met IBM Developer Messaging communauté à la disposition des développeurs MQTT qui écrivent des applications pour IBM MessageSight et IBM WebSphere MQ.

MQTT.org est l'endroit idéal pour discuter et se renseigner sur les implémentations et les extensions du protocole MQTT.

MQTT est un projet Eclipse open source qui dépend de Eclipse Technology Project. La communauté Paho développe des clients et des serveurs open source. Voir Eclipse Paho.

Introduction à MQTT

Découvrez comment envoyer des messages entre des applications mobiles à l'aide de MQ Telemetry Transport (MQTT). Le protocole est destiné à être utilisé sur les réseaux sans fil et à faible bande passante. Une application mobile qui utilise MQTT envoie et reçoit des messages en appelant une bibliothèque MQTT. Les messages sont échangés par l'intermédiaire d'un serveur de messagerie MQTT. Le client et le serveur MQTT gèrent la complexité et la fiabilité de la distribution des messages pour l'app mobile, tout en maintenant un faible coût de gestion du réseau.

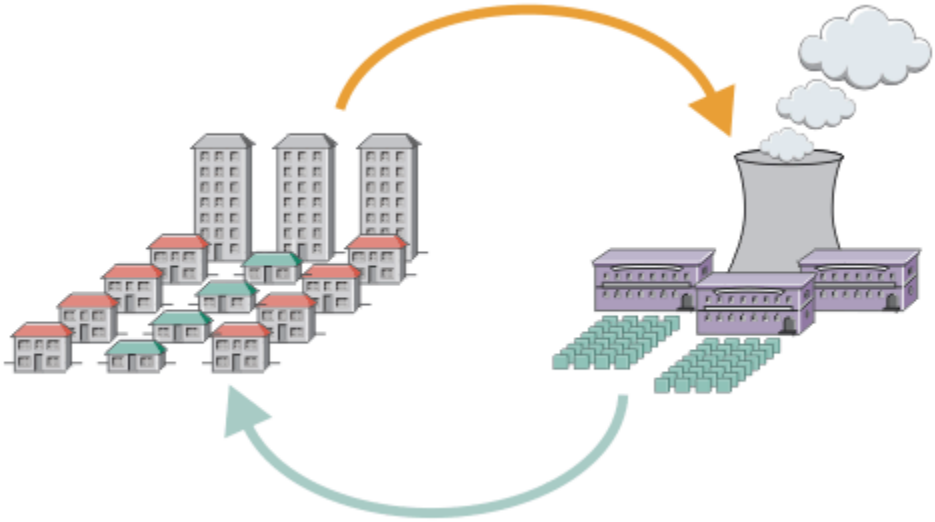
Les applications MQTT fonctionnent sur des appareils mobiles, tels que des téléphones intelligents et des tablettes. MQTT est aussi utilisé dans le cadre de la télémétrie pour recevoir des données des détecteurs et les contrôler à distance. Pour les appareils mobiles et les détecteurs, MQTT constitue un protocole de type publication/abonnement hautement évolutif qui garantit la distribution. Pour envoyer et recevoir des messages MQTT, ajoutez une bibliothèque client MQTT à votre application.

La bibliothèque client MQTT est petite. La bibliothèque agit comme une boîte aux lettres, envoyant et recevant des messages avec d'autres applications MQTT qui sont connectées à un serveur MQTT. En envoyant des messages au lieu de rester connecté à un serveur qui attend une réponse, les MQTT conservent la durée de vie de la batterie. La bibliothèque envoie des messages aux autres appareils par l'intermédiaire d'un serveur MQTT qui exécute le protocole MQTT version 3.1. Vous pouvez envoyer des messages à un client spécifique, ou utiliser la messagerie de type publication/abonnement pour se connecter à un grand nombre d'appareils.

Les bibliothèques clientMQTT connectent les applications des appareils mobiles et des détecteurs à un serveur MQTT à l'aide du protocole MQTT.

IBM MessageSight et IBM WebSphere MQ sont des serveurs MQTT. Ils peuvent connecter d'importants volumes d'applications client MQTT et connecter des réseaux MQTT et IBM WebSphere MQ. Voir «Initiation aux serveurs MQTT», à la page 138. IBM WebSphere MQ et IBM MessageSight peuvent tous deux constituer un pont entre les applications Web externes exécutées sur des périphériques mobiles et des détecteurs, ainsi que d'autres types d'applications de messagerie et de publication/abonnement exécutées au sein de l'entreprise. Le pont facilite la mise en oeuvre de "solutions intelligentes" qui incorporent les appareils mobiles et les détecteurs.

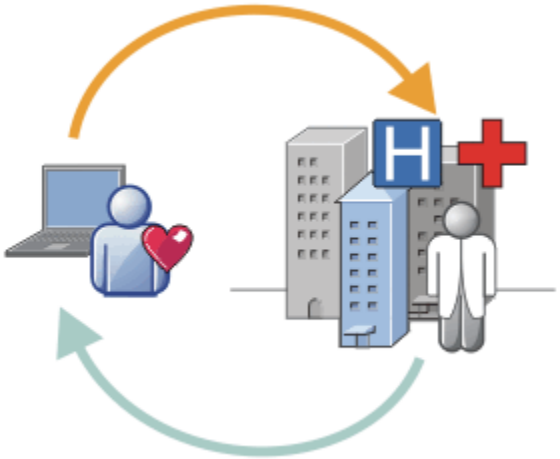
Ces solutions permettent d'utiliser l'abondance d'informations disponibles sur Internet dans les applications fonctionnant sur des appareils mobiles et des détecteurs. L'électricité intelligente et les services de santé intelligents sont deux exemples d'application intelligente basée sur la télémétrie.



- Message MQTT contenant les données relatives à la consommation d'énergie envoyées au fournisseur de services.
- Une application de télémétrie envoie des commandes de contrôle basées sur l'analyse des données relatives à la consommation d'énergie.
- Pour plus d'informations, voir Scénario de télémétrie : surveillance et contrôle de l'énergie à domicile.

Figure 3. Compteur électrique intelligent

Figure 4. Moniteur de santé intelligent



- Une application de télémétrie envoie à votre hôpital et à votre docteur les données relatives à votre santé.
- Des alertes de message MQTT ou des commentaires peuvent être envoyés en fonction de l'analyse des données relatives à votre santé.
- Pour plus d'informations, voir Scénario de télémétrie : surveillance de patient à domicile.

Vous pouvez intégrer MQTT à de petits appareils en écrivant vous-même une app pour le MQTT protocol. Pour vous faciliter la tâche, IBM fournit des bibliothèques client qui prennent en charge les apps qui fonctionnent sur MQTT. Voir «Initiation aux client MQTT», à la page 11. IBM fournit des bibliothèques client pour les applications iOS et pour les Android applications V 7.5.0.1, ainsi qu'un JavaScript client de navigation pour les applications Web indépendantes de la plateforme. V 7.5.0.1 Les pages JavaScript du client se connectent à IBM MessageSight et IBM WebSphere MQ avec le protocole MQTT sur WebSockets. IBM fournit également des modèles d'app MQTT pour C et Java sous Linux et Windows.

Les bibliothèques C et Java fonctionnent sous iOS, Android, Windows et un certain nombre de plateformes UNIX and Linux. Vous pouvez porter le code source C de la bibliothèque client MQTT sur d'autres plateformes. Les bibliothèques client MQTT pour C et Java sont disponibles avec une licence open source du projet Eclipse Paho. Voir Eclipse Paho. La spécification du MQTT protocol est libre et disponible sur le site MQTT.org.

MQTT protocol

Le MQTT protocol peut être qualifié de léger dans la mesure où les clients sont petits et qu'il utilise de façon efficace la bande passante réseau. Le protocole MQTT garantit la distribution des messages et des transferts "autonomes après diffusion". Dans le protocole, la distribution des messages est découplée de l'application. L'étendue du découplage dans une application dépend de la façon dont sont écrits le client MQTT et le serveur MQTT. La distribution découplée libère l'application des connexions serveur et de l'attente des messages. Le modèle d'interaction est semblable à celui de la messagerie électronique, mais il est optimisé pour la programmation d'application.

Le protocole MQTT V3.1 est publié. Voir MQTT V3.1 Protocol Specification. La spécification identifie un certain nombre de fonctions distinctives du protocole :

- Il s'agit d'un protocole de type publication/abonnement.

  Outre la distribution de type "un à plusieurs" des messages, la fonction de publication/abonnement découple les applications. Ces deux fonctions sont importantes dans les applications qui ont beaucoup de clients.
- Il ne dépend aucunement du contenu des messages.
- Il fonctionne sur TCP/IP, qui fournit la connectivité réseau de base.
- Il propose trois qualités de service pour la distribution des messages :

  "Au plus une fois"
  Le réseau IP sous-jacent fait son maximum pour distribuer les messages. Des messages peuvent être perdus.

  Cette qualité de service est par exemple suffisante pour la communication des données d'un détecteur d'ambiance. La perte d'un relevé individuel est sans importance s'il est rapidement suivi d'un autre.

  "Au moins une fois"
  Les messages arrivent de façon certaine, mais ils peuvent arriver en double.

  "Une seule fois"
  Les messages arrivent de façon certaine, une seule fois.

  Cette qualité de service est par exemple nécessaire aux systèmes de facturation. Les doublons ou les messages perdus peuvent causer des problèmes et générer des demandes de paiement erronées.
- Il gère le flux des messages sur le réseau de manière économique. Par exemple, l'en-tête de longueur fixe ne prend que deux octets, et les échanges sont réduits pour alléger le trafic du réseau.
- Il dispose de la fonction "Dernières volontés et testament" qui notifie les abonnés de la déconnexion anormale d'un client du serveur MQTT. Voir «Publication Last will and testament (dernières volontés et testament)», à la page 131.

MQTT version 3.1 est pris en charge par IBM IBM WebSphere MQ et IBM MessageSight. MQTT est implémenté sur TCP/IP. Une autre version du protocole, MQTT-S, est disponible pour les réseaux non TCP/IP. Voir MQTT-S version 1.2 specification.

Communautés MQTT

IBM met IBM Developer Messaging communauté à la disposition des développeurs MQTT qui écrivent des applications pour IBM MessageSight et IBM WebSphere MQ.

MQTT.org est l'endroit idéal pour discuter et se renseigner sur les implémentations et les extensions du protocole MQTT.

MQTT est un projet Eclipse open source qui dépend de Eclipse Technology Project. La communauté Paho développe des clients et des serveurs open source. Voir Eclipse Paho.

Initiation aux client MQTT

Familiarisez-vous avec le développement d'une app mobile ou machine-to-machine (M2M) en générant et en exécutant un modèle d'app client MQTT qui utilise une bibliothèque client MQTT. Les modèles d'app et les bibliothèques client associées sont disponibles dans Module de clientMobile Messaging et M2M d'IBM. Il existe des versions des applications et des bibliothèques client écrites dans Java, dans JavaScriptet dans C. Vous pouvez exécuter ces applications sur la plupart des plateformes et des appareils, y compris les appareils et les produits Android à partir de Apple.

Avant de commencer

Pour générer et exécuter votre application, vous devez être expérimenté dans la génération d'applications pour l'appareil ou la plateforme cible, et le langage de programmation utilisé. Une petite expérience est généralement suffisante pour mettre en oeuvre un modèle d'application sur l'appareil ou la plateforme de votre choix.

Si vous utilisez un serveur MQTT d'entreprise, tel que IBM WebSphere MQ ou IBM MessageSight, vous pouvez échanger des informations entre le modèle d'app et vos apps d'entreprise existantes.

Pourquoi et quand exécuter cette tâche

Vos objectifs sont les suivants :

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.
2. Téléchargez le Module de clientMobile Messaging et M2M.
3. Générez, pour l'appareil ou la plateforme cible, les modèles d'application du pack client.
4. Vérifiez que les modèles se comportent comme prévu en les connectant au serveur MQTT.

En même temps que vous créez et testez les modèles d'application pour votre appareil ou votre plateforme, vous créez un environnement de développement fonctionnel que vous pouvez réutiliser pour générer vos propres applications client.

Module de clientMobile Messaging et M2M contient le kit de développement de logiciels MQTT. Celui-ci contient les ressources suivantes :

- Des apps client MQTT écrites en Java, en JavaScript et en C.
- Des bibliothèques client MQTT qui prennent en charge ces apps client et leur permettent de s'exécuter sur la plupart des plateformes et des appareils.

Le kit de développement de logiciels contient aussi le code source du client MQTT pour C. Vous pouvez adapter ce code source pour générer des bibliothèques client MQTT pour C pour d'autres plateformes. Pour cela, consultez la rubrique «Création des bibliothèques du client MQTT pour C», à la page 31. Le code source du client MQTT pour C est également disponible avec une licence open source d'Eclipse Paho.

Procédure

Les articles suivants vous aident à générer et à exécuter un modèle d'appMQTT sur un ordinateur de bureau ou sur un périphérique mobile pour Android ou d'Apple :

- «Initiation au client MQTT pour Java», à la page 12
- «Initiation au client MQTT pour Java sous Android», à la page 18
- V 7.5.0.1
  «Initiation au client de messagerie MQTT pour JavaScript», à la page 24
- «Initiation au client MQTT pour C», à la page 26

- «Initiation au client MQTT pour C sous iOS», à la page 48

Que faire ensuite

Pour développer une application MQTT, vous devez disposer des compétences dans les domaines suivants :

- Programmation dans la langue requise pour l'appareil ou la plateforme.
- Programmation pour l'appareil ou la plateforme cible.
- Conception d'applications de publication/abonnement.
- Conception de programmes pour le modèle de programmation MQTT.
- Conception de programmes exécutables sur l'appareil mobile sélectionné.
- Utilisation de SSL et de JAAS pour la sécurisation des programmes.

Vous n'avez pas besoin de compétences de programmation réseau pour connecter un client MQTT à un autre appareil ou une autre application, car MQTT est un système de messagerie et de files d'attente. Les bibliothèques client MQTT gèrent les connexions réseau pour l'application.

Vous disposez de deux solutions pour intégrer votre client MQTT aux applications d'entreprise existantes. Vous pouvez partager les sujets de publication/d'abonnement MQTT avec, par exemple, une application IBM WebSphere MQ ou JMS, ou vous pouvez écrire vous-même un adaptateur d'intégration sous la forme d'un autre client MQTT.

Les sources d'information consultables aujourd'hui sont les suivantes :

- Développement d'applications pour WebSphere MQ Telemetry
- MQTT.org
- Eclipse Paho

Concepts associés

«Initiation aux serveurs MQTT», à la page 138

Initiation au client MQTT pour Java

Familiarisez-vous avec les modèles d'application du client MQTT for Java en utilisant IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT. Les modèles d'application utilisent une bibliothèque client du kit de développement de logiciels (SDK) MQTT d'IBM. Le modèle d'application `SampleAsyncCallBack` est un modèle permettant d'écrire des applications MQTT pour Android et d'autres systèmes d'exploitation gérés par des événements.

Avant de commencer

- Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible". Voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.
- S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .

Pourquoi et quand exécuter cette tâche

L'objectif de cette tâche est de vérifier que vous pouvez générer et exécuter un modèle d'application client MQTT pour Java, le connecter à IBM WebSphere MQ ou à IBM MessageSight en tant que serveur MQTT version 3, et échanger des messages.

Réalisez cette tâche pour exécuter le modèle d'application depuis le plan de travail Eclipse, ou depuis une ligne de commande. La procédure suivie dans le modèle s'applique à Windows. Avec de petites modifications, vous pouvez exécuter le modèle d'application sur toutes les plateformes qui prennent en charge JSE version 1.5 ou ultérieure.

Vous pouvez exécuter les applications sur le même serveur que IBM WebSphere MQ, dans lequel l'environnement d'exécution des applications qui se connectent à IBM WebSphere MQ est configuré pour vous. Réalisez la tâche «Configuration du service MQTT depuis la ligne de commande», à la page 142 pour installer et configurer IBM WebSphere MQ avec l'option IBM WebSphere MQ Telemetry sous Windows ou Linux. Lorsque l'environnement est installé et configuré, lancez le modèle d'application, `MQTTV3Sample`, pour vérifier l'installation.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 . Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Facultatif : Configurez le serveur MQTT.
   - Dans IBM WebSphere MQ, vous devez compléter l'une des tâches suivantes pour définir un gestionnaire de files d'attente et configurer ses services de télémétrie (MQXR) :
     - «Configuration du service MQTT depuis la ligne de commande», à la page 142
     - «Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer», à la page 145
   - Pour les autres serveurs, consultez leur documentation. Aucune étape de configuration n'est nécessaire pour Really Small Message Broker. Voir la rubrique Really Small Message Broker.
3. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.

   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.

   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*`\SDK`.
4. Installez un kit de développement Java (JDK) version 6 ou ultérieure.

   Etant donné que vous développez une application Java for Android, JDK doit provenir de Oracle. Vous pouvez récupérer le JDK depuis le site Téléchargements Java SE.
5. Compilez et exécutez un ou plusieurs modèles d'application de client MQTT pour Java :
   - «Compilez et exécutez les modèles de programme Paho depuis la ligne de commande», à la page 14
   - «Compilation et exécution de tous les modèles d'app client Java MQTT depuis Eclipse», à la page 16
   - «Initiation au client MQTT pour Java sous Android», à la page 18

   Les modèles d'app Java MQTT contenus dans le kit de développement de logiciels sont les suivants :

   `MQTTV3Sample`
   Ce modèle est aussi compris dans IBM WebSphere MQ et se lie au package `com.ibm.micro.client.mqttv3.jar`.

   `Sample`
   `Sample` se trouve dans le package Paho et est lié au package `org.eclipse.paho.client.mqttv3` . Il est semblable à `MQTTV3Sample`, et attend la fin de chaque action de MQTT.

   `SampleAsyncWait`
   `SampleAsyncWait` se trouve dans le package `org.eclipse.paho.client.mqttv3`. Il utilise l'API synchrone MQTT, et attend la fin d'une action sur une autre unité d'exécution. L'unité d'exécution principale peut effectuer d'autres travaux pendant qu'il se synchronise sur l'unité qui attend la fin de l'action de MQTT.

SampleAsyncCallBack
SampleAsyncCallBack se trouve dans le package `org.eclipse.paho.client.mqttv3`. Il appelle l'API asynchrone MQTT. L'API asynchrone n'attend pas la fin du traitement d'un appel par MQTT. Elle revient à l'application. Celle-ci poursuit ses autres tâches, et attend l'arrivée du prochain événement pour le traiter. MQTT renvoie une notification d'événement à l'application lorsqu'il a terminé son traitement. L'interface MQTT pilotée par les événements est bien adaptée au modèle de programmation des activités d'Android et des autres systèmes d'exploitation, également pilotés par les événements.

Regardez par exemple la manière dont le modèle `mqttExerciser` intègre MQTT à Android à l'aide du modèle de programmation des activités et des services.

mqttExerciser
`mqttExerciser` est un exemple de programme pour Android. Il est décrit séparément car il est construit et exécuté différemment. Voir «Initiation au client MQTT pour Java sous Android», à la page 18.

Résultats

Vous avez compilé et exécuté des modèles d'application Java MQTT connectés à un serveur IBM WebSphere MQ ou IBM MessageSight en tant que serveur MQTT.

Que faire ensuite

Consultez la documentation Javadoc de référence. Reportez-vous à l'étape «3», à la page 17 de la rubrique «Compilation et exécution de tous les modèles d'app client Java MQTT depuis Eclipse», à la page 16. Vous pouvez également ouvrir les fichiers Javadoc html qui se trouvent dans le répertoire `SDK\clients\java\doc\javadoc` de Module de clientMobile Messaging et M2M.

Compilez et exécutez les modèles de programme Paho depuis la ligne de commande

Compilez et exécutez le Paho modèle d'application `Sample.java` à partir de la ligne de commande. Le modèle se trouve dans le kit de développement de logiciels MQTT. L'exemple est généré avec les bibliothèques client MQTT Paho dans le package `org.eclipse.paho.client.mqttv3` . Il contient un diffuseur de publications et un abonné MQTT. Le même répertoire contient deux autres modèles Paho qui peuvent être générés et exécutés de la même façon. Ils appellent en revanche la bibliothèque MQTT asynchrone.

Avant de commencer

Réalisez les étapes «1», à la page 13 à «4», à la page 13 de la tâche principale pour configurer un serveur MQTT et télécharger Module de clientMobile Messaging et M2M.

Pourquoi et quand exécuter cette tâche

Compilez et exécutez `Sample.java` à partir du sous-répertoire des modèles de client du kit de développement de logiciels, `SDK\clients\java\samples`. Le code Java se trouve dans le répertoire `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`.

Procédure

1. Créez un script dans le répertoire des exemples client pour compiler et exécuter `Sample` sur la plateforme choisie.

   Le script suivant compile et exécute le modèle sous 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
```

Figure 5. Compilez et exécutez `Sample.java`.

2. Exécutez le script.

   Résultats :

```
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
```

Figure 6. Abonné `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
```

Figure 7. Diffuseur de publications `MQTTV3Sample`

   Il n'est pas possible de relancer une application de type abonné qui fonctionne déjà. L'identificateur client du nouvel abonné est le même que celui de l'ancien. Vous ne pouvez pas exécuter simultanément deux clients MQTT portant le même identificateur. L'option `-i` vous permet de définir l'identificateur du client pour pouvoir exécuter simultanément plusieurs abonnés.

   L'option `-c` vous permet de relancer le même client en affectant la valeur False au paramètre `cleansession`. Vous pouvez ainsi explorer le comportement des sessions client interrompues.

3. Mettez fin à l'abonné en appuyant sur Entrée ou en fermant la fenêtre.

Que faire ensuite

Créez des scripts pour compiler et exécuter les autres modèles du sous-répertoire `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app`. Copiez le script de la Figure 5, à la page 15 et remplacez `Sample` par `SampleAsyncWait` ou `SampleAsyncCallBack`. Les autres modèles sont des versions asynchrones du programme synchrone `Sample`.

`SampleAsyncWait`
`SampleAsyncWait` se trouve dans le package `org.eclipse.paho.client.mqttv3`. Il utilise l'API synchrone MQTT, et attend la fin d'une action sur une autre unité d'exécution. L'unité d'exécution principale peut effectuer d'autres travaux pendant qu'il se synchronise sur l'unité qui attend la fin de l'action de MQTT.

`SampleAsyncCallBack`
`SampleAsyncCallBack` se trouve dans le package `org.eclipse.paho.client.mqttv3`. Il appelle l'API asynchrone MQTT. L'API asynchrone n'attend pas la fin du traitement d'un appel par MQTT. Elle revient à l'application. Celle-ci poursuit ses autres tâches, et attend l'arrivée du prochain événement pour le traiter. MQTT renvoie une notification d'événement à l'application lorsqu'il a terminé son traitement. L'interface MQTT pilotée par les événements est bien adaptée au modèle de programmation des activités d'Android et des autres systèmes d'exploitation, également pilotés par les événements.

Regardez par exemple la manière dont le modèle `mqttExerciser` intègre MQTT à Android à l'aide du modèle de programmation des activités et des services.

Les modèles asynchrones montrent comment réduire la durée du blocage d'une application MQTT pendant qu'elle attend le client MQTT. Il est important d'éliminer les appels bloquants dans l'unité d'exécution principale pour améliorer les temps de réponse et conserver la durée de vie de la batterie dans un environnement mobile.

Les modèles présentent deux manières d'appeler les interfaces asynchrones dans le client MQTT.

1. `SampleAsyncWait` ne se bloque pas lorsque le MQTT attend des interactions réseau.
2. `SampleAsyncCallBack` ne se bloque pas en attendant MQTT, quelles que soient les actions qu'il réalise. Ce dernier est nécessaire lorsqu'une page JavaScript appelle le client MQTT depuis un navigateur. Les pages JavaScript ne doivent pas se bloquer. Les réponses aux actions doivent être renvoyées à l'unité d'exécution principale, qui appelle le gestionnaire d'événements MQTT écrit par vous pour traiter la notification.

Compilation et exécution de tous les modèles d'app client Java MQTT depuis Eclipse

Compilez et exécutez les modèles d'application Java du client MQTT qui se trouvent dans le Module de clientMobile Messaging et M2M. Ils contiennent un diffuseur de publications et un abonné MQTT.

Pourquoi et quand exécuter cette tâche

Compilez et exécutez les exemples MQTT Java , `Sampleet` `MQTTV3Sample` dans Eclipse. `Sample` se trouve dans le sous-répertoire des clients SDK *`sdkroot`*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` et `MQTTV3Sample.java` se trouve dans *`sdkroot`*`\SDK\clients\java\samples`.

Procédure

1. Téléchargez Eclipse IDE for Java Developers.
2. Créez un projet Java appelé `MQTT Samples` dans Eclipse.
   a) **Fichier** > **Nouveau projet Java** > et entrez `MQTT Samples`. Cliquez sur **Suivant**.

      Vérifiez que la version de l'environnement d'exécution Java est correcte ou ultérieure.JSE doit être à la version 1.5 ou ultérieure.

   b) Dans la fenêtre **Paramètres Java**, cliquez sur l'option de **liaison de dossiers source supplémentaires**.

   c) Accédez au répertoire dans lequel vous avez installé le dossier du kit de développement de logiciels Java MQTT. Sélectionnez le dossier *`sdkroot`*`\SDK\clients\java\samples` et cliquez sur **OK** > **Suivant** > **Terminer**.

   d) Dans la fenêtre **Paramètres Java**, cliquez sur **Bibliothèques** > **Ajouter des fichiers JAR externes...**.

   e) Accédez au répertoire dans lequel vous avez installé le dossier du kit de développement de logiciels Java MQTT. Localisez le dossier *`sdkroot`*`\SDK\clients\java` et sélectionnez les fichiers `org.eclipse.paho.client.mqttv3.jar` et `com.ibm.micro.client.mqttv3.jar` ; cliquez sur **Ouvrir** > **Terminer**.

      Le modèle `MQTTV3Sample.java` se lie à `com.ibm.micro.client.mqttv3.jar`, et les modèles de l'arborescence de répertoires paho se lient à `org.eclipse.paho.client.mqttv3.jar`. Le fichier `com.ibm.micro.client.mqttv3.jar` est conservé pour que les applications MQTT existantes continuent de générer et de s'exécuter sans modification.

   Le projet `MQTT Samples` se construit avec des avertissements, mais sans erreur.

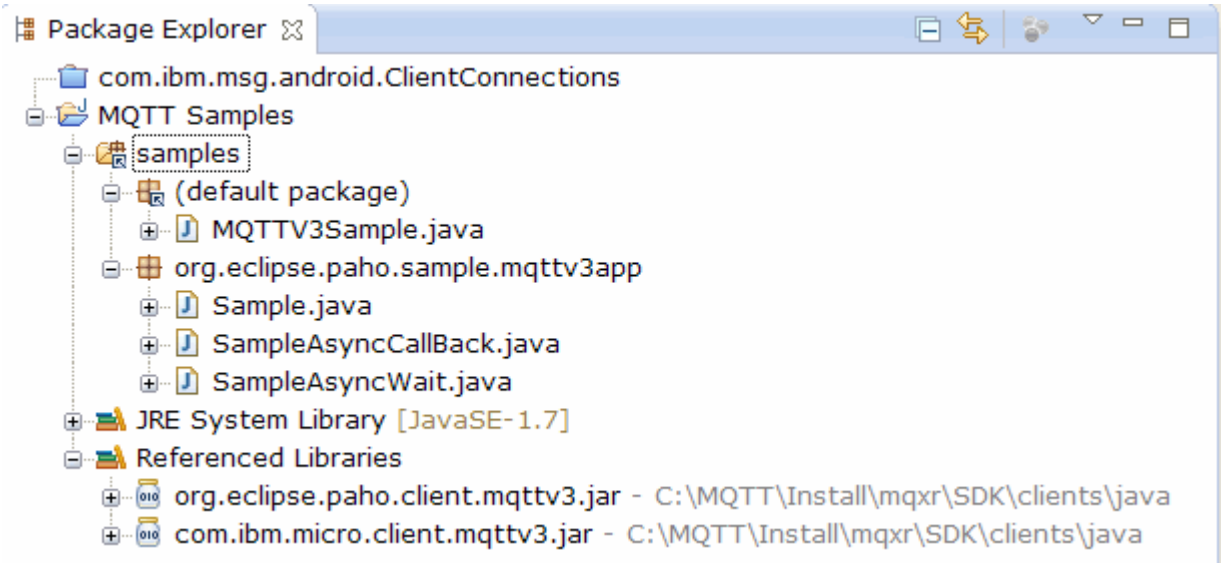


Figure 8. Client MQTT pour le projet Java

3. Facultatif : Installez le Javadoc du client MQTT.

   Lorsque le client MQTT Javadoc est installé, l'éditeur Java décrit les classes MQTT dans l'infobulle.

   a) Ouvrez **Explorateur de packages** > **Bibliothèques référencées** dans votre projet Java. Cliquez avec le bouton droit de la souris sur `org.eclipse.paho.client.mqttv3.jar` **> Propriétés**.

   b) Dans le navigateur des propriétés, cliquez sur **Emplacement Javadoc**.

   c) Cliquez sur **URL Javadoc** > **Parcourir** dans la page **Emplacement Javadoc** et recherchez le dossier `SDK\clients\java\doc\javadoc` **> OK**.

   d) Cliquez sur **Valider** > **OK**.

      Vous êtes invité à ouvrir le navigateur pour afficher la documentation.

   e) Répétez cette procédure pour le fichier `com.ibm.micro.client.mqttv3.jar`.

4. Créez une configuration d'exécution de diffuseur de publications et d'abonné pour l'application `mqttv3app.Sample`.

   a) Cliquez avec le bouton droit de la souris sur la classe **Sample** , puis cliquez sur **Exécuter en tant que** > **Configurations d'exécution**.

   b) Cliquez avec le bouton droit de la souris sur **Application Java** > **Nouveau** et entrez le nom `SampleSubscriber`.

   c) Cliquez sur l'onglet des arguments, et entrez les arguments du programme, puis cliquez sur **Apply**.

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

   d) Répétez la dernière étape pour créer une configuration `SamplePublisher`, en omettant le paramètre `-a subscribe`.

5. Exécutez l'abonné `mqttv3app.Sample`, puis le diffuseur de publications.

   a) Cliquez sur **Run** > **Run configurations**.

   b) Cliquez sur **SampleSubscriber** > **Exécuter**.

      Ouvrez la vue **Console**. L'abonné attend une publication.

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

   c) Cliquez sur **SamplePublisher** > **Exécuter**.

Ouvrez la vue **Console**. Vous voyez la publication qui est créée par le diffuseur de publications :

```
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) Passez dans la vue console de l'abonné.

L'icône de permutation des consoles est  .

L'abonné a reçu la publication :

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

6. Facultatif : Créez une configuration d'exécution de diffuseur de publications et d'abonné pour `MQTTV3Sample`.
   a) Cliquez avec le bouton droit de la souris sur la classe **MQTTV3Sample** , puis cliquez sur **Exécuter en tant que** > **Configurations d'exécution**.
   b) Cliquez avec le bouton droit de la souris sur **Application Java** > **Nouveau** et entrez le nom `MQTTV3SampleSubscriber`.
   c) Cliquez sur l'onglet des arguments, et entrez les arguments du programme, puis cliquez sur **Apply**.

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

   d) Répétez la dernière étape pour créer une configuration `MQTTV3SamplePublisher`, en omettant le paramètre `-a subscribe`.
7. Facultatif : Exécutez l'abonné `MQTTV3Sample`, puis le diffuseur de publications.
   a) Cliquez sur **Run** > **Run configurations**.
   b) Cliquez sur **MQTTV3SampleSubscriber** > **Exécuter**.

Ouvrez la vue **Console**. L'abonné attend une publication.

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

   c) Cliquez sur **MQTTV3SamplePublisher** > **Exécuter**.

Ouvrez la vue **Console**. Vous voyez la publication qui est créée par le diffuseur de publications.

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

   d) Passez dans la vue console de l'abonné.

L'icône de permutation des consoles est  .

L'abonné a reçu la publication :

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

Initiation au client MQTT pour Java sous Android

Vous pouvez installer un Exemple d'application Java de client MQTT pour Android qui échange des messages avec un serveur MQTT. L'application utilise une bibliothèque client du kit de développement de logiciels (SDK) MQTT d'IBM. Vous pouvez générer l'application vous-même ou télécharger un modèle d'app préconfiguré.

Avant de commencer

- Pour connaître les plateformes client MQTT prises en charge et celles de référence, voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.
- S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .
- Le modèle d'app client MQTT fonctionne sur Ice Cream Sandwich (Android versions 4.0 et supérieures). Cette version d'Android donne également une résolution d'écran plus fine sur les tablettes.

Pourquoi et quand exécuter cette tâche

Le Exemple d'application Java de client MQTT pour Android s'appelle "`mqttExerciser`". Celle-ci utilise une bibliothèque client du kit de développement de logiciels MQTT et échange des messages avec un serveur MQTT.

Vous pouvez soit générer le modèle d'application vous-même, puis l'exporter à partir de Eclipse en tant que `mqttExerciser.apk`, soit utiliser le modèle d'application préconfiguré disponible sous forme de fichier `mqttExerciser.apk` dans le dossier *`sdkroot`*`\SDK\clients\android\samples\apks` de Module de clientMobile Messaging et M2M. Si vous choisissez de générer l'application vous-même, l'environnement de développement généré permet d'inclure des services de messagerie mobile dans les applications for Android. Cela devrait vous aider lorsque vous commencerez à inclure la messagerie mobile dans vos propres applications.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 . Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Utilisez les outils appropriés.

   Installez un kit de développement Java (JDK) version 6 ou ultérieure. Etant donné que vous développez une application Java for Android, JDK doit provenir de Oracle. Vous pouvez récupérer le JDK depuis le site Téléchargements Java SE.

   Vous avez également besoin d'un environnement de développement Eclipse. Il doit s'agir d'Eclipse version 3.6.2 (Helios) ou supérieure. Eclipse doit disposer d'un compilateur Java de niveau 6 au minimum pour correspondre à votre JDK. Vous pouvez obtenir tout cela à partir de Eclipse Foundation.

   Enfin, il vous faut le kit de développement de logiciels (SDK) Android. Vous pouvez le récupérer sous Obtenir le kit de développement de logiciels Android.
3. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.

   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.

   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*`\SDK`.
4. Facultatif : Créez le modèle d'application for Android mqttExerciser.

   Configurez les outils Eclipse et Android , puis importez et générez le projet `mqttExerciser` à partir du SDK MQTT .

   Remarque : Si vous ne souhaitez pas effectuer cette opération maintenant, vous pouvez utiliser le modèle d'application préconfiguré disponible sous forme de fichier `mqttExerciser.apk` dans le dossier *`sdkroot`*`\SDK\clients\android\samples\apks` du SDK MQTT .

a) Démarrez l'environnement de développement Eclipse avec l'environnement d'exécution Java (JRE) depuis le kit de développement Java (JDK).

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

b) Sélectionnez et installez un ensemble de packages et de plateformes du SDK Android.

Voir Ajout de plateformes et de packages pour la liste des plateformes et des packages recommandés par Google .

Remarque : La plateforme SDK doit être au minimum au niveau de l'API 16 d'Android. Le projet n'est pas compilable avec un niveau d'API inférieur.

c) Ajoutez le plug-in Android Development Tools (ADT) à Eclipse.

d) Importez le projet du modèle d'app `mqttExerciser` dans Eclipse, et corrigez les erreurs.

i) Importez l'exemple de projet d'application à partir du SDK MQTT , dans le chemin *sdkroot*\SDK\clients\android\samples\mqttExerciser.

La vue **Problems** contient de nombreuses erreurs liées à la génération. Elles seront résolues dans les étapes suivantes.

ii) Copiez la bibliothèque `org.eclipse.paho.client.mqttv3.jar` dans le dossier **libs** du projet Android . Windows Par exemple, sous Windows, il se trouve dans le dossier *sdkroot*\SDK\clients\java . La fenêtre **File Operation** s'affiche. Conservez la sélection **Copy files**, puis cliquez sur **OK**.

iii) Cliquez avec le bouton droit de la souris sur le dossier de projet, `com.ibm.msg.android`; cliquez sur **Outils Android ...** > **Ajouter une bibliothèque de support ...**. Lisez et acceptez les termes de la licence, puis cliquez sur **Install**.

iv) Cliquez avec le bouton droit de la souris sur le dossier de projet, `com.ibm.msg.android`; cliquez sur **Outils Android ...** > **Corriger les propriétés du projet**.

v) Si l'espace de travail contient encore environ 84 erreurs faisant référence à l'outrepassement d'une méthode de superclasse, le niveau de compatibilité du compilateur est probablement défini sur 1.5 ou une valeur inférieure. Pour la version 16 du SDK Android, le niveau de compatibilité du compilateur ne doit pas être supérieur à 1.5. Pour résoudre les erreurs restantes, procédez comme suit :

a) Vérifiez et (si nécessaire), mettez à jour votre logiciel SDK Android ainsi que les plug-in Eclipse correspondants vers la version 17 du logiciel SDK Android.

b) Cliquez avec le bouton droit de la souris sur le dossier de projet **com.ibm.msg.android** , puis sélectionnez **Propriétés** > **Java Compiler**. Vérifiez le niveau de compatibilité du compilateur, définissez une valeur au moins égale à 1.6, puis régénérez l'espace de travail.

Le projet se construit avec des avertissements, mais sans erreur.

5. Installez et démarrez le Exemple d'application Java client MQTT sur un périphérique Android.

Voir la page `developer.android.com` Exécution de votre application.

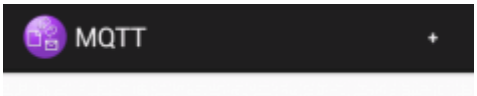
Si vous avez généré vous-même l'application sous la forme d'un projet Eclipse, vous pouvez la démarrer depuis Eclipse.

Si vous disposez du fichier d'application .APK `mqttExerciser.apk`, vous pouvez l'installer en dehors d'Eclipse à l'aide de la commande d'installation Android Debug Bridge (ADB). Cette commande prend comme argument l'emplacement du fichier APK. Si vous utilisez le modèle d'application préconfiguré, l'emplacement est *sdkroot*\SDK\clients\android\samples\apks\mqttExerciser.apk.

6. Utilisez le modèle d'application for Android mqttExerciser pour la connexion, l'abonnement et la publication dans un sujet.

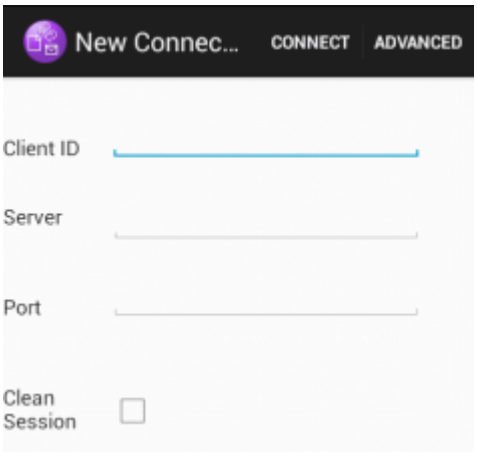
a) Ouvrez le Exemple d'application Java de client MQTT pour Android.

Cette fenêtre est ouverte sur votre périphérique Android :



b) Connectez-vous à un serveur MQTT .

i) Cliquez sur le symbole **+** pour ouvrir une nouvelle connexion à MQTT.



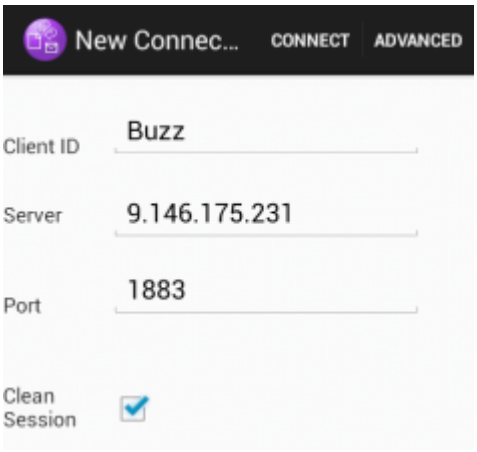
ii) Entrez un identifiant unique de votre choix dans la zone **ID client**. Soyez patient, car la saisie peut être lente.

iii) Dans la zone **Serveur**, entrez l'adresse IP de votre serveur MQTT.

Il s'agit du serveur que vous avez sélectionné à la première étape principale. L'adresse IP ne doit pas être `127.0.0.1`.

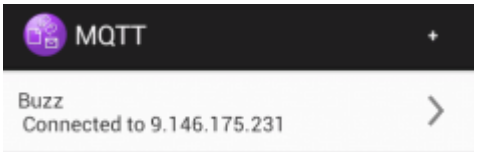
iv) Entrez le numéro de port de la connexion MQTT.

Le numéro de port par défaut pour une connexion MQTT normale est 1883.



v) Cliquez sur **Connect**.

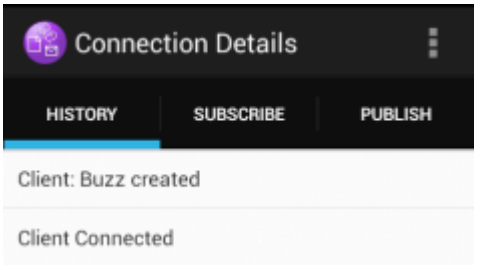
Si la connexion est établie, le message "Connexion" apparaît, suivi de cette fenêtre :



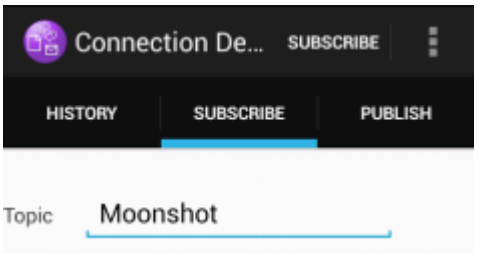
c) Abonnez-vous à une rubrique.

i) Cliquez sur le message **Connecté**.

La fenêtre **Détails de connexion** s'ouvre et affiche l'historique :



ii) Cliquez sur l'onglet **Abonnement** et entrez une chaîne de rubrique.

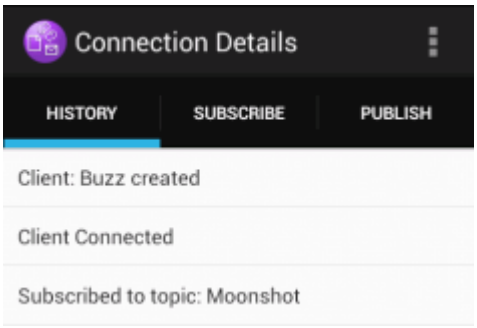


iii) Cliquez sur l'action **S'abonner**.

Un message "Abonné" s'affiche rapidement.

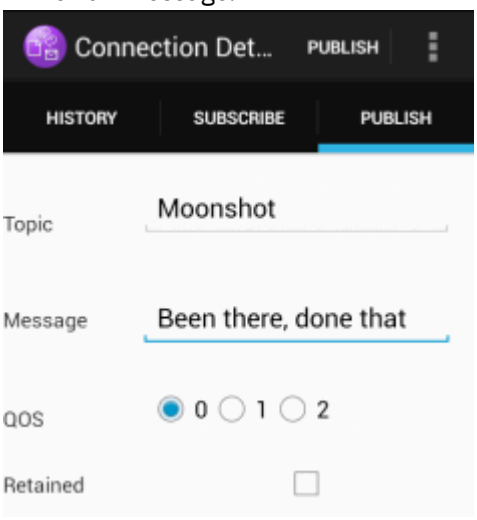
iv) Cliquez sur l'onglet **Historique**.

L'historique contient maintenant l'abonnement :



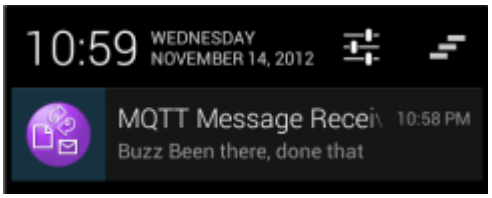
d) Diffusez maintenant une publication dans la même rubrique.

i) Cliquez sur l'onglet **Publication** et entrez la même chaîne de rubrique que pour l'abonnement. Entrez un message.

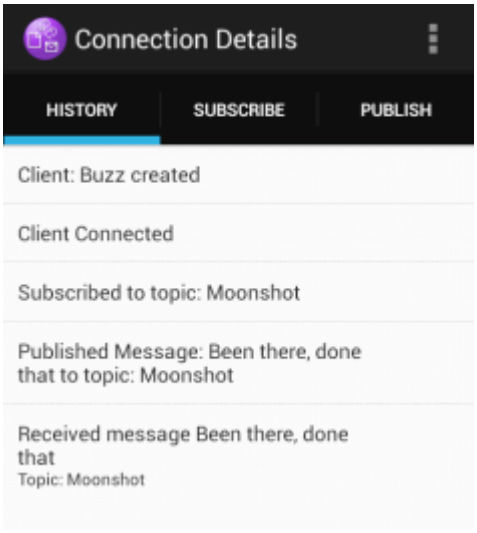


ii) Cliquez sur l'action **Publier**.

Deux messages s'affichent rapidement, "Publié" suivi par "Abonné". La publication s'affiche dans la zone d'état (faites glisser vers le bas la barre de séparation pour ouvrir la fenêtre de statut).



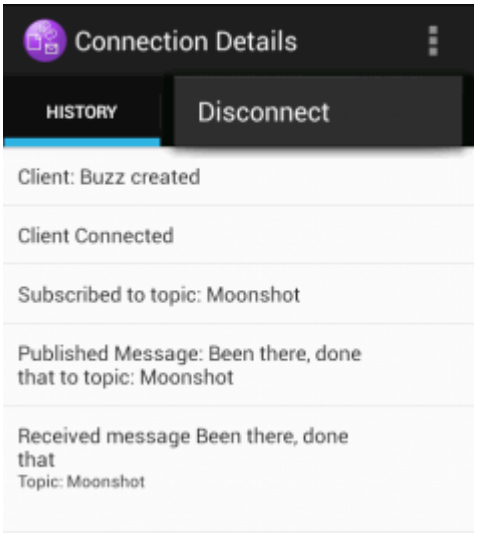
iii) Cliquez sur l'onglet **Historique** pour afficher l'historique en totalité.



e) Déconnectez l'instance du client.

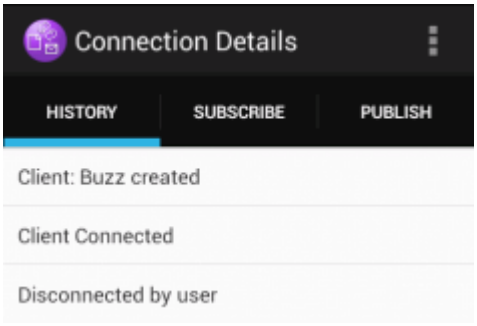
i) Cliquez sur l'icône de menu dans la barre d'actions. 

Le Exemple d'application Java de client MQTT pour Android ajoute un bouton **Déconnexion** à la fenêtre MQTT **Détails de connexion**.

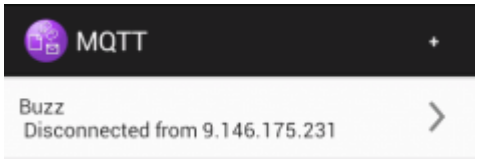


ii) Cliquez sur **Déconnexion**.

De Connecté, le statut passe à Déconnecté :



f) Cliquez sur **Précédent** pour revenir à la liste des sessions du Exemple d'application Java client MQTT.



- Cliquez sur le symbole plus pour lancer une nouvelle session du Exemple d'application Java client MQTT.
- Cliquez sur le client déconnecté pour le reconnecter.
- Cliquez sur **Précédent** pour revenir au tableau de bord.

g) Cliquez sur le bouton Tâche pour lister les applications en cours. Localisez le Exemple d'application Java client MQTT. Faites glisser l'icône en dehors de l'écran pour le fermer.

Que faire ensuite

Si vous avez généré le modèle d'application vous-même, vous êtes prêt à commencer à développer vos propres applications Android qui appellent les bibliothèques MQTT pour échanger des messages. Vous pouvez modéliser vos applications Android sur les classes dans `mqttExerciser`. Pour analyser l'exemple, générez la documentation Javadoc des classes de `com.ibm.msg.android` et `com.ibm.msg.android.service` du projet `mqttExerciser`.

Information associée

Gestion des projets dans Eclipse avec Android Development Tools

Initiation au client de messagerie MQTT pour JavaScript

Vous pouvez vous familiariser avec le client de messagerie MQTT pour JavaScript en affichant la page d'accueil du modèle de client de messagerie, et en navigant dans les ressources accessibles par les liens. Pour afficher cette page d'accueil, configurez un serveur MQTT de manière qu'il accepte les connexions du Pages de l'exemple de client de messagerie MQTT JavaScript, puis entrez l'URL que vous avez configurée sur le serveur dans un navigateur Web. Le client de messagerie MQTT pour JavaScript démarre automatiquement sur votre périphérique, et la page d'accueil du modèle de client de messagerie s'affiche. Cette page contient des liens vers des utilitaires, la documentation de l'API, un tutoriel et d'autres informations utiles.

Avant de commencer

Pour une utilisation avancée, ou une utilisation en production, vous voudrez remodeler ou retirer la page d'accueil du client de messagerie. Notez que la compatibilité des interfaces utilisateur résultant du modèle de code avec les standards et les exigences d'accessibilité n'est pas garantie.

Vous avez besoin d'un serveur MQTT pour prendre en charge le client de messagerie MQTT pour JavaScript. Ce serveur doit prendre en charge le protocole MQTT V3.1 sur WebSockets. IBM MessageSight, et IBM WebSphere MQ Version 7.5.0, Fix Pack 1 et versions ultérieures, prennent en charge MQTT protocol sur WebSockets. Voir «Initiation aux serveurs MQTT», à la page 138. Pour installer

IBM WebSphere MQ pour une évaluation de 90 jours, voir «InstallationIBM WebSphere MQ», à la page 141.

Le WebSocket protocol a été établi récemment. S'il existe un pare-feu entre votre client et le serveur, vérifiez que celui-ci ne bloque pas le trafic WebSockets. De même, si votre navigateur ne prend pas encore en charge WebSocket protocol[1]vous ne pourrez pas utiliser l'utilitaire client ou les tutoriels disponibles à partir de la page d'accueil du modèle de client de messagerie. Le tableau Tableau 1, à la page 25 répertorie les navigateurs dont les dernières versions ont été testées et fonctionnent avec le client de messagerie.

Tableau 1. Navigateurs pris en charge pour une utilisation avec le client de messagerie MQTT pour JavaScript

Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 et versions ultérieures Chrome for Android 25.0 et versions ultérieures	Safari 6.0 et versions ultérieures Chrome 14.0 et versions ultérieures	Firefox 6.0 et versions ultérieures Chrome 14.0 et versions ultérieures	Firefox 6.0 et versions ultérieures Chrome 14.0 et versions ultérieures

Pourquoi et quand exécuter cette tâche

La plupart des opérations de cette tâche configurent le serveur MQTT. La seule chose nécessaire pour accéder au client de messagerie pour JavaScript est d'utiliser un navigateur qui prend en charge WebSocket protocol.

Sur IBM WebSphere MQ, suivez la procédure d'activation de IBM WebSphere MQ Telemetry en créant les modèles de canaux. Etablissez la connexion au canal MQTT WebSockets par défaut sur le port 1883. L'URL de l'exemple de page d'accueil du client de messagerie est `http://`*`hostname`*`:1883` sur IBM WebSphere MQ.

Sur IBM MessageSight, installez et configurez le dispositif, configurez le concentrateur de messagerie pour qu'il accepte les connexions, et créez un noeud final MQTT WebSockets.

Procédure

1. Téléchargez le Module de clientMobile Messaging et M2M, et choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Voir «Initiation aux client MQTT», à la page 11.
2. Configurez votre serveur MQTT pour qu'il accepte les connexions des pages HTML du modèle de client de messagerie MQTT pour JavaScript.
   - Sous IBM WebSphere MQ :
     - Si vous disposez déjà d'un gestionnaire de files d'attente IBM WebSphere MQ configuré pour MQTT, modifiez le protocole dans la définition du canal pour prendre en charge à la fois MQTT et HTTP. Voir **`ALTER CHANNEL`**.
     - Pour créer un gestionnaire de files d'attente IBM WebSphere MQ et configurer l'exemple de noeud final MQTT WebSockets, effectuez l'une des tâches suivantes :
       - «Configuration du service MQTT depuis la ligne de commande», à la page 142
       - «Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer», à la page 145
3. Ouvrez un navigateur Web sur votre appareil.
4. Entrez l'URL de la page d'accueil du client de messagerie.
   - Sous IBM WebSphere MQ, il s'agit de `http://`*`hostname`*`:1883`

[1] En particulier, s'il ne prend pas en charge la norme RFC 6455 (WebSocket).

- Sous IBM MessageSight, il s'agit de `http://hostname:port`

  où *hostname* est le nom DNS ou l'adresse IP du socket Ethernet que vous avez configuré sur votre dispositif IBM MessageSight comme noeud final auquel le client doit se connecter et *port* est le numéro de port TCP/IP que vous avez affecté au noeud final pour le client.

La page d'accueil du client de messagerie s'affiche.

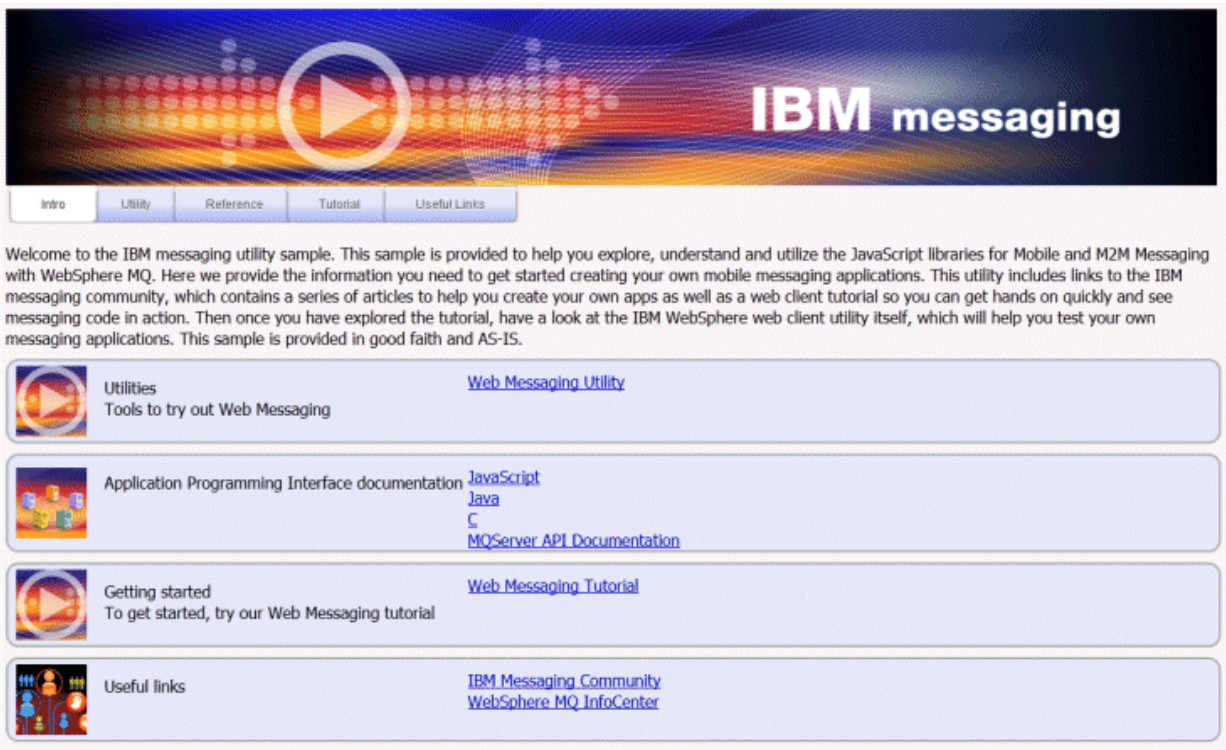


Figure 9. Page d'accueil du modèle de client de messagerie MQTT pour JavaScript

Résultats

Vous avez configuré un canal MQTT pour WebSockets.

Dans la page d'accueil du modèle de client de messagerie MQTT pour JavaScript, cliquez sur **Web Messaging Utility** pour tester différentes fonctions de l'API du client de messagerie. Par exemple, vous pouvez vous connecter au gestionnaire de files d'attente, vous abonnez à des messages ou en publier. Vous pouvez aussi cliquer sur **Web Messaging Tutorial** pour apprendre à créer une page Web qui appelle le client de messagerie MQTT pour l'API JavaScript.

Concepts associés

«Le client de messagerie MQTT pour JavaScript et les apps Web», à la page 118

«Comment programmer des apps de messagerie dans JavaScript», à la page 122

Tâches associées

«Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

Initiation au client MQTT pour C

Familiarisez-vous avec le modèle de client MQTT pour C sur toutes les plateformes sur lesquelles vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle de client MQTT pour C avec IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

Avant de commencer

- S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .
- Pour connaître les plateformes client MQTT pour C prises en charge et celles de référence, voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.

Pourquoi et quand exécuter cette tâche

Suivez cette tâche pour compiler et exécuter le modèle de client MQTT pour C sous Windows à partir de la ligne de commande ou de Microsoft Visual Studio 2010. Microsoft Visual Studio 2010 est aussi utilisé pour compiler le client dans le modèle pour la ligne de commande. Modifiez les scripts de ligne de commande pour compiler et exécuter le modèle sur d'autres plateformes.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 . Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Installez un environnement de développement C sur la plateforme de génération.

   Les fichiers makefile dans les exemples de cette rubrique ciblent les outils suivants :

   - 

Pour iOS, sous Apple Mac avec OS X 10.8.2 et les outils de développement iOS du site Xcode.
   - 

Pour Linux, la version gcc 4.4.6 de Red Hat® Enterprise Linux version 6.2.

     Le plus bas niveau pris en charge de la bibliothèque C `glibc` est 2.12, et celui du noyau Linux est 2.6.32.
   - 

Pour Microsoft Windows, Visual Studio version 10.0.
3. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.

   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.

   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*`\SDK`.
4. Facultatif : Suivez les étapes décrites dans «Création des bibliothèques du client MQTT pour C», à la page 31.

   Ne réalisez cette étape que si Module de clientMobile Messaging et M2M ne contient pas la bibliothèque client C destinée à votre système d'exploitation.
5. Compilez et exécutez l'exemple d'application C du client MQTT , `MQTTV3Sample.c`.

   - Sur la ligne de commande, suivez la procédure de la rubrique «Compilation et exécution du modèle d'app C client MQTT depuis la ligne de commande», à la page 28.
   - A partir d'un environnement de développement intégré, suivez la procédure de la rubrique «Compilation et exécution du modèle d'app C client MQTT à partir de Microsoft Visual Studio», à la page 29.

Compilation et exécution du modèle d'app C client MQTT depuis la ligne de commande

Compilez et exécutez le modèle d'app C client MQTT depuis la ligne de commande. Le modèle se trouve dans le kit de développement de logiciels MQTT. Il contient un diffuseur de publications et un abonné MQTT.

Avant de commencer

Installez un environnement de développement C, par exemple Microsoft Visual Studio 2010 tel qu'il est utilisé dans l'exemple.

Pourquoi et quand exécuter cette tâche

Compilez et exécutez l'exemple C, `MQTTV3Sample`, dans le sous-répertoire des clients SDK, *sdkroot*`\SDK\clients\c\samples`.

Procédure

Créez un script dans le répertoire des exemples client pour compiler et exécuter `Sample` sur la plateforme choisie.

Le script suivant compile et exécute le modèle sur une plateforme Windows 32 bits, générée avec Microsoft Visual Studio 2010. Exécutez le script à partir du sous-répertoire *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
```

Résultats

Le diffuseur de publications et l'abonné écrivent leur sortie dans leur fenêtre de commande :

```
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 . . .
```

Figure 10. Sortie du diffuseur de publications

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

Figure 11. Sortie de l'abonné

Compilation et exécution du modèle d'app C client MQTT à partir de Microsoft Visual Studio

Compilez et exécutez le modèle d'app C client MQTT à partir de Microsoft Visual Studio. Le modèle se trouve dans Module de clientMobile Messaging et M2M. Il contient un diffuseur de publications et un abonné MQTT.

Avant de commencer

Le modèle utilise Microsoft Visual Studio 2010. Vous pouvez utiliser d'autres environnements de développement C sous Windows et sur d'autres plateformes, par exemple Eclipse IDE for C/C++ Developers.

Pourquoi et quand exécuter cette tâche

Compilez et exécutez l'exemple C, `MQTTV3Sample` avec Microsoft Visual Studio 2010. `MQTTV3Sample.c` se trouve dans le sous-répertoire des clients SDK, *sdkroot*`\SDK\clients\c\samples`.

Procédure

1. Démarrez Microsoft Visual Studio.
2. Créez un projet à partir de code existant.
   a) Cliquez sur **Fichier** > **Nouveau projet** > **à partir du code existant**.
   b) Sélectionnez **Visual C++** comme type de projet à créer.
   c) Cliquez sur **Suivant**.
3. Définissez les paramètres de la fenêtre **Spécifier l'emplacement du projet et les fichiers sources**.
   a) Cliquez sur **Parcourir** et localisez le répertoire *sdkroot*`\SDK\clients\c\samples` .
   b) Nommez le projet `MQTTV3Sample`.
   c) Cliquez sur **Suivant**.
4. Sélectionnez **Projet d'application console** dans la liste **Type de projet**. Cliquez sur **Terminer**.
5. Configurez uniquement la configuration Déboguer.

   Par défaut, Microsoft Visual Studio crée une configuration release et une configuration de débogage. Dans le tutoriel, vous configurez la configuration de débogage. Pour supprimer les erreurs de génération, désélectionnez l'option **Générer** pour la version release.

   a) Cliquez sur **Projet** > **MQTTV3Sample Propriétés** > **Gestionnaire de configuration**. Sélectionnez la **Configuration de la solution active Release** et désélectionnez **Générer**.
   b) Sélectionnez **Déboguer** comme **Configuration de la solution active** > **Fermer**.

   Dans les étapes qui suivent, vérifiez que vous modifiez la configuration de débogage.
6. Modifies les paramètres **C/C++** dans les **Pages de propriétés de MQTTV3Sample**.
   a) Dans la fenêtre **MQTTV3Sample Pages de propriétés** , ouvrez **Propriétés de configuration** > **C/C++** > **Général**.
   b) Dans la liste des propriétés générales, cliquez sur **Autres répertoires d'inclusion**, ajoutez votre chemin de répertoire à *sdkroot*`\SDK\clients\c\include` et cliquez sur **Appliquer**.
7. Modifiez les paramètres de l'**Editeur de liens**.
   a) Ouvrez **Propriétés de configuration** > **Editeur de liens** > **Général**.
   b) Dans la liste des propriétés générales, cliquez sur **Répertoires de bibliothèques supplémentaires**et ajoutez votre chemin de répertoire à *sdkroot*`\SDK\clients\c\windows_ia32`
   c) Dans la liste des propriétés de l'éditeur de liens, cliquez sur **Ligne de commande**. Entrez `mqttv3c.lib` dans la zone de saisie **Options supplémentaires** et cliquez sur **Appliquer**.
8. Retirez le fichier source `MQTTV3SSample.c` du projet.

a) Ouvrez le dossier **MQTTV3sample** > **Fichiers source** dans la fenêtre **Explorateur de solutions** .

b) Cliquez avec le bouton droit de la souris sur **MQTTV3SSample.c** > **Exclure du projet**

9. Générez le projet `MQTTV3Sample`.

a) Cliquez avec le bouton droit sur le projet **MQTTV3sample** dans l'Explorateur de solutions et cliquez sur **Générer**.

La génération se termine sans erreur.

10. Ajoutez deux nouveaux projets pour exécuter **`MQTTV3Sample`** comme instances d'exécution Subscriber (abonné) et Publisher (diffuseur de publications).

Les projets Publisher et Subscriber doivent contenir les commandes d'exécution de **`MQTTV3Sample`**. Ils ne sont pas générés et ne contiennent pas de code.

a) Dans l' **Explorateur de solutions**, cliquez avec le bouton droit de la souris sur **Solution ` MQTTV3Sample `** > **Ajouter** > **Nouveau projet**.

b) Entrez `Subscriber` dans la zone **Nom**. Conservez la sélection **Application console Win32**. Cliquez sur **OK**.

L'**Assistant Application Win32** démarre.

c) Dans l'**Assistant Application Win32**, cliquez sur **Suivant**. Cochez **Projet vide** > **Terminer**

d) Répétez ces opérations pour ajouter le projet Publisher.

e) Cliquez avec le bouton droit sur le projet Subscriber et cliquez sur **Définir comme projet de démarrage**.

La fenêtre **Explorateur de solutions** est présentée à la Figure 12, à la page 30.

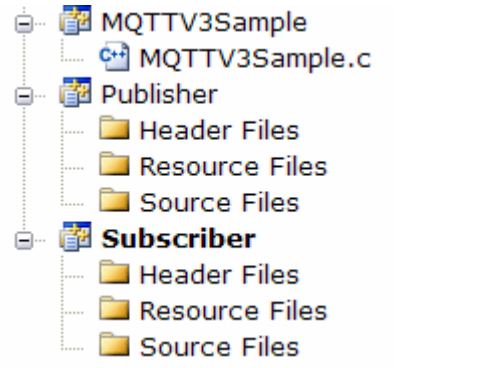


Figure 12. Solution MQTTV3Sample

11. Configurez les pages de propriétés de Subscriber.

a) Cliquez avec le bouton droit de la souris sur **Abonné** dans l'Explorateur de solutions **Propriétés** > **Propriétés de configuration** > **Propriétés** > **Débogage**

Vérifiez que le titre de la fenêtre est bien **Pages de propriétés de Subscriber**.

b) Cliquez sur **Environnement**. Entrez `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` et cliquez sur **Appliquer**.

Modifiez *`sdkroot`* en fonction de votre environnement.

c) Cliquez sur **Commande**, et remplacez `$(TargetPath)` par le chemin du module `MQTTV3Sample`.

Exemple: *`sdkroot`*`\SDK\clients\c\samples\debug\MQTTV3Sample`

Modifiez *`sdkroot`* en fonction de votre environnement.

d) Cliquez sur **Arguments de commande** et entrez `-a subscribe -b localhost -p 1883`, puis cliquez sur **Appliquer**.

12. Configurez les pages de propriétés de Publisher.

Conseil : Vous pouvez passer d'un projet de pages de propriétés à un autre en cliquant sur les projets dans la fenêtre **Explorateur de solutions**.

a) Répétez pour Publisher les mêmes opérations que pour Subscriber.

L'argument de commande est `-b localhost -p 1883`

13. Arrêtez le processus de génération des projets Publisher et Subscriber.
    a) Dans les pages de propriétés de l'un ou de l'autre des projets, cliquez sur **Gestionnaire de configurations** et désélectionnez **Générer** pour les configurations release et de débogage de Publisher et de Subscriber. Cliquez sur **Fermer**.
14. Exécutez le modèle.
    a) Cliquez sur **F5** pour démarrer Subscriber
    b) Cliquez avec le bouton droit sur **Publisher** dans l'explorateur de solutions, puis cliquez sur **Déboguer** > **Démarrer une nouvelle instance**.

Résultats

Le diffuseur de publications (Publisher) et l'abonné (Subscriber) écrivent leur sortie dans leur fenêtre de commande respective. Visual Studio ferme la fenêtre du diffuseur de publications. Regardez la fenêtre de l'abonné, présentée dans la figure qui suit, puis fermez la fenêtre de l'abonné.

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

Figure 13. Sortie de l'abonné

Que faire ensuite

Générez et exécutez le diffuseur de publications et l'abonné asynchrones.Les exemples sont `MQTTV3ASample.c` et `MQTTV3ASSample.c` dans *`sdkroot`*`\SDK\clients\c\samples`.

Création des bibliothèques du client MQTT pour C

Procédez de la manière suivante pour générer les bibliothèques du client MQTT pour C. Cette rubrique contient les options de compilation et de liens pour différentes plateformes, ainsi que des exemples de génération de bibliothèques sous iOS et Windows.

Avant de commencer

1. Ne générez la bibliothèque du client C qu'en cas de nécessité. Si l'une des bibliothèques client préconfigurées du kit de développement de logiciels (SDK), stockées dans le sous-répertoire `SDK\clients\c`, correspond à la plateforme cible, utilisez-la de préférence.
2. Configurez un serveur MQTT pour tester la bibliothèque que vous générez avec le Exemple d'application C du client MQTT. Voir «Initiation aux serveurs MQTT», à la page 138. Vérifiez la configuration du serveur en exécutant l'un de vos modèles d'app client MQTT.
3. Si vous générez une version sécurisée de la bibliothèque C prenant en charge SSL (Secure Sockets Layer), vous devez également générer la bibliothèque OpenSSL. Voir «Génération du package OpenSSL», à la page 45.

   Important : Le téléchargement et la redistribution du package OpenSSL sont soumis à une réglementation stricte en matière d'importation et d'exportation, ainsi qu'à des conditions de licence open source. Lisez soigneusement les restrictions et les avertissements avant de télécharger le package.

Pourquoi et quand exécuter cette tâche

Suivez les instructions de la rubrique «Génération du package OpenSSL», à la page 45 pour télécharger et générer la bibliothèque OpenSSL. Vous devez générer OpenSSL pour pouvoir générer une version sécurisée de la bibliothèque du client MQTT pour C. En revanche, la création d'une version non sécurisée

de la bibliothèque MQTT ne nécessite pas OpenSSL. Les exemples présentés dans les opérations qui suivent concernent la génération de la bibliothèque sous iOS et sous Windows.

Pour générer la bibliothèque du client MQTT pour C, téléchargez les outils de la bibliothèque de développement C et le kit d'outils de développement (SDK) MQTT sur la plateforme de génération. Créez un fichier makefile pour générer une bibliothèque destinée à la plateforme cible, en y incorporant les options décrites à la rubrique «Options de générationMQTT pour différentes plateformes», à la page 32. Les procédures de création et d'exécution d'un fichier makefile par plateforme sont décrites ici :

- iOS «Création des bibliothèques du client MQTT pour C sur Apple Mac destinées aux périphériques iOS», à la page 34
- Windows «Création des bibliothèques MQTT sous Windows», à la page 40

Procédure

1. Installez un environnement de développement C sur la plateforme de génération.

   Les fichiers makefile dans les exemples de cette rubrique ciblent les outils suivants :

   - iOS Pour iOS, sous Apple Mac avec OS X 10.8.2 et les outils de développement iOS du site Xcode.
   - Linux Pour Linux, la version gcc 4.4.6 de Red Hat Enterprise Linux version 6.2.

     Le plus bas niveau pris en charge de la bibliothèque C `glibc` est 2.12, et celui du noyau Linux est 2.6.32.
   - Windows Pour Microsoft Windows, Visual Studio version 10.0.
2. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.

   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.

   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*`\SDK`.
3. Développez le code source des bibliothèques du client MQTT pour C.

   Le fichier compressé du code source est *`sdkroot`*`\SDK\clients\c\source.zip`.
4. Facultatif : Générez OpenSSL.

   Voir «Génération du package OpenSSL», à la page 45.
5. Générez les bibliothèques du client MQTT pour C.

   Les commandes et les options de génération des bibliothèques sont répertoriées à la rubrique «Options de générationMQTT pour différentes plateformes», à la page 32.

   Suivez les procédures décrites dans les exemples suivants pour écrire un fichier makefile destiné à générer les bibliothèques du client MQTT pour C destinées à la plateforme cible.

   - «Création des bibliothèques du client MQTT pour C sur Apple Mac destinées aux périphériques iOS», à la page 34
   - «Création des bibliothèques MQTT sous Windows», à la page 40

Options de générationMQTT pour différentes plateformes

Le tableau suivant contient les options de compilation et de génération des bibliothèques du client MQTT pour C sur différentes plateformes.

Tableau 2. Options de générationMQTT pour différentes plateformes

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

Tableau 2. Options de générationMQTT pour différentes plateformes (suite)

Plateforme	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 *Création des bibliothèques du client MQTT pour C sur Apple Mac destinées aux périphériques iOS*

Suivez la procédure ci-dessous pour écrire un fichier makefile pour générer sur Apple Mac les bibliothèques du client MQTT pour C destinées à être utilisées avec des périphériques iOS.

Avant de commencer

1. Installez les outils de génération, développez et exécutez le fichier makefile sous Apple Mac avec OS X version 10.8.2 ou ultérieure.
2. Installez les outils de ligne de commande pour Xcode, qui incluent le programme **make** . Téléchargez les outils de ligne de commande depuis le site Web Xcode.

Pourquoi et quand exécuter cette tâche

Créez un fichier makefile qui génère les bibliothèques du client MQTT pour C pour l'iPhone ou l'iPad fonctionnant avec un processeur ARMv7 ou ARMv7s, et pour le simulateur d'iPhone fonctionnant avec un processeur i386-64 bits. Voir List of iOS devices.

Conseil : La rubrique «Contenu du fichier makefile MQTTios.mak», à la page 38 contient le texte complet du fichier makefile.

1. Copiez et collez ce texte dans un fichier.
2. Remplacez par une tabulation le premier caractère de chaque ligne qui suit une cible. Voir l'étape «8», à la page 36.
3. Exécutez le fichier à l'aide de la commande indiquée à l'étape «9», à la page 38 de cette procédure.

Procédure

1. Téléchargez et installez les outils de développement iOS .
   a. Connectez-vous avec un ID utilisateur disposant des privilèges d'administration.

b. Votre Apple Mac doit être au moins à la version 10.8.2.

c. Accédez au site Web Xcode pour télécharger Xcode depuis l'App Store Mac.

d. Installez Xcode, l'environnement de ligne de commande et le simulateur.

Si l'App Store Mac propose plusieurs versions du simulateur, choisissez celle qui est compatible avec le niveau d'iOS prévu pour votre application mobile.

2. Créez le fichier makefile `MQTTios.mak`

   Ajoutez un prologue :

```
# La sortie de génération est générée dans le répertoire en cours.
# MQTTCLIENT_DIR doit pointer vers le répertoire de base contenant le code source du client
MQTT.
# MQTTCLIENT_DIR par défaut est le répertoire en cours
# La valeur par défaut de TOOL_DIR est /Applications/Xcode.app/Contents/Developer/Platforms
# OPENSSL_DIR par défaut est sdkroot/openssl, relatif à sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR doit pointer vers le répertoire de base contenant la génération OpenSSL .
# Exemple: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
```

3. Définissez l'emplacement du code source MQTT .

   Exécutez le fichier makefile dans le même répertoire que les fichiers source MQTT ou définissez le paramètre de ligne de commande `MQTTCLIENT_DIR` :

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

   Ajoutez les lignes suivantes au fichier makefile :

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

   L'exemple définit VPATH dans le répertoire où **make** recherche les fichiers source qui ne sont pas explicitement identifiés ; par exemple, tous les fichiers d'en-tête requis dans la génération.

4. Facultatif : Définissez l'emplacement des bibliothèques OpenSSL .

   Cette étape est requise pour générer les versions SSL des bibliothèques du client MQTT pour C.

   Définissez le chemin d'accès par défaut aux bibliothèques OpenSSL sur le même répertoire que celui dans lequel vous avez développé le SDK MQTT . Sinon, définissez le paramètre de ligne de commande `OPENSSL_DIR`.

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

   Conseil : *OpenSSL* est le répertoire OpenSSL qui contient tous les sous-répertoires OpenSSL . Déplacez l'arborescence dans laquelle vous l'avez décompressé si elle contient des répertoires parent vides inutiles.

5. Définissez les répertoires des outils de développement.

   Si vous avez installé Xcode dans un autre répertoire, définissez `TOOL_DIR` sur la ligne de commande.

```
ifndef REP_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. Sélectionnez tous les fichiers source requis pour générer chaque bibliothèque MQTT . Définissez également le nom et l'emplacement de la bibliothèque MQTT à générer.

   Ajoutez la ligne suivante au fichier makefile pour répertorier tous les fichiers source MQTT :

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

Les fichiers source varient selon que vous générez une bibliothèque synchrone ou asynchrone, et que celle-ci inclut ou non SSL.

Ajoutez une ou plusieurs des lignes suivantes, en fonction des cibles à générer. Les bibliothèques partagées sont crées dans le répertoire `darwin_x86_64`.

- Synchrone, non sécurisée :

```
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
```

- Synchrone, sécurisée :

```
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
```

- Asynchrone, non sécurisée :

```
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
```

- Asynchrone, sécurisée :

```
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. Définissez le compilateur et les options de compilation.

   Consultez les options des différentes plateformes présentées dans Options de génération MQTT pour différentes plateformes.

   a) Définissez le projet Gnu C et C++ (**gcc**) en tant que compilateur.

   Sélectionnez trois compilateurs croisés pour générer la bibliothèque pour différents périphériques et le simulateur d'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) Ajoutez les options de compilation.

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

   c) Ajoutez les chemins d'inclusion.

```
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. Définissez les cibles de la génération.

   Conseil : Chaque ligne qui suit la définition de l'implémentation d'une cible doit commencer par un caractère de tabulation.

   a) Définissez la cible **all**.

   La cible "all" génère toutes les bibliothèques.

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

   Définie en premier, elle devient la cible par défaut.

a) Générez la bibliothèque synchrone non sécurisée `libmqttv3c.a`.

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

L'instruction `rm *.o` supprime tous les fichiers objet créés pour chaque bibliothèque. `lipo` concatène les trois bibliothèques en un seul fichier.

b) Générez la bibliothèque asynchrone non sécurisée `libmqttv3a.a`.

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

L'instruction `rm *.o` supprime tous les fichiers objet créés pour chaque bibliothèque. `lipo` concatène les trois bibliothèques en un seul fichier.

c) Générez la bibliothèque synchrone sécurisée `libmqttv3cs.a`.

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

L'instruction `rm *.o` supprime tous les fichiers objet créés pour chaque bibliothèque. `lipo` concatène les trois bibliothèques en un seul fichier.

d) Générez la bibliothèque asynchrone sécurisée `libmqttv3as.a`.

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

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

L'instruction `rm *.o` supprime tous les fichiers objet créés pour chaque bibliothèque. `lipo` concatène les trois bibliothèques en un seul fichier.

e) Définissez la cible **`clean`**.

La cible "clean" retire tous les fichiers et les répertoires générés par le fichier makefile.

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

9. Exécutez le makefile.

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

Résultats

Les fichiers suivants sont créés dans le répertoire *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
```

Contenu du fichier makefile `MQTTios.mak`

```
# La sortie de génération est générée dans le répertoire en cours.
# MQTTCLIENT_DIR doit pointer vers le répertoire de base contenant le code source du client
MQTT.
# MQTTCLIENT_DIR par défaut est le répertoire en cours
# La valeur par défaut de TOOL_DIR est /Applications/Xcode.app/Contents/Developer/Platforms
# OPENSSL_DIR par défaut est sdkroot/openssl, relatif à sdkroot/sdk/clients/c/mqttv3c/src
# OPENSSL_DIR doit pointer vers le répertoire de base contenant la génération OpenSSL .
# Exemple: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef REP_OPENSSL_DIR
  OPENSSL_DIR = ${MQTTCLIENT_DIR}/ ../../../../../openssl-1.0.1c
endif
ALL_SOURCE_FILES = ${wildcard ${MQTTCLIENT_DIR}/*.c}
ifndef REP_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: nettoyage
clean:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *Création des bibliothèques MQTT sous Windows*

Procédez de la manière suivante pour écrire un fichier makefile pour générer les bibliothèques du client MQTT pour C sous Windows.

Avant de commencer

1. Si nécessaire, installez une version de **Make** sur votre poste de travail de génération compatible avec les fichiers makefile écrits pour Gnu make ; sinon, téléchargez Gnu make et générez-la. Voir Gnu Make. Le site Web, Make for Windows, fournit une version installable de **Make** for Windows.
2. Les commandes Linux sont également nécessaires à Windows pour utiliser la cible clean dans l'exemple de fichier makefile. Vous pouvez vous procurer les commandes Linux pour Windows sur les sites Web tels que Cygwin.

Pourquoi et quand exécuter cette tâche

Créez un fichier makefile pour générer les bibliothèques du client MQTT pour C pour Windows 32 bits.

Conseil : La rubrique «Contenu du fichier makefile MQTTwin.mak», à la page 44 contient le texte complet du fichier makefile.

1. Copiez et collez ce texte dans un fichier.
2. Remplacez par une tabulation le premier caractère de chaque ligne qui suit une cible. Voir l'étape «7», à la page 42.
3. Exécutez le fichier à l'aide de la commande indiquée à l'étape «9», à la page 44 de cette procédure.

Procédure

1. Créez le fichier makefile `MQTTwin.mak`

   Ajoutez un prologue :

   ```
   # La sortie de génération est générée dans le répertoire en cours.
   # MQTTCLIENT_DIR doit pointer vers le répertoire de base contenant le code source du client
   MQTT.
   # MQTTCLIENT_DIR par défaut est le répertoire en cours
   # La valeur par défaut de OPENSSL_DIR est sdkroot\openSSL, relative à
   sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR doit pointer vers le répertoire de base contenant la génération OpenSSL .
   # Exemple: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # Définissez l'environnement de génération, par exemple:
   # %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
   # set path= %path% ; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
   ```

2. Définissez l'emplacement du code source MQTT .

   Exécutez le fichier makefile dans le même répertoire que les fichiers source MQTT ou définissez le paramètre de ligne de commande `MQTTCLIENT_DIR` :

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

   Ajoutez les lignes suivantes au fichier makefile :

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

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

L'exemple définit VPATH dans le répertoire où **make** recherche les fichiers source qui ne sont pas explicitement identifiés ; par exemple, tous les fichiers d'en-tête requis dans la génération.

3. Facultatif : Définissez l'emplacement des bibliothèques OpenSSL .

   Cette étape est requise pour générer les versions SSL des bibliothèques du client MQTT pour C.

   Définissez le chemin d'accès par défaut aux bibliothèques OpenSSL sur le même répertoire que celui dans lequel vous avez développé le SDK MQTT . Sinon, définissez le paramètre de ligne de commande `OPENSSL_DIR`.

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

   Conseil : *OpenSSL* est le répertoire OpenSSL qui contient tous les sous-répertoires OpenSSL . Déplacez l'arborescence dans laquelle vous l'avez décompressé si elle contient des répertoires parent vides inutiles.

4. Sélectionnez tous les fichiers source requis pour générer chaque bibliothèque MQTT . Définissez également le nom et l'emplacement de la bibliothèque MQTT à générer.

   Ajoutez la ligne suivante au fichier makefile pour répertorier tous les fichiers source MQTT :

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

   Les fichiers source varient selon que vous générez une bibliothèque synchrone ou asynchrone, et que celle-ci inclut ou non SSL.

   Ajoutez une ou plusieurs des lignes suivantes, en fonction des cibles à générer. Les bibliothèques partagées et les fichiers manifeste sont créés dans le répertoire `windows_ia32`.

   - Synchrone, non sécurisée :

     ```
     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
     ```

   - Synchrone, sécurisée :

     ```
     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
     ```

   - Asynchrone, non sécurisée :

     ```
     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
     ```

   - Asynchrone, sécurisée :

     ```
     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. Définissez le compilateur et les options de compilation.

   Consultez les options des différentes plateformes présentées dans Options de génération MQTT pour différentes plateformes.

   a) Définissez Microsoft Visual C++ comme compilateur.

```
CC = cl
```

b) Ajoutez les options de prétraitement.

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

c) Ajoutez les options de compilation.

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

d) Ajoutez les chemins d'inclusion.

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

e) Facultatif : Ajoutez une option de prétraitement si vous générez une bibliothèque sécurisée.

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

f) Facultatif : Ajoutez les fichiers d'en-tête OpenSSL si vous générez une bibliothèque sécurisée.

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

Conseil : Les fichiers d'en-tête sont dans `${OPENSSL_DIR}/inc32/openssl`, mais le fichier `ssl.h` est contenu dans "`openssl/ssl.h`".

6. Définissez l'éditeur de liens et ses options.

a) Définissez Microsoft Visual C++ comme éditeur de liens.

```
LD = link
```

b) Ajoutez les options d'édition de liens.

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

c) Ajoutez les chemins des bibliothèques.

```
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) Ajoutez les fichiers de sortie intermédiaires.

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

e) Facultatif : Ajoutez les bibliothèques OpenSSL si vous générez une bibliothèque sécurisée.

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

f) Facultatif : Ajoutez le chemin de la bibliothèque OpenSSL si vous générez une bibliothèque sécurisée.

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

7. Définissez les quatre cibles de la génération.

a) Définissez la cible **all**.

Conseil : Chaque ligne qui suit la définition de l'implémentation d'une cible doit commencer par un caractère de tabulation.

La cible "all" génère toutes les bibliothèques.

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

b) Générez la bibliothèque synchrone non sécurisée `mqttv3c.dll`.

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

L'instruction `-rm ${CURDIR}/MQTTAsync.obj` supprime tous les fichiers `MQTTAsync.obj` créés pour une cible antérieure. `MQTTAsync.obj` et `MQTTClient.obj` sont mutuellement exclusifs.

c) Générez la bibliothèque asynchrone non sécurisée `mqttv3a.dll`.

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

L'instruction `-rm ${CURDIR}/MQTTClient.obj` supprime tous les fichiers `MQTTClient.obj` créés pour une cible antérieure. `MQTTAsync.obj` et `MQTTClient.obj` sont mutuellement exclusifs.

d) Générez la bibliothèque synchrone sécurisée `mqttv3cs.dll`.

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

L'instruction `-rm ${CURDIR}/MQTTAsync.obj` supprime tous les fichiers `MQTTAsync.obj` créés pour une cible antérieure. `MQTTAsync.obj` et `MQTTClient.obj` sont mutuellement exclusifs.

e) Générez la bibliothèque asynchrone sécurisée `mqttv3as.dll`.

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

L'instruction `-rm $(CURDIR}/MQTTClient.obj` supprime tous les fichiers `MQTTClient.obj` créés pour une cible antérieure. `MQTTAsync.obj` et `MQTTClient.obj` sont mutuellement exclusifs.

f) Définissez la cible **clean**.

La cible "clean" retire tous les fichiers et les répertoires générés par le fichier makefile.

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

8. Définissez le chemin Windows d'exécution du fichier makefile.

Définissez les éléments en italiques en fonction de votre installation.

a) Définissez l'environnement de Microsoft Visual Studio.

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

b) Définissez la variable `Path` pour inclure le programme make et l'environnement de commande Linux .

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

9. Exécutez le makefile.

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

Conseil : Le caractère de séparation des fichiers doit être une barre oblique, et non une barre oblique inversée.

Résultats

Les fichiers suivants sont créés dans le répertoire *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
```

Contenu du fichier makefile `MQTTwin.mak`

```
# La sortie de génération est générée dans le répertoire en cours.
# MQTTCLIENT_DIR doit pointer vers le répertoire de base contenant le code source du client
MQTT.
# MQTTCLIENT_DIR par défaut est le répertoire en cours
# La valeur par défaut de OPENSSL_DIR est sdkroot\openSSL, relative à
sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR doit pointer vers le répertoire de base contenant la génération OpenSSL .
# Exemple: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Définissez l'environnement de génération, par exemple:
# %comspec% /k "" C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat"" x86
# set path= %path% ; C:\Program Files\GnuWin32\bin;C:\cygwin\bin
ifndef MQTTCLIENT_DIR
  MQTTCLIENT_DIR = ${CURDIR}
endif
VPATH = ${MQTTCLIENT_DIR}
ifndef REP_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: nettoyage
clean:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

Génération du package OpenSSL

Générez le package OpenSSL avant de générer les bibliothèques client MQTT sécurisées pour C, `mqttv3cs` et `mqttv3as`. La génération crée les bibliothèques requises pour créer une version sécurisée de la bibliothèque du client MQTT pour C, et l'outil de gestion des certificats OpenSSL.

Avant de commencer

1. 

La personnalisation du fichier makefile iOS est réservée aux périphériques cible qui fonctionnent sous iOS6. Elle peut varier pour les versions antérieures ou ultérieures d'iOS.
2. 

La personnalisation du fichier makefile Windows est réservée à Windows 32 bits.

Pourquoi et quand exécuter cette tâche

Téléchargez et installez le package OpenSSL et tous les logiciels prérequis. Personnalisez les fichiers makefile OpenSSL, et générez les bibliothèques OpenSSL pour la plateforme cible. Sous Windows et Linux, le programme make génère également l'outil de création et de gestion de clés OpenSSL.

Procédure

1. Installez le package OpenSSL.
   a) Téléchargez le package OpenSSL depuis le site OpenSSL.

      Important : Le téléchargement et la redistribution du package OpenSSL sont soumis à une réglementation stricte en matière d'importation et d'exportation, ainsi qu'à des conditions de licence open source. Lisez soigneusement les restrictions et les avertissements avant de télécharger le package.

   b) Développez le contenu du fichier compressé dans *sdkroot*.

      La page de téléchargement du dernier package est indiquée dans l'onglet **News** du siteOpenSSL. Le package est compressé sous la forme d'un fichier tar avec l'extension `tar.gz`. Lorsqu'il est décompressé, le package crée un premier dossier `openssl`*version* (par exemple *openssl-1.0.1c*). Les exemples font référence au chemin d'accès au dossier en tant que %openssl% sous Windows et $openssl sous iOS; par exemple, sous Windows, %openssl% est *sdkroot*\openssl-1.0.1c.

      Conseil : Vérifiez le chemin créé par la décompression du package OpenSSL. Certains packages ont deux niveaux de dossier `openssl`*version*.

2. Windows

   Facultatif : Sous Windows, téléchargez et installez perl. Voir perl.org.

   Pour cet exemple, perl a été téléchargé depuis le site ActivePerl Downloads.

3. iOS

   Facultatif : Sous iOS, créez trois répertoire supplémentaires.

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

   Pour iOS, vous devez générer le package OpenSSL pour trois plateformes matérielles différentes.

4. Générez le fichier makefile OpenSSL pour créer le package OpenSSL destiné à votre matériel et à votre système d'exploitation.
   a) Ouvrez une fenêtre de commande dans le répertoire %openssl% ou $openssl.
   b) Exécutez la commande perl **Configure** avec les paramètres appropriés.
   - 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

   Sous iOS, personnalisez le fichier makefile OpenSSL généré pour les différents périphériques Apple.
   a) Faites trois copies du fichier makefile $openssl/Makefile généré.

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

b) Modifiez l'instruction "CC=gcc" dans chaque fichier makefile.

L'instruction CC=gcc se trouve environ à la ligne 62. Remplacez-la par les commandes suivantes :

$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) Modifiez l'instruction "CFLAG=..." dans chaque fichier makefile.

L'instruction se trouve environ à la ligne 63 (inscrite ici sur 3 lignes pour une meilleure lisibilité) :

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

L'emplacement des kits de développement de logiciels varie selon les choix que vous avez faits pour votre installation Xcode. La version des kits de développement de logiciels dépend du niveau du système d'exploitation pour lequel vous créez le fichier makefile.

Simulateur d'iPhone

Le fichier makefile du simulateur d'iPhone est $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

Les fichiers makefile iOS sont $openssl/Makefile_arm7 et $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. Exécutez le fichier makefile généré.

- Windows

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

- iOS Sous 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
```

Résultats

La génération crée les bibliothèques partagées, les fichiers lib et les fichiers d'en-tête requis pour générer des versions sécurisées de la bibliothèque du client MQTT pour C.

Initiation au client MQTT pour C sous iOS

Apprenez à mettre en place des applications iOS pour échanger des messages avec un serveur MQTT. Les bibliothèques client MQTT pour C destinées aux appareils iOS (c'est à dire l'iPhone et l'iPad), doivent être générées à partir du kit de développement de logiciels MQTT.

Avant de commencer

1. Lien vers iOS Dev Center et savoir comment développer des applications pour iOS.
2. Procurez-vous un Apple Mac avec OS X 10.8.2 ou une version ultérieure pour exécuter l'environnement de développement intégré (IDE) d'Xcode.
3. (Facultatif) Configurez un environnement de développement C sous Windows ou Linux. Il est conseillé de générer et d'exécuter les modèles d'app C client MQTT sous Windows ou Linux avant de développer une app MQTT iOS. Sinon, vous pouvez simplement étudier le code source sans générer les modèles.
4. Pour connaître les plateformes client pour C MQTT prises en charge et celles de référence, voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.
5. S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .

Pourquoi et quand exécuter cette tâche

Dans cette procédure, vous allez :

1. Apprendre à programmer pour MQTT en étudiant, en générant et en exécutant les modèles d'app clientMQTT et les bibliothèques client MQTT pour C.
2. Installer l'environnement de développement d'Xcode pour iOS sur Apple Mac.
3. Réaliser les tâches nécessaires à la «Création des bibliothèques du client MQTT pour C», à la page 31 MQTT iOS.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 . Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.

   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.

   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*\SDK.
3. Facultatif : Familiarisez-vous avec l'API MQTT en étudiant le Exemple d'application C du client MQTT.

   a) Générez le modèle d'app client MQTT pour C `MQTTV3sample.c` pour Windows ou Linux. Voir «Initiation au client MQTT pour C», à la page 26.

b) Connectez-vous à un serveur MQTT version 3, diffusez des publications et abonnez-vous à des sujets sur le serveur.

c) Etudier le code source et la documentation de l'API MQTT . Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

Apprenez à créer et à reprendre des clients MQTT, à diffuser des publications et à vous abonner aux sujets MQTT en étudiant le modèle synchrone. Le modèle synchrone est plus simple que le modèle asynchrone. Si vous n'avez jamais programmé pour MQTT auparavant, écrivez un programme MQTT synchrone pour vous familiariser avec le modèle de programmation et l'API MQTT.

d) Générez la bibliothèque client MQTTpour C sous Windows ou Linux. Voir «Création des bibliothèques du client MQTT pour C», à la page 31.

e) Générez et exécutez les modèles d'app C client MQTT asynchrone diffuseur de publications et abonné.

f) Etudiez le code source du modèle de client C asynchrone MQTT et la documentation de référence MQTT.

Pour écrire une application MQTT pour un appareil mobile, vous devez utiliser l'interface asynchrone. Les applications bien écrites qui appellent l'interface asynchrone répondent plus rapidement et conservent la durée de vie des batteries mieux que les applications écrites avec l'interface synchrone.

L'interface asynchrone a deux degrés :

i) Le premier degré débloque l'application pendant que la bibliothèque client MQTT attend les publications du serveur.

ii) Le second degré débloque l'application pendant que la bibliothèque client se connecte au serveur, crée des abonnements et diffuse de publications.

4. Téléchargez et installez les outils de développement iOS .

a. Connectez-vous avec un ID utilisateur disposant des privilèges d'administration.

b. Votre Apple Mac doit être au moins à la version 10.8.2.

c. Accédez au site Web Xcode pour télécharger Xcode depuis l'App Store Mac.

d. Installez Xcode, l'environnement de ligne de commande et le simulateur.

Si l'App Store Mac propose plusieurs versions du simulateur, choisissez celle qui est compatible avec le niveau d'iOS prévu pour votre application mobile.

5. Générez les bibliothèques client MQTT pour C sous iOS. Voir «Création des bibliothèques du client MQTT pour C», à la page 31.

Que faire ensuite

1. Vérifiez les bibliothèques client MQTT pour C que vous avez générées :

a. Utilisez l'environnement de développement d'Xcode pour compiler le Exemple d'application C du client MQTT synchrone, et établissez un lien à la bibliothèque client MQTT asynchrone non sécurisée pour C.

b. Utilisez l'environnement de développement d'Xcode pour exécuter le modèle d'app client MQTT asynchrone pour C sur un appareil iOS. Connectez le modèle au serveur MQTT version 3 que vous avez configuré. Voir «Configuration du service MQTT depuis la ligne de commande», à la page 142.

2. Créez une app client C MQTT pour iOS. Les exemples de code de l'application asynchrone C peuvent vous aider. Les exemples sont `MQTTV3ASample.c` et `MQTTV3ASSample.c` dans *sdkroot*`\SDK\clients\c\samples`. Comme exercice, commencez par implémenter le modèle MQTT de publication/abonnement.

Conseil : Pour avoir une idée de ce à quoi peut ressembler l'app et de ce qu'elle peut faire, regardez les captures d'écran du Exemple d'application C du client MQTT. Voir «Initiation au client MQTT pour Java sous Android», à la page 18.

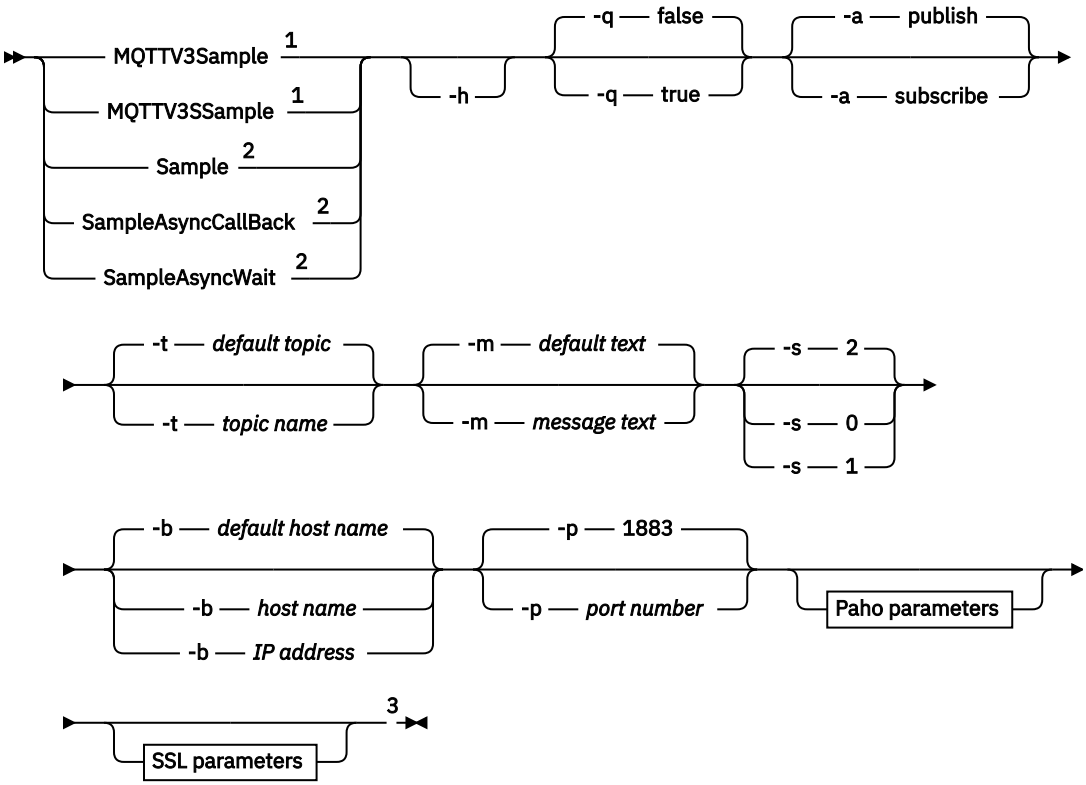
Modèles de programme de ligne de commande MQTT

Syntaxe et paramètres des modèles de programme de ligne de commande MQTT.

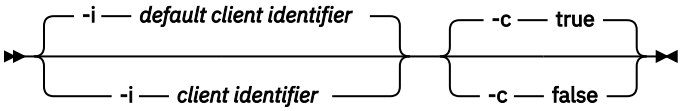
Objet

Publication et abonnement à un sujet.

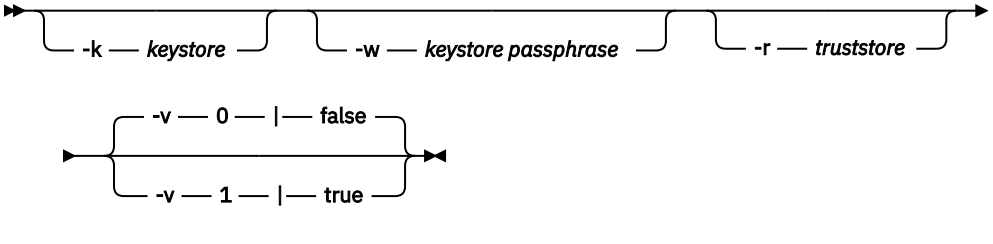
Syntax



Paho parameters



SSL parameters



Remarques :

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

Paramètres

-h
Imprime ce texte d'aide et quitte

-q
Définit le mode silencieux au lieu d'utiliser le mode par défaut false.

-a publish|subscribe
Définit l'action `publish` ou `subscribe`, au lieu d'utiliser l'action par défaut, publication.

-t *nom du sujet*
Publiez ou abonnez-vous à *`topic name`*, au lieu de publier ou de vous abonner à la rubrique par défaut. Les sujets par défaut sont les suivants :

Modèles Paho

Publication
`Sample/Java/v3`

S'abonner
`Sample/#`

Exemples IBM WebSphere MQ

Publication
`MQTTV3Sample/Java/v3` ou `MQTTV3Sample/C/v3`

S'abonner
`MQTTV3Sample/#`

-m *texte du message*
Publiez *`message text`* au lieu d'envoyer le texte par défaut. Le texte par défaut est "`Message from MQTTv3 C client`" ou "`Message from MQTTv3 Java client`"

-s 0|1|2
Définit la qualité de service (QoS) à la place de celle par défaut, 2.

-b *nom d'hôte*
Connectez-vous à *`host name`* ou à l'adresse IP au lieu de vous connecter au nom d'hôte par défaut. Le nom d'hôte par défaut pour les modèles Paho est `m2m.eclipse.org`. Pour les modèles IBM WebSphere MQ, il s'agit de `localhost`.

-p *numéro de port*
Utilisez le port *`port number`* au lieu du port par défaut, 1883.

Paramètres Paho

-i *identificateur client*
Définissez l'identificateur de client sur *`client identifier.`* L'identificateur de client par défaut est `SampleJavaV3_"+action`, où `action` est `publish` ou `subscribe`.

-c true|false
Définit l'attribut de la session de nettoyage. La valeur par défaut est `true` : les abonnements ne sont pas durables.

Paramètres SSL

-k *magasin de clés*
Définissez le chemin d'accès au magasin de clés contenant la clé privée qui identifie le client sur *`keystore`*. Pour les modèles C, le magasin est un fichier PEM (Privacy-Enhanced Mail). Pour les exemples Java, il s'agit d'un magasin de clés Java (JKS).

-w *phrase de passe du magasin de clés*
Définissez la phrase passe pour autoriser le client à accéder au magasin de clés dans *`keystore passphrase`*.

-r ***magasin de clés de confiance***
Définissez le chemin d'accès au magasin de clés contenant les clés publiques des serveurs MQTT auxquelles le client fait confiance dans *truststore*. Le magasin de clés est un fichier PEM (Privacy-Enhanced Mail). Pour les modèles C, le magasin est un fichier PEM (Privacy-Enhanced Mail). Pour les exemples Java, il s'agit d'un magasin de clés Java (JKS).

-v **0|false|1|true**
Définissez la valeur 1|`true` pour l'option de vérification pour exiger un certificat serveur. La valeur par défaut est 0|`false`: le certificat du serveur n'est pas vérifié. Le canal SSL est toujours chiffré.

Définissez l'option sur 0|1 pour les programmes C et `true`|`false` pour les programmes Java .

Tâches associées

«Initiation au client MQTT pour Java», à la page 12
Familiarisez-vous avec les modèles d'application du client MQTT for Java en utilisant IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT. Les modèles d'application utilisent une bibliothèque client du kit de développement de logiciels (SDK) MQTT d'IBM. Le modèle d'application `SampleAsyncCallBack` est un modèle permettant d'écrire des applications MQTT pour Android et d'autres systèmes d'exploitation gérés par des événements.

«Initiation au client MQTT pour C», à la page 26
Familiarisez-vous avec le modèle de client MQTT pour C sur toutes les plateformes sur lesquelles vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle de client MQTT pour C avec IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

«Création des bibliothèques du client MQTT pour C», à la page 31
Procédez de la manière suivante pour générer les bibliothèques du client MQTT pour C. Cette rubrique contient les options de compilation et de liens pour différentes plateformes, ainsi que des exemples de génération de bibliothèques sous iOS et Windows.

Sécurité MQTT

Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. L'identification consiste à nommer le client auquel on donne des droits d'accès. L'authentification cherche à prouver l'identité du client, et l'autorisation consiste à gérer ses droits.

Testez les modèles de sécurité

- «Génération et exécution du Exemple d'application Java client MQTT», à la page 56
- «Connexion du modèle d'application Java du client MQTT sous Android sur SSL», à la page 64
- «Authentification d'une application Java client MQTT avec JAAS», à la page 73
- V 7.5.0.1 «Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
- «Génération et exécution du Exemple d'application C du client MQTT», à la page 86

Identification

Identifiez un client MQTT par son identificateur, par son ID utilisateur, ou par un certificat numérique publique. L'un ou l'autre de ces attributs définit l'identité du client. Un serveur MQTT authentifie le certificat envoyé par le client à l'aide du protocole SSL, ou l'identité du client avec un mot de passe défini par ce dernier. Le serveur contrôle les ressources auxquelles le client peut accéder, en fonction de l'identité du client.

Le serveur MQTT s'identifie auprès du client avec son adresse IP et son certificat numérique. Le client MQTT utilise le protocole SSL pour authentifier le certificat envoyé par le serveur. Dans certains cas, il utilise le nom DNS du serveur pour vérifier que le serveur qui a envoyé le certificat est bien enregistré comme son détenteur.

Définissez l'identité du client de l'une des manières suivantes :

Identificateur client
La classe `MqttClient` (`MQTTClient_create` ou `MQTTAsync_create` en C) définit l'identificateur du client. Appelez le constructeur de classe pour définir l'identificateur du client en tant que paramètre, ou renvoyer un identificateur généré de manière aléatoire.L'identificateur du client doit être unique parmi tous les clients qui se connectent au serveur, et ne doit pas être identique au nom du gestionnaire de files d'attente sur le serveur. Tous les clients doivent avoir un identificateur de client, même s'il n'est pas utilisé pour la vérification d'identité. Voir «Identificateur de client», à la page 129.

ID utilisateur
La classe `MqttClient` (`MQTTClient_create` ou `MQTTAsync_create` en C) définit l'ID utilisateur du client comme un attribut de `MqttConnectOptions` (`MqttClient_ConnectOptions` en C). L'ID utilisateur ne doit pas forcément être unique.

Certificat numérique du client
Le certificat numérique du client est stocké dans son magasin de clés. L'emplacement du magasin de clés varie en fonction du client :

- **Java**

  Définissez l'emplacement et les propriétés du magasin de clés du client en appelant la méthode `setSSLProperties` de `MqttConnectOptions` et en transmettant les propriétés du magasin de clés. Voir SSL - Modification du fichier Example.java. L'outil **keytool** gère les clés et les magasins de clés Java.

- **C**

  `MQTTClient_create` ou `MQTTAsync_create` définit les propriétés du magasin de clés comme attributs de `MQTTClient_SSLOptions ssl_opts`. L'outil **openSSL** crée et gère les clés et les magasins de clés accessibles par le client MQTT pour C.

- **Android**

  Gérez le magasin de clés d'un périphérique Android à l'aide du menu **Paramètres** > **Sécurité**. Chargez les nouveaux certificats à partir de la carte SD.

Définissez l'identité du serveur en stockant sa clé privée dans le magasin de clés du serveur :

IBM WebSphere MQ
Le magasin de clés du serveur MQTT est un attribut du canal de télémétrie auquel le client est connecté.

Définissez l'emplacement et les attributs du magasin de clés à l'aide de IBM WebSphere MQ Explorerou de la commande **DEFINE CHANNEL** ; voir DEFINE CHANNEL (MQTT). Plusieurs canaux peuvent partager un magasin de clés.

Authentification

Un client MQTT peut authentifier le serveur MQTT auquel il se connecte, et le serveur peut authentifier le client qui se connecte à lui.

Un client authentifie un serveur à l'aide du protocole SSL. Un serveur MQTT authentifie un client à l'aide du protocole SSL, d'un mot de passe, ou des deux.

Si le client authentifie le serveur sans que le serveur authentifie le client, le client est généralement reconnu comme un client anonyme. Il est courant d'établir une connexion client anonyme sur SSL, puis d'authentifier le client avec un mot de passe chiffré par la session SSL. Les mots de passe sont beaucoup plus employés que les certificats pour authentifier les clients, à cause des problèmes de diffusion et de gestion des certificats. Les certificats sont généralement utilisés dans appareils à risque, tels que les DAB et les terminaux pour cartes à puce, et dans les dispositifs personnalisés, tels que les compteurs électriques intelligents.

Authentification du serveur par un client

Un client MQTT vérifie qu'il est connecté au bon serveur en authentifiant le certificat du serveur à l'aide du protocole SSL. Vous utilisez cette forme de vérification lorsque vous naviguez dans un site Web avec le protocole HTTPS.

Le serveur envoie au client son certificat public, signé par une autorité de certification. Le client vérifie la signature de l'autorité de certification sur le certificat en utilisant la clé publique de celle-ci. Il vérifie aussi la durée de validité du certificat. Ces contrôles permettent d'établir si le certificat est valide.

Les certificats de l'autorité de certification, souvent appelés certificats racine, sont stockés dans le magasin de clés de confiance du client :

- **Java**

  Appelez la méthode `setSSLProperties` de `MqttConnectOptions` pour transmettre les propriétés du magasin de clés de confiance et définir l'emplacement et les propriété du magasin de clés de confiance du client. Voir SSL - Modification du fichier Example.java. Gérez les certificats et les magasins de clés de confiance avec l'outil **keytool**.

- **C**

  `MQTTClient_create` ou `MQTTAsync_create` définit les propriétés du magasin de clés de confiance comme attributs de `MQTTClient_SSLOptions ssl_opts`. Gérez les certificats et les magasins de clés de confiance avec l'outil **openSSL**.

- **Android**

  Gérez le magasin de clés de confiance d'un périphérique Android à l'aide du menu **Paramètres** > **Sécurité**. Chargez les nouveaux certificats racine à partir de la carte SD.

Authentification du client par un serveur

Un serveur MQTT vérifie qu'il est connecté au bon client en authentifiant le certificat du client à l'aide du protocole SSL, ou en authentifiant son identité avec un mot de passe.

Il authentifie le client avec le mot de passe envoyé par celui-ci au serveur dans un en-tête du MQTT protocol. Avec le mot de passe, le serveur peut choisir d'authentifier l'identificateur du client, son ID utilisateur ou son certificat. Le choix dépend du serveur. Généralement, le serveur authentifie l'ID utilisateur. Vérifiez les mots de passe sur une connexion SSL qui a été sécurisée en vérifiant le serveur, pour éviter d'envoyer des mots de passe en clair.

- **IBM WebSphere MQ**

  IBM WebSphere MQ authentifie le certificat d'un client à l'aide du protocole SSL. Stockez les certificats racine dans le magasin de clés de IBM WebSphere MQ Telemetry. Vous ne pouvez authentifier un certificat client que dans le cadre d'une authentification SSL mutuelle. Autrement dit, vous devez fournir au client le certificat public du serveur, et fournir au serveur le certificat public du client.

  IBM WebSphere MQ Telemetry utilise le même magasin pour son certificat privé et son certificat public, ainsi que pour d'autres certificats publics, tels que les certificats racine fournis par les autorités de certification.

  Définissez l'emplacement et les attributs du magasin de clés à l'aide de IBM WebSphere MQ Explorerou de la commande **DEFINE CHANNEL** ; voir DEFINE CHANNEL (MQTT). Plusieurs canaux peuvent partager un magasin de clés.

  IBM WebSphere MQ authentifie l'ID utilisateur ou l'identificateur du client en appelant le service d'authentification et d'autorisation Java (JAAS).

  Configurez JAAS dans une strophe `MQXRConfig` du fichier `jaas.config`. Le fichier est stocké dans le répertoire `qmgrs\`*`QmgrName`*`\mqxr` dans le chemin de données IBM WebSphere MQ .

  Vérifiez l'authenticité du client en écrivant une méthode `login` pour `JAASLoginModule`. Voir «Configuration JAAS du canal de télémétrie», à la page 116.

IBM WebSphere MQ Telemetry transmet à la méthode `JAASLoginModule.login` les paramètres suivants:

- ID utilisateur
- Mot de passe
- Identificateur de client
- Identificateur réseau
- Nom du canal
- ValidPrompts

Authorization

L'autorisation ne fait pas partie du MQTT protocol. Elle est fournie par les serveurs MQTT. Ce qui est autorisé dépend de ce que fait le serveur. Les serveurs MQTT sont des courtiers de publication/abonnement, et les règles d'autorisation MQTT doivent définir les clients qui sont autorisés à se connecter au serveur, et les sujets auxquels un client peut s'abonner et diffuser des publications. Si le serveur MQTT peut être administré par un client, les règles doivent également définir quels clients peuvent administrer quels aspects du serveur.

Le nombre de clients potentiels est énorme. Il n'est donc pas possible d'autoriser chaque client individuellement. Un serveur MQTT dispose d'un moyen pour rassembler les clients par profil ou par groupe.

L'identité d'un client MQTT, du point de vue des accès et des autorisations, n'est pas unique. L'identité d'un client ne doit pas être confondue avec son identificateur. Il peuvent être identiques, mais sont généralement différents. Par exemple, un nom d'utilisateur peut être commun à plusieurs services, dont certains utilisent une "connexion unique". Couramment, le serveur MQTT d'une entreprise appelle un service d'autorisation qui fourni des identités et des autorisations communes à différentes applications.

IBM WebSphere MQ

IBM WebSphere MQ dispose d'un service d'autorisation connectable. Le service d'autorisation par défaut fourni sousWindows et Linux est le gestionnaire des droits d'accès aux objets (OAM). Voir Contrôle de l'accès aux objets à l'aide de la méthode d'accès aux objets (OAM) sur les systèmes UNIX, Linux et Windows. Il associe les ID utilisateur et les groupes du système d'exploitation, et les opérations sur les objets IBM WebSphere MQ, tels que les sujets et les files d'attente.

Vous pouvez configurer un canal de télémétrie pour accéder à IBM WebSphere MQ avec un ID utilisateur fixe. C'est ainsi que le modèle de canal est configuré. Ou vous pouvez accéder à IBM WebSphere MQ avec l'ID utilisateur défini par le client MQTT. La rubrique Autorisation de l'accès aux objets WebSphere MQ pour les clients MQTT décrit les manières de configurer IBM WebSphere MQ Telemetry pour établir un contrôle d'accès à granularité fine, moyenne ou épaisse.

Tâches associées

«Génération et exécution du Exemple d'application C du client MQTT», à la page 86
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app C sécurisé sur tous les systèmes sur lesquels vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle d'app C sur IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

«Génération et exécution du Exemple d'application Java client MQTT», à la page 56
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app Java sécurisé sous IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible"

«Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

«Connexion du modèle d'application Java du client MQTT sous Android sur SSL», à la page 64
Découvrez et exécutez l'exemple de client Android MQTT connecté à IBM WebSphere MQ via SSL.

«Authentification d'une application Java client MQTT avec JAAS», à la page 73
Cette rubrique explique comment authentifier un client à l'aide de JAAS. Effectuez les étapes de cette tâche pour modifier l'exemple de programme `JAASLoginModule.java` et configurer IBM WebSphere MQ pour authentifier une application Java client MQTT avec JAAS.

Génération et exécution du Exemple d'application Java client MQTT

Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app Java sécurisé sous IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible"

Avant de commencer

1. Vous devez avoir accès à un serveur MQTT version 3.1 qui prend en charge MQTT protocol sur SSL.
2. S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .
3. Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible". Voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.
4. Les canaux SSL doivent être démarrés.

Pourquoi et quand exécuter cette tâche

La présente rubrique illustre la manière de compiler et d'exécuter le Exemple d'application Java client MQTT sécurisé sous Windows à partir de la ligne de commande.

Sécurisez le canal SSL avec des clés signées par une autorité de certification, ou avec des clés autosignées.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 sur SSL. Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Facultatif : Installez un kit de développement Java (JDK) version 7 ou ultérieure.

   version 7 est requis pour exécuter la commande **keytool** afin de certifier les certificats. Si vous n'avez pas besoin de certifier des certificats, la version 7 de JDK n'est pas nécessaire.
3. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.
   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.
   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*\SDK.
4. Créez et exécutez les scripts pour générer des paires de clés et des certificats et configurez IBM WebSphere MQ en tant que serveur MQTT .

   Suivez les étapes décrites dans «Génération de clés et de certificats», à la page 96 pour créer et exécuter les scripts. La rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 58 contient les scripts.
5. Vérifiez que les canaux SSL sont opérationnels et configurés conformément à vos attentes.

   Dans IBM WebSphere MQ, entrez la commande suivante dans une fenêtre de commande :

- 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. Créez les scripts destinés à la génération et à l'exécution du Exemple d'application Java client MQTT sécurisé.
   a) Créez et exécutez `ssjavaclient.bat` pour tester un canal SSL sécurisé avec des certificats autosignés.
   b) Créez et exécutez `cajavaclient.bat` pour tester un canal SSL sécurisé avec des certificats signés par l'autorité de certification.

Scripts d'exécution du client Java MQTT sécurisé

Avant d'exécuter les scripts qui suivent, exécutez ceux de la rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 58.

Client Java MQTT sécurisé avec des certificats autosignés.
Exécutez ce script avec les certificats autosignés que vous avez créés en exécutant le 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
```

Figure 14. `ssjavaclient.bat`

Exécutez le client Java MQTT sécurisé avec des certificats signés par l'autorité de certification.
Exécutez ce script avec les certificats signés par l'autorité de certification que vous avez créés en exécutant le 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
```

Figure 15. `cajavaclient.bat`

Concepts associés

«Sécurité MQTT», à la page 52
Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. L'identification consiste à nommer le client auquel on donne des droits d'accès. L'authentification cherche à prouver l'identité du client, et l'autorisation consiste à gérer ses droits.

Tâches associées

«Génération de clés et de certificats», à la page 96
La procédure qui suit permet de générer des clés et des certificats pour les clients Java et C, y compris les applications Android et iOS, et les serveurs IBM WebSphere MQ et IBM MessageSight.

«Connexion du modèle d'application Java du client MQTT sous Android sur SSL», à la page 64
Découvrez et exécutez l'exemple de client Android MQTT connecté à IBM WebSphere MQ via SSL.

«Authentification d'une application Java client MQTT avec JAAS», à la page 73
Cette rubrique explique comment authentifier un client à l'aide de JAAS. Effectuez les étapes de cette tâche pour modifier l'exemple de programme `JAASLoginModule.java` et configurer IBM WebSphere MQ pour authentifier une application Java client MQTT avec JAAS.

Exemples de script permettant de configurer des certificats SSL pour Windows

** **

Les exemples de fichier de commandes créent les certificats et les magasins de certificats, selon la procédure décrite dans cette tâche. En outre, l'exemple configure l'utilisation du magasin de certificats du serveur dans le gestionnaire de files d'attente du client MQTT. Il supprime et recrée le gestionnaire de files d'attente en appelant le script `SampleMQM.bat` fourni avec IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` définit les noms et les chemins des certificats et des autres paramètres nécessaires aux commandes **keytool** et **openSSL**. Les paramètres sont décrits dans les commentaires du script.

```
@echo off
@rem Set the path where you installed the MQTT SDK
@rem and short cuts to the samples directories.
set SDKRoot=C:\MQTT
```

```
set jsamppath=%SDKRoot%\sdk\clients\java\samples
set csamppath=%SDKRoot%\sdk\clients\c\samples
```

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

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

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

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

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

```
set cltcertcasigned=%certpath%\cltcacertsigned.cer
set cltcertselfsigned=%certpath%\cltcertselfsigned.cer
```

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

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

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

cleancert.bat

Les commandes du script `cleancert.bat` suppriment le gestionnaire de files d'attente du client MQTT qui pourraient éventuellement verrouiller le magasin de certificats, puis tous les magasins de clés et les certificats créés par les exemples de script de sécurité.

```
@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

Les commandes du script `genkeys.bat` créent les paires de clés destinées à votre autorité de certification privée, au serveur et à un client.

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

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

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

sscerts.bat

Les commandes du script `sscerts.bat` exportent les certificats autosignés du client et du serveur de leurs magasins de clés, et importent le certificat du serveur dans le magasin de clés de confiance du client, et le certificat du client dans le magasin de clés du serveur. Le serveur n'a pas de magasin de clés de confiance. Les commandes créent un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

```
@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

Le script importe le certificat racine de l'autorité de certification dans les magasins de clés privés. Ce certificat est nécessaire pour créer la chaîne de certificats contenant le certificat racine et le certificat signé. Le script `cacerts.bat` exporte les demandes de certificat client et serveur de leurs magasins de clés respectifs. Le script signe les demandes de certificat avec la clé de l'autorité de certification privée dans le magasin de clés `cajkskeystore.jks`, puis réimporte les certificats dans les magasins de clés d'où provenaient les demandes. L'importation crée la chaîne de certificats avec le certificat racine de l'autorité de certification. Le script crée un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

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

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

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

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

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

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

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

mqcerts.bat

Le script répertorie les magasins de clés et les certificats qui se trouvent dans le répertoire des certificats. Il crée ensuite le gestionnaire de files d'attente de l'exemple MQTT et configure les canaux de télémétrie sécurisés.

```
@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
```

Connexion du modèle d'application Java du client MQTT sous Android sur SSL

Découvrez et exécutez l'exemple de client Android MQTT connecté à IBM WebSphere MQ via SSL.

Avant de commencer

Dans cet article, on considère que vous utilisez au minimum l'API Android niveau 14 (ICS 4.0). Les niveaux inférieurs disposent d'un magasin de clés, mais seules les applications système peuvent y accéder.

1. Vous devez avoir accès à un serveur MQTT version 3.1 qui prend en charge MQTT protocol sur SSL.
2. S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .
3. Si vous testez la connexion sur un appareil Android plus ancien, vous aurez peut-être besoin d'une carte SD pour transférer le certificat sur l'appareil.
4. Si vous testez la connexion sur un périphérique virtuel Android, configurez une carte SD pour celui-ci.
5. Les canaux SSL doivent être démarrés.

Pourquoi et quand exécuter cette tâche

Complétez cette tâche pour exécuter le Exemple d'application Java de client MQTT pour Android sur SSL. Une connexion SSL établit un canal chiffré sécurisé entre votre appareil Android et le serveur MQTT. L'identité du serveur est authentifiée.

Avec Android, vous pouvez authentifier le serveur avec SSL. Vous pouvez aussi authentifier l'appareil, bien que le modèle d'application ne prenne pas en charge cette possibilité. Pour authentifier l'appareil, utilisez l'API KeyChain ou JAAS afin d'authentifier l'identificateur client, l'adresse IP du client ou le nom d'utilisateur et le mot de passe fournis par l'application MQTT Android.

Tous les certificats X.509 que vous installez dans le magasin de clés de confiance Android doivent être signés par une autorité de certification. Dans le modèle, vous créez une autorité de certification qui signe le certificat installé par vous dans l'appareil Android. De nombreux certificats racine sont préinstallés dans les appareils Android.

Vous devez créer un verrou dans votre périphérique Android avant d'y installer un certificat digne de confiance. Le verrou empêche l'ajout de certificats à votre insu.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 sur SSL. Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Exécutez le modèle d'app client for Android MQTT "`MQTTExerciser`" sur un canal MQTT non sécurisé. Voir «Initiation au client MQTT pour Java sous Android», à la page 18.

   Vous utilisez à nouveau l'application pour tester le canal sécurisé.

   Si vous avez démarré un périphérique virtuel Android, laissez-le fonctionner.

3. Facultatif : Installez un kit de développement Java (JDK) version 7 ou ultérieure.

   version 7 est requis pour exécuter la commande **keytool** afin de certifier les certificats. Si vous n'avez pas besoin de certifier des certificats, la version 7 de JDK n'est pas nécessaire.
4. Créez et exécutez les scripts pour générer des paires de clés et des certificats et configurez IBM WebSphere MQ en tant que serveur MQTT .

   Suivez les étapes décrites dans «Génération de clés et de certificats», à la page 96 pour créer et exécuter les scripts. La rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 68 contient les scripts.

   Vous avez besoin du certificat public de l'autorité de certification et du magasin de clés du serveur. Vous n'avez pas besoin de certificats client, ni de certificats au format `.pem` ou `.p12`.
5. Vérifiez que les canaux SSL sont opérationnels et configurés conformément à vos attentes.

   Dans IBM WebSphere MQ, entrez la commande suivante dans une fenêtre de commande :

   - 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. Installez le certificat de l'autorité de certification dans le magasin de clés de confiance Android.

   Dans le modèle, le fichier de l'autorité de certification est `cacert.cer`.

   a) Renommez le certificat `cacert.crt`.

   b) Copiez le certificat dans la racine de la mémoire interne ou sur la carte SD.

      Pour un périphérique virtuel actif, ouvrez Eclipse ou copiez le certificat sur le périphérique virtuel à l'aide d'Android Debug Bridge (ADB) :

      Eclipse

      i) Exécutez Eclipse, et ouvrez la perspective DDMS.

      ii) Dans la vue principale, ouvrez l'**explorateur de fichiers**.

      iii) Ouvrez le répertoire `mnt/sdcard`.

      iv) Faites glisser le fichier `cacert.crt` dans le répertoire `mnt/sdcard`.

      ADB

      i) Ouvrez une fenêtre de commande et définissez son répertoire de travail sur `android-sdk\platform-tools` dans le répertoire d'installation android ; par exemple, `C:\Program Files\Android\android-sdk\platform-tools`.

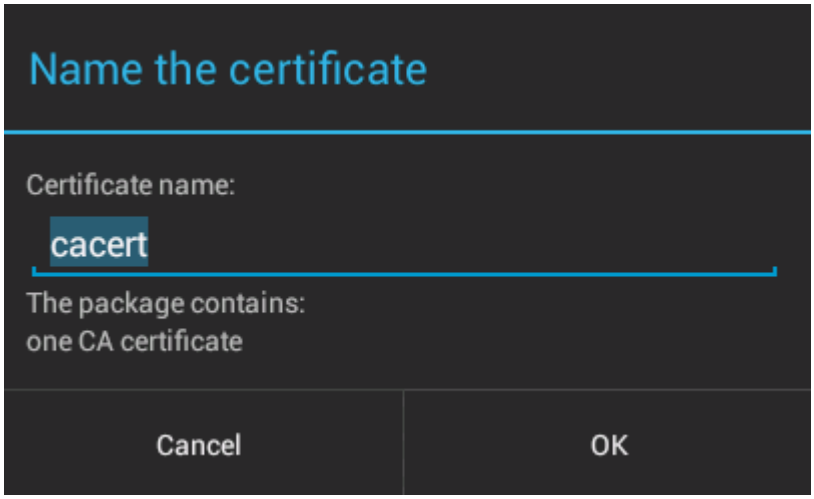
      ii) Copiez le certificat dans le répertoire `mnt/sdcard` :

      ```
      adb push %cacert% /mnt/sdcard/cacert.crt
      ```
7. Installez le certificat dans le magasin de clés de confiance de l'appareil Android.

   Le certificat doit contenir une clause Basic Constraints avec la valeur `Subject Type=CA`.

   a) Déverrouillez l'appareil et cliquez sur le bouton **Widgets**.

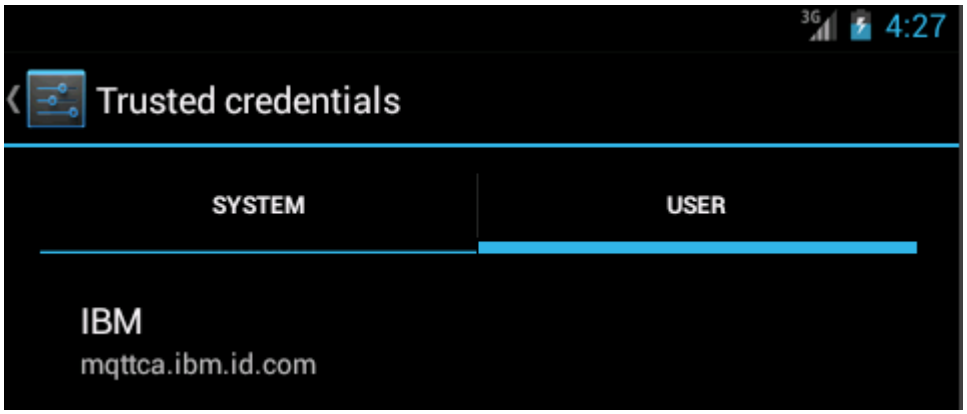
   b) Cliquez sur **Paramètres** > **Sécurité** > **Stockage des données d'identification** > **Installer à partir de la carte SD**.



c) Vérifiez le nom du fichier de certificat et cliquez sur **OK**.

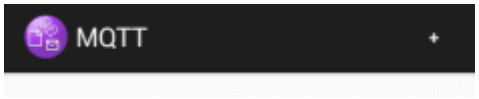
Remarque : Si vous n'avez pas défini de verrou sur l'appareil, Android vous invite à le faire maintenant.

8. Vérifiez que le certificat est installé sur l'appareil.
   a) Cliquez sur **Données d'identification sécurisées** > **Utilisateur** et attendez quelques minutes que votre certificat s'affiche dans la liste des certificats d'utilisateur.

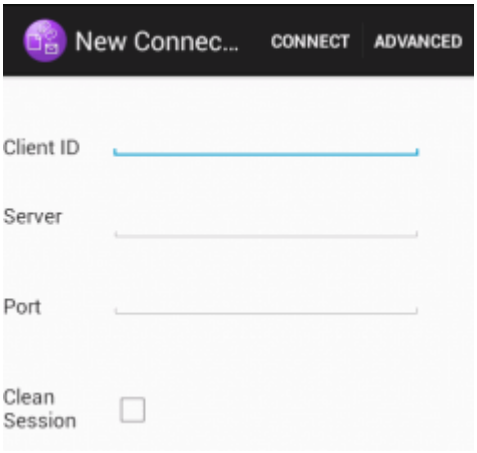


9. Exécutez à nouveau l'app `MQTTExerciser`, et établissez une connexion à un canal MQTT que vous avez configuré pour les clients SSL anonymes.
   a) Ouvrez le Exemple d'application Java de client MQTT pour Android.

   Cette fenêtre est ouverte sur votre périphérique Android :



   b) Connectez-vous à un serveur MQTT .
      i) Cliquez sur le symbole **+** pour ouvrir une nouvelle connexion à MQTT.



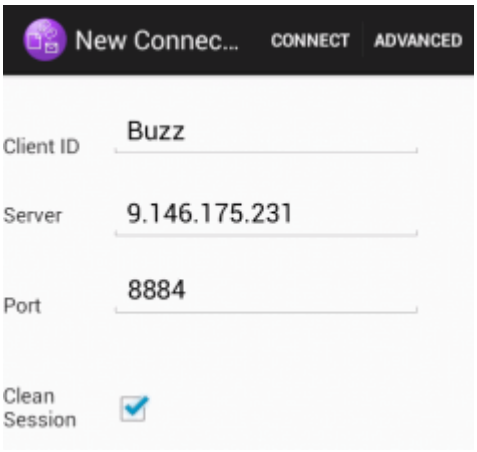
ii) Entrez un identifiant unique de votre choix dans la zone **ID client**. Soyez patient, car la saisie peut être lente.

iii) Dans la zone **Serveur**, entrez l'adresse IP de votre serveur MQTT.

Il s'agit du serveur que vous avez sélectionné à la première étape principale. L'adresse IP ne doit pas être `127.0.0.1`.

iv) Entrez le numéro de port de la connexion MQTT.

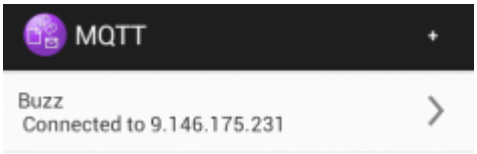
Définissez le numéro de port sur 8884, défini par la variable `%sslportopt%` dans les modèles de script. Ce numéro de port est celui du canal MQTT que vous avez configuré pour les clients SSL anonymes en exécutant les modèles de script de l'étape «4», à la page 65.



v) Cliquez sur l'onglet **Advanced**, et sélectionnez l'option **SSL**. Cliquer sur **Sauvegarder**.

vi) Cliquez sur **Connect**.

Si la connexion est établie, le message "Connexion" apparaît, suivi de cette fenêtre :



Résultats

L'app `MQTTExerciser` se comporte comme sur une connexion non sécurisée, sauf qu'elle est un peu plus longue à se connecter et à échanger des messages.

Concepts associés

«Sécurité MQTT», à la page 52

Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. L'identification consiste à nommer le client auquel on donne des droits d'accès. L'authentification cherche à prouver l'identité du client, et l'autorisation consiste à gérer ses droits.

Tâches associées

«Génération de clés et de certificats», à la page 96
La procédure qui suit permet de générer des clés et des certificats pour les clients Java et C, y compris les applications Android et iOS, et les serveurs IBM WebSphere MQ et IBM MessageSight.

«Génération et exécution du Exemple d'application Java client MQTT», à la page 56
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app Java sécurisé sous IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible"

«Authentification d'une application Java client MQTT avec JAAS», à la page 73
Cette rubrique explique comment authentifier un client à l'aide de JAAS. Effectuez les étapes de cette tâche pour modifier l'exemple de programme `JAASLoginModule.java` et configurer IBM WebSphere MQ pour authentifier une application Java client MQTT avec JAAS.

Exemples de script permettant de configurer des certificats SSL pour Windows

** **

Les exemples de fichier de commandes créent les certificats et les magasins de certificats, selon la procédure décrite dans cette tâche. En outre, l'exemple configure l'utilisation du magasin de certificats du serveur dans le gestionnaire de files d'attente du client MQTT. Il supprime et recrée le gestionnaire de files d'attente en appelant le script `SampleMQM.bat` fourni avec IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` définit les noms et les chemins des certificats et des autres paramètres nécessaires aux commandes **keytool** et **openSSL**. Les paramètres sont décrits dans les commentaires du script.

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

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

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

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

```
set cakeypass=%password%
@rem ca key store
set cajkskeystore=%certpath%\cakeystore.jks
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

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

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

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

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

```
set cltcapemtruststorepass=%password%
set cltsrvp12truststore=%certpath%\cltsrvtruststore.p12
set cltsrvp12truststorepass=%password%
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

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

cleancert.bat

Les commandes du script `cleancert.bat` suppriment le gestionnaire de files d'attente du client MQTT qui pourraient éventuellement verrouiller le magasin de certificats, puis tous les magasins de clés et les certificats créés par les exemples de script de sécurité.

```
@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

Les commandes du script `genkeys.bat` créent les paires de clés destinées à votre autorité de certification privée, au serveur et à un client.

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

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

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

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

sscerts.bat

Les commandes du script `sscerts.bat` exportent les certificats autosignés du client et du serveur de leurs magasins de clés, et importent le certificat du serveur dans le magasin de clés de confiance du client, et le certificat du client dans le magasin de clés du serveur. Le serveur n'a pas de magasin de clés de confiance. Les commandes créent un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

```
@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

Le script importe le certificat racine de l'autorité de certification dans les magasins de clés privés. Ce certificat est nécessaire pour créer la chaîne de certificats contenant le certificat racine et le

certificat signé. Le script `cacerts.bat` exporte les demandes de certificat client et serveur de leurs magasins de clés respectifs. Le script signe les demandes de certificat avec la clé de l'autorité de certification privée dans le magasin de clés `cajkskeystore.jks`, puis réimporte les certificats dans les magasins de clés d'où provenaient les demandes. L'importation crée la chaîne de certificats avec le certificat racine de l'autorité de certification. Le script crée un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

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

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

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

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

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

```
@rem Omit this step, unless you are authenticating clients.
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

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

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

mqcerts.bat

Le script répertorie les magasins de clés et les certificats qui se trouvent dans le répertoire des certificats. Il crée ensuite le gestionnaire de files d'attente de l'exemple MQTT et configure les canaux de télémétrie sécurisés.

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

```
@rem
@echo Create queue manager and define mqtt channels and certificate stores
call "%MQ_FILE_PATH%\mqxr\Samples\SampleMQM" >> %mqlog%
echo DEFINE CHANNEL(%chlreq%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreq%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlopt%)      CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslopt%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) | runmqsc %qm% >> %mqlog%
V7.5.0.1
echo DEFINE CHANNEL(%chlsslreqws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portsslreqws%)
SSLCAUTH(%authreq%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlssloptws%) CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portssloptws%)
SSLCAUTH(%authopt%) SSLKEYR('%srvjkskeystore%') SSLKEYP('%srvjkskeystorepass%')
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
echo DEFINE CHANNEL(%chlws%)       CHLTYPE(MQTT) TRPTYPE(TCP) PORT(%portws%)
MCAUSER(%mcauser%) PROTOCOL(HTTP) | runmqsc %qm% >> %mqlog%
@echo MQ logs saved in %mqlog%echo
```

Authentification d'une application Java client MQTT avec JAAS

Cette rubrique explique comment authentifier un client à l'aide de JAAS. Effectuez les étapes de cette tâche pour modifier l'exemple de programme JAASLoginModule.java et configurer IBM WebSphere MQ pour authentifier une application Java client MQTT avec JAAS.

Avant de commencer

1. Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible". Voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M.
2. S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .

3. Vous devez avoir accès aux exemples MQXR JAASLoginModule et JAASPrincipal Java dans une installation IBM WebSphere MQ . Ces exemples se trouvent dans le répertoire %MQ_FILE_PATH% \mqxr\samples.
4. Effectuez la procédure sous Windows ou Linux. Les modèles correspondent à Windows.
5. Pour réaliser l'étape «1», à la page 74, vous devez être autorisé à créer le gestionnaire de files d'attente MQXR_SAMPLE_QM sur IBM WebSphere MQ.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous avez généré les paramètres d'identification du client MQTT Sample à partir de votre version de JAASLoginModule. L'écriture des paramètres client implique la modification de l'exemple de programme JAASLoginModule et la configuration de IBM WebSphere MQ pour charger votre version de JAASLoginModule.

Procédure

1. Effectuez les opérations décrites à la rubrique «Compilation et exécution de tous les modèles d'app client Java MQTT depuis Eclipse», à la page 16 pour exécuter le client Paho MQTT Sample.

   Votre objectif consiste à préparer un environnement au développement eu au test de l'authentification JAAS. La personnalisation du module d'authentification JAAS nécessite un environnement de développement Java. Dans le modèle, vous testez la configuration JAAS en exécutant le modèle de client Paho pour Java. Dans un souci de simplicité, utilisez le même environnement de développement pour modifier le modèle de client et le modèle de module de connexion JAAS. Vous pouvez également tester le module de connexion JAAS avec le client MQTT pour C, ou avec un autre client MQTT de votre choix.

2. Facultatif : Ajoutez des paramètres de nom d'utilisateur et de mot de passe au modèle Paho MQTT.

   Remarque : Si le client Paho pour Java contient les paramètres de nom d'utilisateur et de mot de passe, cette étape est superflue. Recherchez d'éventuelles mises à jour sur le site de téléchargement. Voir la sélection des correctifs, sinon, remplacez votre copie de Sample.java.

   a) Ouvrez l'explorateur de fichiers dans le package org.eclipse.paho.sample.mqttv3app du projet contenant les modèles Paho.

   b) Cliquez avec le bouton droit sur Sample.java **Copy** > **Paste**. Dans la fenêtre **Name Conflict**, entrez le nom SampleForJAAS.

   c) Ajoutez les lignes de code suivantes à la méthode main.

      i) Après la ligne "boolean ssl = false;", déclarez les variables userName et password :

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

      Pour garantir la compatibilité avec les versions plus anciennes des serveurs MQTT, par défaut, ne définissez pas de paramètres de nom d'utilisateur et de mot de passe.

      ii) Après la ligne, "case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;", analysez les deux nouveaux paramètres en entrée :

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

      iii) Avant la ligne, "if (action.equals("publish")) {", ajoutez userName et password aux arguments de constructeur Sample :

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

   d) Ajoutez userName et password au constructeur de Sample.

      Remplacez :

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

A:

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

e) Ajoutez les lignes de code suivantes au constructeur `Sample`.

Après la ligne "`conOpt.setCleanSession(clean);`", définissez les variables `userName` et `password` dans l'objet `conOpt` de la méthode `Sample` :

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

f) Dans les méthodes `publish` et `subscribe`, modifiez les lignes de code suivantes :

Remplacez la ligne "`client.connect();`" par

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

3. Créez un projet Java, `JAASSample`, pour votre modèle JAAS.
   a) Dans l'espace de travail Eclipse, ouvrez l'assistant **Nouveau projet Java** : cliquez sur **Fichier** > **Nouveau** > **Projet Java**.
   b) Dans la zone **Nom du projet**, entrez `JAASSample`.
   c) Dans les options du JRE, sélectionnez `J2SE-1.5` comme environnement d'exécution Java et cliquez sur **Suivant**.

      Le JRE doit correspondre à celui qui est exécuté par votre serveur IBM WebSphere MQ. IBM WebSphere MQ Version 7.5 exécute `J2SE-1.5`.
   d) Dans la fenêtre **Paramètres Java**, cliquez sur l'onglet **Bibliothèques** et cliquez sur **Ajouter des fichiers JAR externes...**. Accédez à la`%MQ_FILE_PATH%\mqxr\lib` et sélectionnez **MQXR.jar**; cliquez sur **Terminer**.
4. Importez les classes JAAS samples `JAASLoginModule` et `JAASPrincipal` .
   a) Cliquez avec le bouton droit de la souris sur le projet `JAASSample` dans l'explorateur de packages **Importer ...** > **Général** > **Système de fichiers** et cliquez sur **Suivant**.
   b) Naviguez jusqu'à `%MQ_FILE_PATH%\mqxr\samples` et cochez `JAASLoginModule.java` et `JAASPrincipal.java`. Cliquez sur **Finish**.
   c) Sélectionnez et cliquez avec le bouton droit de la souris sur les deux fichiers Java dans l'explorateur de packages, **Restructurer ...** > **Déplacer**.
   d) Dans la fenêtre **Déplacement**, vérifiez que `JAASSample` est bien la destination des deux éléments, et cliquez sur **Créer un package...**.
   e) Entrez `samples` dans la zone **Nom** de l'assistant **Nouveau package Java**. Cliquez sur **Terminer** > **OK**.

   Eclipse construit les classes Java importées et génère plusieurs avertissements relatifs à des valeurs non utilisées.
5. Renommez la classe `JAASLoginModule`.

   Renommez la classe pour qu'elle soit aisément distinguable du modèle de classe `JAASLoginModule` fourni avec IBM WebSphere MQ.
   a) Cliquez avec le bouton droit de la souris sur `JAASloginModule.java` dans l'explorateur de packages **Restructurer ...** > **Renommer**.

b) Dans la fenêtre **Modification du nom de l'unité de génération**, remplacez le contenu de la zone **Nouveau nom**, JAASloginModule, par MyJAASloginModule. Cliquez sur **Terminer**.

6. Modifiez la classe MyJAASloginModule pour créer le contenu des zones de rappel.

   a) Ajoutez les lignes de code suivantes à MyJAASloginModule.java.

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

   Placez les lignes juste avant l'instruction:"if (true) loggedIn = true;"

   b) Appuyez sur CTRL+Shift+O pour réorganiser les importations et sauvegarder le fichier.

7. Renommez JAASPrincipal en MyJAASPrincipal.

   Renommez la classe pour éviter toute confusion avec la classe JAASPrincipal. Dans le modèle, ne modifiez pas le contenu de la classe MyJAASPrincipal.

8. Attribuez à l'ID utilisateur qui exécute les processus de gestionnaire de files d'attente des droits read et execute sur vos classes JAAS .

   a) Dans l'explorateur Windows, ouvrez le répertoire de l'espace de travail Eclipse. Dans l'exemple, l'emplacement de l'espace de travail Eclipse est représenté par la variable Eclipse , *workspace_loc*.

   b) Naviguez jusqu'au répertoire qui contient les classes MyJAASLoginModule et MyJAASPrincipal.

   Le chemin de répertoire est *workspace_loc*\JAASSample\bin\samples

   c) Sélectionnez et cliquez avec le bouton droit sur les deux classes, puis cliquez sur **Propriétés**. Cliquez sur l'onglet **Sécurité** de la fenêtre **Propriétés**.

   d) Cliquez sur **Ajouter ...**, entrez le nom d'objet mqm, puis cliquez sur **Vérifier les noms** pour le vérifier ; cliquez sur **OK**.

   e) Sélectionnez **mqm** dans la liste **Noms d'utilisateurs ou de groupes** et cochez **Lecture et exécution** et **Lecture** dans la liste des droits sur mqm. Cliquez sur OK.

9. Configurez IBM WebSphere MQ pour exécuter votre classe MyJAASLoginModule .

   a) Ajoutez un fichier service.env à la configuration de IBM WebSphere MQ pour définir les chemins d'accès aux classes permettant de charger la classe MyJAASLoginModule.

   Créez un fichier service.env avec l'instruction de chemin d'accès aux classes suivante dans votre répertoire *WMQ_DATA_PATH* :

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

   Où *user.dir* est la racine du répertoire des fichiers de classe compilés dans votre espace de travailEclipse . Le répertoire *WMQ_DATA_PATH* contient le répertoire qmgrs . Voir Autres variables d'environnement.

   Conseil : CLASSPATH=*user.dir*\JAASSample\bin peut être la seule instruction du fichier service.env

   b) Ajoutez une strophe, MyJAASStanza, pour permettre au fichier jaas.config d'identifier la classe MyJAASLoginModule par rapport aux chemins d'accès aux classes du fichier service.env.

   jaas.config se trouve dans le répertoire mqxr du gestionnaire de files d'attente ; *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.

   La strophe est la suivante :

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

c) Configurez un canal IBM WebSphere MQ Telemetry avec le nom de votre strophe de configuration JAAS.

Exécutez la commande suivante depuis une fenêtre de commande :

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

La commande lie le canal MyJAAS à la strophe MyJAASStanza dans le fichier jaas.config. Si vous ne définissez pas l'MCAUSER, ou si vous définissez l'option USECLTID dans la définition du canal, celui-ci autorise l'accès aux ressources du gestionnaire de files d'attente avec le nom d'utilisateur fourni par le programme du client MQTT. Dans le modèle, le nom d'utilisateur fourni par le client est "Guest". Les autorisations existantes pour Guest, définies par le fichier de commandes SampleMQM, sont aussi utilisées dans l'exemple.

10. Redémarrez le service IBM WebSphere MQ Telemetry pour qu'il lise les nouvelles données de configuration.
Pour redémarrer le service IBM WebSphere MQ Telemetry, démarrez le gestionnaire de files d'attente ou le service depuis IBM WebSphere MQ Explorer ou, pour l'exemple de configuration, exécutez les commandes suivantes :

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

11. Exécutez le programme Sample.

Pour définir une configuration d'exécution pour SampleForJAAS, suivez la même procédure qu'à l'étape «1», à la page 74, avec les modifications suivantes :

a) Définissez le port numéro 1890, qui correspond à la configuration du canal MQTT.

b) Ajoutez les paramètres -u Guest -z password aux mots de passe dans l'onglet **(x)= Arguments** pour les configurations d'abonné et de diffuseur de publications que vous avez créées pour le programme Sample.

Les exemples de programme produisent le même résultat, à l'exception du numéro de port qui est maintenant 1890 au lieu de 1883.

Dans le répertoire *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM , ouvrez le fichier mqxr.stdout . Le résultat de MyJAASLoginModule est inscrit dans mqxr.stdout :

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

Que faire ensuite

Si l'exemple ne fonctionne pas, consultez la rubrique relative au traitement des incidents liés à «Résolution des problèmes : Le module de connexion JAAS n'est pas appelé par le service de télémétrie», à la page 188, et essayez les astuces suivantes pour le déboguer.

1. Ajoutez -verbose aux paramètres dans *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties. Ce journal vous permet de voir si le chargement des classes a abouti.

   La sortie est écrite dans *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr.
2. Recherchez dans *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log les exceptions qui sont émises dans MyJAASLoginModule. Par exemple, une exception est émise si vous tentez de créer un mot de passe (password) null, qui est une matrice de caractères et non une chaîne.
3. Regardez dans *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log. Si le nom d'utilisateur dans userName n'est pas autorisé à accéder aux ressources du gestionnaire de files

d'attente, et si le canal est configuré sans l'option MCAUSER ou USECLTID, les erreurs sont signalées dans ce journal.

4. Vérifiez que le nom de la section dans *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config est identique à celui du canal MQTT configuré pour le port auquel le client Sample tente de se connecter.
5. Vérifiez que le chemin de la strophe correspond à celui de la classe MyJAASLoginModule dans Eclipse. Par exemple :

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

6. Pour éviter que l'erreur se trouve dans le chemin d'accès aux classes du fichier service.env dans *WMQ_DATA_PATH* ne soit pas correctement sélectionnée, modifiez la ligne "set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%" dans %MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT pour inclure votre chemin d'accès aux classes. Vous pouvez aussi renvoyer le chemin d'accès aux classes en utilisant la commande echo. Cependant, le chemin d'accès aux classes ne contient pas celui qui est défini dans service.env. Ceci ne fonctionne donc que si vous modifiez le fichier controlMQXR.BAT.

Concepts associés

«Sécurité MQTT», à la page 52
Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. L'identification consiste à nommer le client auquel on donne des droits d'accès. L'authentification cherche à prouver l'identité du client, et l'autorisation consiste à gérer ses droits.

«Configuration JAAS du canal de télémétrie», à la page 116
Configurez JAAS pour authentifier le Nom d'utilisateur envoyé par le client.

Tâches associées

«Résolution des problèmes : Le module de connexion JAAS n'est pas appelé par le service de télémétrie», à la page 188
Vérifiez si le module de connexion JAAS n'est pas appelé par le service de télémétrie (MQXR), et configurez JAAS pour corriger le problème.

«Génération et exécution du Exemple d'application Java client MQTT», à la page 56
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app Java sécurisé sous IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible"

«Connexion du modèle d'application Java du client MQTT sous Android sur SSL», à la page 64
Découvrez et exécutez l'exemple de client Android MQTT connecté à IBM WebSphere MQ via SSL.

Information associée

Autres variables d'environnement

Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets

Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

Avant de commencer

1. Vous devez avoir accès à un serveur MQTT version 3 qui prend en charge le MQTT protocol sur WebSockets.
2. Le navigateur doit prendre en charge SSL et le WebSocket protocol. Voir «Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL», à la page 176.

3. Les canaux SSL doivent être démarrés.

Pourquoi et quand exécuter cette tâche

Complétez cette tâche pour exécuter les modèles de page du client de messagerie MQTT pour JavaScript sur SSL. La tâche vous dirige vers «Génération de clés et de certificats», à la page 96 pour créer les certificats et configurer IBM WebSphere MQ.

Sécurisez le canal SSL avec des clés signées par une autorité de certification, ou avec des clés autosignées.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le MQTT protocol sur WebSockets sécurisé.

   - IBM MessageSight et les éditions de IBM WebSphere MQ version 7.5.0.1 et ultérieures le font.
2. Facultatif : Installez un kit de développement Java (JDK) de version 7 ou ultérieure.

   Si vous configurez un système de test, et si vous voulez utiliser des certificats autosignés, vous devez utiliser la commande **keytool** du JDK version 7 pour certifier vos certificats. Si vous configurez un système de production, et si vous envoyez des demandes de signature de certificat (CSR) à une autorité de certification externe, vous n'avez pas besoin du JDK version 7.
3. Créez et exécutez les scripts pour générer des paires de clés et des certificats et configurez IBM WebSphere MQ en tant que serveur MQTT .

   Suivez les étapes décrites dans «Génération de clés et de certificats», à la page 96 pour créer et exécuter les scripts. La rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 81 contient les scripts.

   Le nom commun du certificat du serveur doit correspondre au nom DNS du canal du serveur. Certains navigateurs acceptent les certificats qui contiennent la liste des noms communs, par exemple :

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

   D'autres navigateurs acceptent un seul nom commun. Par exemple, Firefox jusqu'à la version 18 accepte un seul nom commun. Cela peut être différent pour les versions suivantes.
4. Vérifiez que les canaux SSL sont opérationnels et configurés conformément à vos attentes.

   Dans IBM WebSphere MQ, entrez la commande suivante dans une fenêtre de commande :

   - 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. Installez les certificats dans le magasin de certificats du navigateur.

   Pour le modèle, choisissez l'un des certificats serveur suivants :

   a. Dans le cas d'un certificat serveur autosigné, le certificat est `srvcertselfsigned.cer`.

   b. Pour un certificat serveur signé par votre autorité de certification privée, le certificat est `cacert.cer`.

   c. Pour un certificat serveur signé par une autorité de certification externe, vérifiez si le certificat racine de l'autorité de certification est déjà installé dans le magasin de certificats.

Selon la prise en charge disponible pour votre navigateur, installez `cacert.cer` dans la liste des autorités de certification racine de confiance. Voir «Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL», à la page 176.

6. Facultatif : Authentifiez le client.

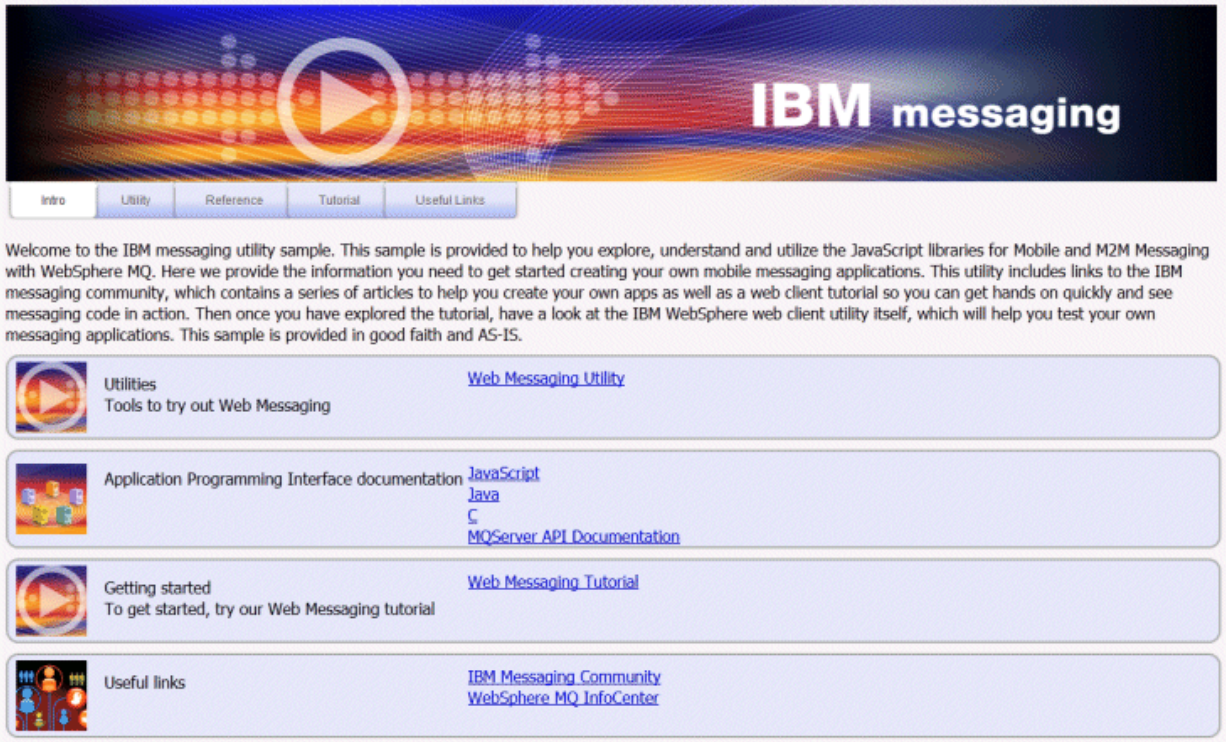
   Dans le modèle, vous installez `cacert.cer` pour authentifier le serveur et chiffrer le canal, mais pas pour authentifier le client. Pour authentifier le client, vous devez installer le magasin de clés du client, `cltkeystore.p12`, dans le navigateur. Tous les navigateurs ne prennent pas en charge l'authentification des clients. Voir «Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL», à la page 176.

7. Connectez-vous au canal WebSockets sécurisé.

   Ouvrez votre navigateur et entrez l'URL du canal WebSockets dans la barre d'adresse. Dans le modèle :

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

   IBM WebSphere MQ répond en affichant la première page du Pages de l'exemple de client de messagerie MQTT JavaScript.

   Si la connexion échoue, et si vous exécutez les modèles de script pour configurer le modèle de gestionnaire de files d'attente MQTT, tentez de vous connecter au canal WebSockets normal sur le port 1886. Si vous y réussissez sur le port 1886, c'est que la défaillance provient de la connexion SSL.

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

Concepts associés

«Le client de messagerie MQTT pour JavaScript et les apps Web», à la page 118

«Comment programmer des apps de messagerie dans JavaScript», à la page 122

Tâches associées

«Génération de clés et de certificats», à la page 96

La procédure qui suit permet de générer des clés et des certificats pour les clients Java et C, y compris les applications Android et iOS, et les serveurs IBM WebSphere MQ et IBM MessageSight.

«Initiation au client de messagerie MQTT pour JavaScript», à la page 24

Vous pouvez vous familiariser avec le client de messagerie MQTT pour JavaScript en affichant la page d'accueil du modèle de client de messagerie, et en navigant dans les ressources accessibles par les liens. Pour afficher cette page d'accueil, configurez un serveur MQTT de manière qu'il accepte les connexions du Pages de l'exemple de client de messagerie MQTT JavaScript, puis entrez l'URL que vous avez configurée sur le serveur dans un navigateur Web. Le client de messagerie MQTT pour JavaScript démarre automatiquement sur votre périphérique, et la page d'accueil du modèle de client de messagerie s'affiche. Cette page contient des liens vers des utilitaires, la documentation de l'API, un tutoriel et d'autres informations utiles.

Référence associée

«Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL», à la page 176

Les différents navigateurs, combinés aux différentes plateformes, présentent des fonctionnalités différentes. La compréhension de ces différences vous aidera à configurer vos applications, autorités de certification et certificats client pour une connexion à l'aide du client de messagerie MQTT pour JavaScript sur SSL et WebSockets.

Exemples de script permettant de configurer des certificats SSL pour Windows

** **

Les exemples de fichier de commandes créent les certificats et les magasins de certificats, selon la procédure décrite dans cette tâche. En outre, l'exemple configure l'utilisation du magasin de certificats du serveur dans le gestionnaire de files d'attente du client MQTT. Il supprime et recrée le gestionnaire de files d'attente en appelant le script `SampleMQM.bat` fourni avec IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` définit les noms et les chemins des certificats et des autres paramètres nécessaires aux commandes **`keytool`** et **openSSL**. Les paramètres sont décrits dans les commentaires du script.

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

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

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

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

```
set cajkskeystorepass=%password%
@rem ca certificate (root certificate)
set cacert=%certpath%\cacert.cer
```

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

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

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

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

```
set cltsrvpemtruststore=%certpath%\cltsrvtruststore.pem
set cltsrvpemtruststorepass=%password%
```

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

cleancert.bat

Les commandes du script `cleancert.bat` suppriment le gestionnaire de files d'attente du client MQTT qui pourraient éventuellement verrouiller le magasin de certificats, puis tous les magasins de clés et les certificats créés par les exemples de script de sécurité.

```
@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

Les commandes du script `genkeys.bat` créent les paires de clés destinées à votre autorité de certification privée, au serveur et à un client.

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

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

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

sscerts.bat

Les commandes du script `sscerts.bat` exportent les certificats autosignés du client et du serveur de leurs magasins de clés, et importent le certificat du serveur dans le magasin de clés de confiance du client, et le certificat du client dans le magasin de clés du serveur. Le serveur n'a pas de magasin de clés de confiance. Les commandes créent un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

```
@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

Le script importe le certificat racine de l'autorité de certification dans les magasins de clés privés. Ce certificat est nécessaire pour créer la chaîne de certificats contenant le certificat racine et le certificat signé. Le script `cacerts.bat` exporte les demandes de certificat client et serveur de leurs magasins de clés respectifs. Le script signe les demandes de certificat avec la clé de l'autorité de

certification privée dans le magasin de clés `cajkskeystore.jks`, puis réimporte les certificats dans les magasins de clés d'où provenaient les demandes. L'importation crée la chaîne de certificats avec le certificat racine de l'autorité de certification. Le script crée un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

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

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

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

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

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

```
%keytool% -delete -alias %caalias% -keystore %cltjkskeystore% -storepass
%cltjkskeystorepass%
```

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

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

mqcerts.bat

Le script répertorie les magasins de clés et les certificats qui se trouvent dans le répertoire des certificats. Il crée ensuite le gestionnaire de files d'attente de l'exemple MQTT et configure les canaux de télémétrie sécurisés.

```
@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
```

Génération et exécution du Exemple d'application C du client MQTT

Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app C sécurisé sur tous les systèmes sur lesquels vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle d'app C sur IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

Avant de commencer

1. Vous devez avoir accès à un serveur MQTT version 3.1 qui prend en charge MQTT protocol sur SSL.
2. S'il existe un pare-feu entre votre client et le serveur, vérifiez qu'il ne bloque pas le trafic MQTT .
3. Les versions binaires des bibliothèques du client pour C sont fournies pour différents systèmes d'exploitation. Pour certains de ces systèmes, la version sécurisée du client n'est pas fournie sous

la forme d'un fichier binaire. Dans ce cas, vous devez suivre les instructions de la rubrique «Création des bibliothèques du client MQTT pour C», à la page 31.

4. Dans le cadre du traitement des incidents, le support IBM peut vous demander d'exécuter le client MQTT pour C sur une plateforme de référence.
5. Les canaux SSL doivent être démarrés.

Pour connaître les plateformes client prises en charge et celles de référence, voir Configuration système requise pour IBM Module de clientMobile Messaging et M2M. Pour des informations détaillées sur la configuration prise en charge pour le client C, consultez les sections correspondantes de la page System Requirements for WebSphere MQ V7.5 Telemetry.

Pourquoi et quand exécuter cette tâche

La présente rubrique illustre la manière de compiler et d'exécuter le Exemple d'application C du client MQTT sécurisé sous Windows à partir de la ligne de commande. Dans cet exemple, Microsoft Visual Studio 2010 est utilisé pour compiler le client. Vous pouvez modifier les scripts de ligne de commande pour compiler et exécuter le modèle d'app sur d'autres systèmes d'exploitation, tels que Linux et iOS.

Remarque :

Les scripts Windows fournis dans cette rubrique sont basées sur l'hypothèse que vous générez la totalité du package OpenSSL à partir du code source. Si vous optez pour les bibliothèques précompilées fournies par IBM, vous préférerez peut-être également vous procurer une version binaire précompilée d'OpenSSL. Il n'y a pas de bibliothèques précompilées utilisables avec iOS.

Sécurisez le canal SSL avec des clés signées par une autorité de certification, ou avec des clés autosignées.

Procédure

1. Choisissez un serveur MQTT auquel vous pouvez connecter l'application client.

   Le serveur doit prendre en charge le protocole MQTT version 3.1 sur SSL. Tous les serveurs MQTT de IBM le font, y compris IBM WebSphere MQ et IBM MessageSight. Consultez «Initiation aux serveurs MQTT», à la page 138.
2. Facultatif : Installez un kit de développement Java (JDK) version 7 ou ultérieure.

   version 7 est requis pour exécuter la commande **keytool** afin de certifier les certificats. Si vous n'avez pas besoin de certifier des certificats, la version 7 de JDK n'est pas nécessaire.
3. Installez un environnement de développement C sur la plateforme de génération.

   Les fichiers makefile dans les exemples de cette rubrique ciblent les outils suivants :

   - iOS Pour iOS, sous Apple Mac avec OS X 10.8.2 et les outils de développement iOS du site Xcode.
   - Linux Pour Linux, la version gcc 4.4.6 de Red Hat Enterprise Linux version 6.2.

     Le plus bas niveau pris en charge de la bibliothèque C `glibc` est 2.12, et celui du noyau Linux est 2.6.32.
   - Windows Pour Microsoft Windows, Visual Studio version 10.0.
4. Téléchargez le Module de clientMobile Messaging et M2M et installez le SDK MQTT .

   Il n'existe aucun programme d'installation, il suffit d'étendre le fichier téléchargé.

   a. Téléchargez le Module de clientMobile Messaging et M2M.
   b. Créez le dossier dans lequel vous allez installer le kit de développement de logiciels.

      Vous pouvez nommer le dossier `MQTT`. Le chemin d'accès à ce dossier est appelé ici *`sdkroot`*.
   c. Développez le contenu du fichier Module de clientMobile Messaging et M2M compressé dans *`sdkroot`*. L'extension crée une arborescence de répertoires qui commence à *`sdkroot`*\SDK.

5. Facultatif : Suivez les étapes décrites dans «Création des bibliothèques du client MQTT pour C», à la page 31.

   Ne réalisez cette étape que si le kit de développement de logiciels MQTT ne contient pas la bibliothèque du client C sécurisé destinée à votre système d'exploitation.

   - Windows Les bibliothèques sont `mqttv3cs.lib` pour la compilation, et `mqttv3cs.dll` pour l'exécution.
   - Linux La bibliothèque est `libmqttv3cs.so`.
   - iOS La bibliothèque est `libmqttv3cs.a`.
6. Créez et exécutez les scripts pour générer des paires de clés et des certificats et configurez IBM WebSphere MQ en tant que serveur MQTT .

   Suivez les étapes décrites dans «Génération de clés et de certificats», à la page 96 pour créer et exécuter les scripts. La rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 90 contient les scripts.
7. Vérifiez que les canaux SSL sont opérationnels et configurés conformément à vos attentes.

   Dans IBM WebSphere MQ, entrez la commande suivante dans une fenêtre de commande :

   - 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. Créez les scripts destinés à la génération et à l'exécution du Exemple d'application C du client MQTT sécurisé.
   a) Créez et exécutez `sscclient.bat` pour tester un canal SSL sécurisé avec des certificats autosignés.
   b) Créez et exécutez `cacclient.bat` pour tester un canal SSL sécurisé avec des certificats signés par l'autorité de certification.

Résultats

Les résultats sont semblables à ceux obtenus lors de l'exécution du client non sécurisé.

```
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
```

Figure 16. Abonné sécurisé

```
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 . . .
```

Figure 17. Diffuseur de publication sécurisé

Scripts d'exécution du Exemple d'application C du client MQTT sécurisé

Avant d'exécuter les scripts qui suivent, exécutez ceux de la rubrique «Exemples de script permettant de configurer des certificats SSL pour Windows», à la page 90.

Exemple d'application C du client MQTT sécurisé avec certificats autosignés.

Exécutez ce script avec les certificats autosignés que vous avez créés en exécutant le 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
```

Figure 18. `sscclient.bat`

Exécutez le modèle d'app C client MQTT sécurisé avec des certificats signés par l'autorité de certification.

Exécutez ce script avec les certificats signés par l'autorité de certification que vous avez créés en exécutant le 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
```

Figure 19. `cacclient.bat`

Concepts associés

«Sécurité MQTT», à la page 52
Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. L'identification consiste à nommer le client auquel on donne des droits d'accès. L'authentification cherche à prouver l'identité du client, et l'autorisation consiste à gérer ses droits.

Tâches associées

«Génération de clés et de certificats», à la page 96
La procédure qui suit permet de générer des clés et des certificats pour les clients Java et C, y compris les applications Android et iOS, et les serveurs IBM WebSphere MQ et IBM MessageSight.

Exemples de script permettant de configurer des certificats SSL pour Windows

Exemple

Les exemples de fichier de commandes créent les certificats et les magasins de certificats, selon la procédure décrite dans cette tâche. En outre, l'exemple configure l'utilisation du magasin de certificats du serveur dans le gestionnaire de files d'attente du client MQTT. Il supprime et recrée le gestionnaire de files d'attente en appelant le script `SampleMQM.bat` fourni avec IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` définit les noms et les chemins des certificats et des autres paramètres nécessaires aux commandes **keytool** et **openSSL**. Les paramètres sont décrits dans les commentaires du 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

Les commandes du script `cleancert.bat` suppriment le gestionnaire de files d'attente du client MQTT qui pourraient éventuellement verrouiller le magasin de certificats, puis tous les magasins de clés et les certificats créés par les exemples de script de sécurité.

```
@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

Les commandes du script `genkeys.bat` créent les paires de clés destinées à votre autorité de certification privée, au serveur et à un client.

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

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

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

sscerts.bat

Les commandes du script `sscerts.bat` exportent les certificats autosignés du client et du serveur de leurs magasins de clés, et importent le certificat du serveur dans le magasin de clés de confiance du client, et le certificat du client dans le magasin de clés du serveur. Le serveur n'a pas de magasin de clés de confiance. Les commandes créent un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

```
@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

Le script importe le certificat racine de l'autorité de certification dans les magasins de clés privés. Ce certificat est nécessaire pour créer la chaîne de certificats contenant le certificat racine et le certificat signé. Le script `cacerts.bat` exporte les demandes de certificat client et serveur de leurs magasins de clés respectifs. Le script signe les demandes de certificat avec la clé de l'autorité de certification privée dans le magasin de clés `cajkskeystore.jks`, puis réimporte les certificats dans les magasins de clés d'où provenaient les demandes. L'importation crée la chaîne de certificats avec le certificat racine de l'autorité de certification. Le script crée un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

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

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

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

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

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

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

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

mqcerts.bat

Le script répertorie les magasins de clés et les certificats qui se trouvent dans le répertoire des certificats. Il crée ensuite le gestionnaire de files d'attente de l'exemple MQTT et configure les canaux de télémétrie sécurisés.

```
@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
```

Génération de clés et de certificats

La procédure qui suit permet de générer des clés et des certificats pour les clients Java et C, y compris les applications Android et iOS, et les serveurs IBM WebSphere MQ et IBM MessageSight.

Avant de commencer

1. Vous devez disposer d'une copie de la commande **keytool**. Toutes les versions de **keytool** ne prennent pas en charge la conversion des magasins de clés JKS (Java keystore) en PKCS (Public Key Cryptographic System) ou la signature des demandes de certificat. L'exemple utilise la commande **keytool** de JDK version 7.0, qui prend en charge ces deux fonctions.
2. Si vous voulez générer des clés et des certificats pour le client pour C, qui sont au format PEM (Privacy-Enhanced Mail), vous devez disposer d'une copie de la commande **openSSL**. Suivez la procédure décrite à la rubrique «Création des bibliothèques du client MQTT pour C», à la page 31 pour générer le package openSSL.
3. Modifiez la valeur des paramètres du script `initcert.bat` en fonction de vos besoins. En particulier, vous voudrez peut-être omettre les paramètres de mot de passe pour ne pas avoir à les enregistrer. La commande **keytool** vous demande les mots de passe manquants.

Pourquoi et quand exécuter cette tâche

La création de connexions SSL sécurisées entre les clients et les serveurs MQTT nécessite des clés et des certificats. Cette tâche vous montre deux manières de créer les clés et les certificats dont vous avez besoin : autosignés et signés par votre propre autorité de certification. La méthode que vous devez suivre dépend de la manière dont vous prévoyez de gérer les magasins de clés et les certificats.

Pour utiliser des certificats signés par une autorité de certification extérieure, remplacez l'étape de signature du fichier `cacerts.bat` par l'envoi des demandes de certificat à cette autorité. L'autorité de certification peut aussi renvoyer un certificat intermédiaire et un certificat racine en plus du certificat signé. Suivez les instructions fournies par l'autorité de certification extérieure pour installer les certificats renvoyés.

Le serveur IBM WebSphere MQ ne recherche les certificats que dans le magasin de certificats que vous définissez dans les paramètres de configuration du canal de télémétrie. Il ne les recherche pas dans le magasin `cacerts` de JSE. Un client Java recherche les certificats dans le magasin de clés de confiance défini par vous. Si vous n'en définissez pas, il recherche dans le magasin de clés `cacerts` du répertoire JSE `jre\lib\security`. Les clients Android recherchent les certificats dans le magasin de certificats prédéfini sur le périphérique Android. Les applications client C et iOS ne les recherchent que dans les magasins définis par elles-même.

Les clients Android et Java recherchent un magasin de clés de confiance préconfigurés pour les certificats de confiance. Les certificats racine de l'autorité de certification sont stockés dans le magasin de certificats de confiance Android et dans le magasin JSE `jre\lib\security\cacerts` . Si le certificat racine de l'autorité de certification qui a certifié le certificat du serveur est déjà installé dans le magasin de clés de confiance pré configuré, ne définissez pas de magasin de clés de confiance client. La seule configuration requise consiste à définir le port TCP/IP du canal sécurisé du serveur MQTT.

Les outils destinés à créer les clés et les certificats, et à gérer les différents formats, ne sont pas simples à utiliser. De nombreuses valeurs sont à paramétrer, et **openSSL** nécessite un fichier de configuration, `openssl.cnf`, et des paramètres de ligne de commande. Il n'existe pas d'outil capable de fournir

l'ensemble des fonctions requises pour gérer les clés et les certificats des applications qui fonctionnent sur C et sur Java. Les canaux de télémétrie de IBM WebSphere MQ ont besoin d'un magasin de clés de type JKS. En conséquence les exemples utilisent principalement les outils Java de gestion de certificats **ikeyman** et **keytool**. Cependant, les outils Java ne prennent pas en charge le format PEM, requis pour les applications client C. Pour créer des magasins de clés au format PEM, utilisez l'outil **openSSL**. L'outil **openSSL** convertit au format PEM les magasins de clés de type PKCS12, et **keytool** fait la conversion entre les formats JKS et PKCS12. Le fichier `openssl.cnf` n'est pas requis pour la conversion des magasins de clés. **openSSL** n'est nécessaire que si vous prévoyez de générer des applications client C ou iOS. Si vous préférez utiliser **openSSL**, vous pouvez vous en servir pour signer les certificats, au lieu de le faire avec **keytool**.

Procédure

1. Ouvrez une fenêtre de commande pour exécuter les scripts suivants.
2. Créez et exécutez le script `initcert.bat` pour définir les paramètres requis pour exécuter les exemples de clients sécurisés MQTT .
3. Créez et exécutez le script `cleancert.bat` pour nettoyer l'environnement en vue de la création de nouveaux magasins de clés et certificats.
4. Créez et exécutez le script `genkeys.bat` pour générer les paires de clés dont vous avez besoin.
5. Effectuez l'une des opérations ci-dessous :
   - Créez et exécutez le script `sscerts.bat` pour créer des certificats autosignés.
   - Créez et exécutez le script `cacerts.bat` pour créer des trousseaux de certificats signés par une autorité de certification.
6. Créez et exécutez le script `mqcerts.bat` pour créer le gestionnaire de files d'attente `MQXR_SAMPLE_QM` et configurer ses canaux de télémétrie.

Tâches associées

«Génération et exécution du Exemple d'application C du client MQTT», à la page 86
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app C sécurisé sur tous les systèmes sur lesquels vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle d'app C sur IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

«Génération et exécution du Exemple d'application Java client MQTT», à la page 56
Un exemple sous Windows vous permet de vous familiariser avec le modèle d'app Java sécurisé sous IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.Vous pouvez exécuter une application client MQTT pour Java sur n'importe quelle plateforme avec JSE 1.5 ou version ultérieure, c'est-à-dire "Java Compatible"

Exemples de script permettant de configurer des certificats SSL pour Windows

Exemple

Les exemples de fichier de commandes créent les certificats et les magasins de certificats, selon la procédure décrite dans cette tâche. En outre, l'exemple configure l'utilisation du magasin de certificats du serveur dans le gestionnaire de files d'attente du client MQTT. Il supprime et recrée le gestionnaire de files d'attente en appelant le script `SampleMQM.bat` fourni avec IBM WebSphere MQ.

`initcert.bat`

`initcert.bat` définit les noms et les chemins des certificats et des autres paramètres nécessaires aux commandes **keytool** et **openSSL**. Les paramètres sont décrits dans les commentaires du 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

Les commandes du script `cleancert.bat` suppriment le gestionnaire de files d'attente du client MQTT qui pourraient éventuellement verrouiller le magasin de certificats, puis tous les magasins de clés et les certificats créés par les exemples de script de sécurité.

```
@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

Les commandes du script genkeys.bat créent les paires de clés destinées à votre autorité de certification privée, au serveur et à un client.

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

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

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

sscerts.bat

Les commandes du script sscerts.bat exportent les certificats autosignés du client et du serveur de leurs magasins de clés, et importent le certificat du serveur dans le magasin de clés de confiance du client, et le certificat du client dans le magasin de clés du serveur. Le serveur n'a pas de magasin de clés de confiance. Les commandes créent un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

```
@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

Le script importe le certificat racine de l'autorité de certification dans les magasins de clés privés. Ce certificat est nécessaire pour créer la chaîne de certificats contenant le certificat racine et le certificat signé. Le script `cacerts.bat` exporte les demandes de certificat client et serveur de leurs magasins de clés respectifs. Le script signe les demandes de certificat avec la clé de l'autorité de certification privée dans le magasin de clés `cajkskeystore.jks`, puis réimporte les certificats dans les magasins de clés d'où provenaient les demandes. L'importation crée la chaîne de certificats avec le certificat racine de l'autorité de certification. Le script crée un magasin de clés de confiance client au format PEM à partir de celui qui est au format JKS.

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

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

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

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

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

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

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

mqcerts.bat

Le script répertorie les magasins de clés et les certificats qui se trouvent dans le répertoire des certificats. Il crée ensuite le gestionnaire de files d'attente de l'exemple MQTT et configure les canaux de télémétrie sécurisés.

```
@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
```

Identification, autorisation et authentification du client MQTT

Le service de télémétrie (MQXR) diffuse des publications ou s'abonne aux rubriques WebSphere MQ au nom des clients MQTT, à l'aide des canaux MQTT. L'administrateur WebSphere MQ configure l'identité du canal MQTT qui est utilisé pour l'autorisation WebSphere MQ. L'administrateur peut définir une identité commune pour le canal ou utiliser le `Nom d'utilisateur` ou l'`Identificateur client` d'un client connecté au canal.

Le service de télémétrie (MQXR) peut authentifier le client à l'aide du `Nom d'utilisateur` fourni par le client ou à l'aide d'un certificat client. Le `Nom d'utilisateur` est authentifié à l'aide d'un mot de passe fourni par le client.

En résumé : l'identification client est la sélection de l'identité du client. En fonction du contexte, le client est identifié par l'`Identificateur client`, le `Nom d'utilisateur`, une identité client commune créée par l'administrateur ou un certificat client. L'identificateur client utilisé pour le contrôle d'authenticité ne doit pas être le même que celui utilisé pour l'autorisation.

Les programmes du client MQTT définissent les `Nom d'utilisateur` et `Mot de passe` qui sont envoyés au serveur à l'aide d'un canal MQTT. Ils peuvent également définir les propriétés SSL requises pour chiffrer et authentifier la connexion. L'administrateur décide s'il est nécessaire d'authentifier le canal MQTT et comment procéder.

Pour autoriser un client MQTT à accéder aux objets IBM WebSphere MQ, accordez l'autorisation d'accès à l'`Identificateur client` ou au `nom d'utilisateur`, ou accordez l'autorisation à une identité client commune. Pour autoriser un client à se connecter à IBM WebSphere MQ, vous devez authentifier le `nom d'utilisateur` ou utiliser un certificat client. Configurez JAAS pour authentifier le `Nom d'utilisateur` et configurer SSL pour authentifier un certificat client.

Si vous définissez un `Mot de passe` au niveau du client, vous devez chiffrer la connexion à l'aide du réseau privé virtuel (VPN) ou configurer le canal MQTT pour utiliser SSL, afin que le mot de passe reste privé.

Il est difficile de gérer des certificats client. Pour cette raison, si les risques associés à l'authentification par mot de passe sont acceptables, ce type d'authentification est souvent utilisé pour l'authentification des clients.

S'il existe un moyen sûr de gérer et de stocker le certificat client, il est possible de faire confiance à l'authentification par certificat. Toutefois, il arrive rarement que les certificats puissent être gérés de manière sécurisée dans les types d'environnements dans lesquels la télémétrie est utilisée. L'authentification des dispositifs utilisant les certificats client est complétée par l'authentification des mots de passe client au niveau du serveur. En raison de la complexité supplémentaire, l'utilisation du certificat client est limitée aux applications hautement sensibles. L'utilisation de deux formulaires d'authentification est appelée authentification à deux facteurs. Vous devez connaître l'un des facteurs, tels que le mot de passe, et disposer de l'autre, un certificat, par exemple.

Dans une application sensible, telle qu'un appareil utilisant un code confidentiel, l'appareil est verrouillé pendant sa fabrication pour empêcher toute contrefaçon avec le matériel et le logiciel internes. Un certificat client digne de confiance, limité dans le temps, est copié dans le dispositif. Le dispositif est déployé à l'emplacement où il doit être utilisé. Une authentification supplémentaire est effectuée chaque fois que le dispositif est utilisé, soit à l'aide d'un mot de passe, soit en utilisant un autre certificat à partir d'une carte à puce.

MQTT client - Identité et autorisation

Utilisez l'`Identificateur client`, le `Nom d'utilisateur` ou une identité client commune pour accorder l'autorisation d'accès aux objets WebSphere MQ.

L'administrateur IBM WebSphere MQ dispose de trois options pour sélectionner l'identité du canal MQTT. L'administrateur effectue son choix lors de la définition ou de la modification du canal MQTT utilisé par le client. L'identité est utilisée pour autoriser l'accès aux rubriques IBM WebSphere MQ. Les choix possibles sont les suivants :

1. L'identificateur client.
2. Une identité fournie au canal par l'administrateur.
3. Le `Nom d'utilisateur` transmis à partir du client MQTT.

Le `Nom d'utilisateur` est un attribut de la classe MqttConnectOptions. Il doit être défini avant que le client ne se connecte au service. Sa valeur par défaut est Null.

Utilisez la commande IBM WebSphere MQ **setmqaut** pour sélectionner les objets et les actions qui sont autorisés à être utilisés par l'identité associée au canal MQTT. Par exemple, pour autoriser une identité de canal, `MQTTClient`, fournie par l'administrateur de gestionnaire de files d'attente, `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
```

Information associée

Autorisation de l'accès aux objets WebSphere MQ pour les clients MQTT

Authentification du client MQTT à l'aide d'un mot de passe

Authentifiez le `Nom d'utilisateur` à l'aide du mot de passe du client. Vous pouvez authentifier le client en utilisant une identité différente de celle utilisée pour autoriser le client à diffuser des publications et à s'abonner aux sujets.

Le service de télémétrie (MQXR) utilise JAAS pour authentifier le `Nom d'utilisateur` du client. JAAS utilise le `mot de passe` fourni par le client MQTT.

L'administrateur IBM WebSphere MQ décide s'il convient d'authentifier le `Nom d'utilisateur`, ou de ne pas l'authentifier du tout en configurant le canal MQTT auquel le client se connecte. Les clients peuvent se voir attribuer des canaux différents et chaque canal peut être configuré pour authentifier ses clients de différentes façons. A l'aide de JAAS, vous pouvez décider quelles sont les méthodes qui doivent authentifier le client, et celles qui peuvent authentifier le client.

Le choix de l'identité pour l'authentification n'affecte pas le choix de l'identité pour l'autorisation. Vous pouvez définir une identité commune pour l'autorisation à des fins d'administration, mais authentifier chaque utilisateur afin qu'il puisse utiliser cette identité. La procédure suivante met en évidence les étapes permettant d'authentifier des utilisateurs individuels pour l'utilisation d'une identité commune :

1. L'administrateur IBM WebSphere MQ définit une identité de canal MQTT pour chaque nom, tel que `MQTTClientUser`, à l'aide d'IBM WebSphere MQ Explorer.
2. L'administrateur IBM WebSphere MQ autorise `MQTTClient` à diffuser des publications et à s'abonner à n'importe quelle rubrique :

   ```
   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. Le développeur d'app client MQTT crée un objet `MqttConnectOptions` et définit le `Nom d'utilisateur` et le `Mot de passe` avant de se connecter au serveur.
4. Le développeur de sécurité crée un JAAS `LoginModule` pour authentifier le `nom d'utilisateur` avec le `mot de passe` et l'inclut dans le fichier de configuration JAAS .
5. L'administrateur IBM WebSphere MQ configure le canal MQTT pour authentifier le `Nom d'utilisateur` du client à l'aide de JAAS.

Authentification du client MQTT à l'aide de SSL

Les connexions entre le client MQTT et le gestionnaire de files d'attente sont toujours initiées par le client MQTT. Le client MQTT est toujours le client SSL. L'authentification de client du serveur et l'authentification de serveur du client MQTT sont toutes les deux facultatives.

En fournissant au client un certificat numérique signé privé, vous pouvez authentifier le client MQTT auprès d' IBM WebSphere MQ. L'administrateur d'IBM WebSphere MQ peut forcer les clients MQTT à s'authentifier sur le gestionnaire de files d'attente à l'aide de SSL. Vous pouvez uniquement demander l'authentification de client comme faisant partie d'une authentification mutuelle.

Comme alternative à l'utilisation de SSL, certains types de réseaux privés virtuels (VPN), tels que IPsec, authentifient les noeuds finaux d'une connexion TCP/IP. Le réseau privé virtuel VPN chiffre chaque paquet transitant sur le réseau. Une fois qu'une telle connexion VPN est établie, vous avez établi un réseau sécurisé. Vous pouvez connecter les clients MQTT aux canaux de télémétrie avec TCP/IP sur le réseau VPN.

L'authentification de client à l'aide de SSL suppose que le client possède une clé confidentielle. La clé confidentielle est la clé privée du client dans le cas d'un certificat autosigné ou une clé fournie par une autorité de certification. La clé permet de signer le certificat numérique du client. Toute personne en possession de la clé publique correspondante peut vérifier le certificat numérique. Les certificats peuvent être accrédités, ou s'ils font partie d'une chaîne, il est possible de remonter par l'intermédiaire de la chaîne jusqu'au certificat racine accrédité. L'opération de vérification du client envoie tous les certificats de la chaîne fournie par le client au serveur. Le serveur vérifie la chaîne de certificats jusqu'à ce qu'il trouve un certificat digne de confiance. Il s'agit soit du certificat public généré par un certificat autosigné, soit d'un certificat racine généralement émis par une autorité de certification. Au cours de l'étape finale facultative le certificat de confiance peut être comparé à une liste de révocation de certificats "en direct".

Le certificat de confiance peut être émis par une autorité de certification et inclus dans l'espace de stockage de certificats de JRE. Il peut s'agir d'un certificat autosigné ou de tout certificat qui a été ajouté au magasin de clés de canal de télémétrie en tant que certificat digne de confiance.

Remarque : Le canal de télémétrie dispose d'un magasin servant à la fois de magasin de clés et de magasin de clés de confiance, qui contient les clés privées d'accès à un ou plusieurs canaux de télémétrie et les certificats publics nécessaires pour authentifier les clients. Comme un canal SSL doit être associé à un magasin de clés et que ce magasin est le même fichier que le magasin de clés de confiance du canal, l'espace de stockage des certificats du JRE n'est jamais référencé. Il en résulte que si l'authentification d'un client requiert un certificat racine de l'autorité de certification, vous devez placer ce certificat dans le magasin de clés du canal, même si le certificat racine de l'autorité de certification est déjà présent dans l'espace de stockage des certificats du JRE. L'espace de stockage des certificats du JRE n'est jamais référencé.

Ayez à l'esprit les menaces que l'authentification de client peut déjouer et le rôle du client et du serveur pour faire échouer ces menaces. L'authentification du certificat client en soi ne suffit pas à empêcher l'accès non autorisé à un système. Si quelqu'un d'autre a pris le possession du dispositif client, ce dernier ne s'exécute pas nécessairement avec les droits du détenteur du certificat. Ne faites jamais confiance à une protection unique contre des attaques non désirées. Utilisez au moins une approche de type authentification à deux facteurs et ajoutez en supplément à la possession d'un certificat la connaissance d'informations privées. Par exemple, utilisez JAAS, et authentifiez le client à l'aide d'un mot de passe émis par le serveur.

La principale menace à laquelle est confronté le certificat client est de se trouver aux mains de quelqu'un de mal intentionné. Le certificat se trouve dans un magasin de clés protégé par mot de passe au niveau du client. Comment se retrouve-t-il dans le magasin de clés ? Comment le client MQTT parvient-il à se procurer le mot de passe du magasin de clés ? Quel est le degré de sécurité de la protection par mot de passe ? Les dispositifs de télémétrie sont faciles à supprimer et peuvent donc être piratés en privé. Le dispositif matériel peut-il être imperméable aux actes de malveillance ? La distribution et la protection de certificat côté client sont reconnues comme un acte difficile, on parle alors de problème gestion de clés.

La menace secondaire est le cas ou le dispositif est mal utilisé pour accéder au serveur de manière non intentionnelle. Par exemple, si l'application MQTT est trafiquée il est possible d'utiliser une faiblesse dans la configuration du serveur à l'aide de l'identité du client authentifié.

Pour authentifier un client MQTT à l'aide de SSL, configurez le canal de télémétrie et le client.

•
•

Configuration du client MQTT pour son authentification par SSL

Pour authentifier le client MQTT à l'aide de SSL, le client se connecte à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port TCP qui correspond au canal de télémétrie qui est configuré pour l'authentification des clients SSL.

Par exemple, côté client :

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

La machine virtuelle Java du client doit utiliser la fabrique de connecteur standard à partir de JSSE. Si vous utilisez Java ME, vous devez vous assurer que le package JSSE est chargé. Si vous utilisez Java SE, JSSE est inclus dans l'environnement d'exécution Java depuis la version Java 1.4.1.

La connexion SSL nécessite la définition de plusieurs propriétés SSL avant la connexion. Vous pouvez définir les propriétés soit en les transmettant à la machine virtuelle Java à l'aide du commutateur `-D` , soit en définissant les propriétés à l'aide de la méthode `MqttConnectionOptions.setSSLProperties` .

Si vous chargez une fabrique de sockets non standard, en appelant la méthode `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, la manière dont les paramètres SSL sont transmis au socket réseau est définie par l'application.

Ajoutez le certificat numérique du client, signé à l'aide de la clé privée du client ou d'une autorité de certification, au magasin de clés protégé par mot de passe au niveau du client. Si le certificat a un trousseau, vous pouvez ajouter les certificats qu'il contient au magasin. Lorsque le serveur vérifie le certificat client, il utilise les certificats envoyés par le client pour les comparer aux certificats se trouvant dans son magasin de clés. Il recherche dans la chaîne de certificats la première correspondance avec un certificat qu'il possède. Le reste de la chaîne est ignoré.

Le client MQTT envoie tous les certificats de son magasin de clés au serveur. Si le serveur authentifie l'une des chaînes de certificats envoyées par le client, le client est authentifié.

Vous pouvez également utiliser les suites de chiffrement SSL pour l'authentification du client.Voici la liste alphabétique des suites de chiffrement actuellement prises en charge :

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`

- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 Si vous envisagez d'utiliser des suites de chiffrement SHA-2, voir «Configuration système requise pour l'utilisation des suites de chiffrement SHA-2 avec les clients MQTT», à la page 175.

Concepts associés

«Configuration du client MQTT l'authentification du canal par SSL», à la page 108
Pour authentifier le canal de télémétrie à l'aide de SSL, le client doit se connecter à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port qui correspond au canal de télémétrie qui est configuré pour SSL. La configuration doit comprendre un magasin de clés protégé par phrase passe contenant un certificat numérique signé du serveur de manière privée.

Authentification du canal de télémétrie à l'aide de SSL

Les connexions entre le client MQTT et le gestionnaire de files d'attente sont toujours initiées par le client MQTT. Le client MQTT est toujours le client SSL. L'authentification de client du serveur et l'authentification de serveur du client MQTT sont toutes les deux facultatives.

Le client tente toujours d'authentifier le serveur, sauf si le client est configuré pour utiliser un CipherSpec qui prend en charge la connexion anonyme. Si l'authentification échoue, la connexion n'est pas établie.

Comme alternative à l'utilisation de SSL, certains types de réseaux privés virtuels (VPN), tels que IPsec, authentifient les noeuds finaux d'une connexion TCP/IP. Le réseau privé virtuel VPN chiffre chaque paquet transitant sur le réseau. Une fois qu'une telle connexion VPN est établie, vous avez établi un réseau

sécurisé. Vous pouvez connecter les clients MQTT aux canaux de télémétrie avec TCP/IP sur le réseau VPN.

L'authentification de serveur à l'aide de SSL permet d'authentifier le serveur auquel vous êtes sur le point d'envoyer des informations confidentielles. Le client effectue les vérifications correspondant aux certificats envoyés à partir du serveur, par rapport aux certificats placés dans son magasin de clés de confiance ou dans son magasin JRE `cacerts` .

Le magasin de certificats JRE est un fichier JKS, `cacerts`. Il se trouve dans `JRE InstallPath\lib\security\`. Il est installé avec le mot de passe par défaut `changeit`. Vous pouvez stocker les certificats que vous accréditez dans l'espace de stockage des certificats JRE ou dans le magasin de clés de confiance. Vous ne pouvez pas utiliser les deux magasins. Utilisez le fichier de clés certifiées si vous souhaitez conserver les certificats publics et les certificats utilisés par d'autres applications Java à des emplacements distincts. Utilisez l'espace de stockage des certificats JRE si vous souhaitez utiliser un certificat commun pour toutes les applications Java qui s'exécutent sur le client. Si vous décidez d'utiliser cet espace de stockage, vérifiez les certificats qu'il contient afin d'être sûr que vous les accréditez.

Vous pouvez modifier la configuration JSSE en indiquant un autre fournisseur d'accréditation. Vous pouvez personnaliser un fournisseur d'accréditation pour effectuer différentes vérifications sur un certificat. Dans certains environnements OGSi qui ont utilisé le client MQTT, l'environnement fournit un fournisseur d'accréditation différent.

Pour authentifier un canal de télémétrie à l'aide de SSL, configurez le serveur et le client.

•

Concepts associés

«Configuration du client MQTT l'authentification du canal par SSL», à la page 108
Pour authentifier le canal de télémétrie à l'aide de SSL, le client doit se connecter à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port qui correspond au canal de télémétrie qui est configuré pour SSL. La configuration doit comprendre un magasin de clés protégé par phrase passe contenant un certificat numérique signé du serveur de manière privée.

Configuration du client MQTT l'authentification du canal par SSL

Pour authentifier le canal de télémétrie à l'aide de SSL, le client doit se connecter à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port qui correspond au canal de télémétrie qui est configuré pour SSL. La configuration doit comprendre un magasin de clés protégé par phrase passe contenant un certificat numérique signé du serveur de manière privée.

Par exemple, côté client :

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

La machine virtuelle Java du client doit utiliser la fabrique de connecteur standard à partir de JSSE. Si vous utilisez Java ME, vous devez vous assurer que le package JSSE est chargé. Si vous utilisez Java SE, JSSE est inclus dans l'environnement d'exécution Java depuis la version Java 1.4.1.

La connexion SSL nécessite la définition de plusieurs propriétés SSL avant la connexion. Vous pouvez définir les propriétés soit en les transmettant à la machine virtuelle Java à l'aide du commutateur `-D` , soit en définissant les propriétés à l'aide de la méthode `MqttConnectionOptions.setSSLProperties` .

Si vous chargez une fabrique de sockets non standard, en appelant la méthode `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, la manière dont les paramètres SSL sont transmis au socket réseau est définie par l'application.

Codez le client de sorte qu'il se connecte au canal de télémétrie à l'aide de SSL, et configurez le client pour accréditer un certificat serveur de l'une des trois façons suivantes :

Utilisation d'un certificat serveur signé par une autorité de certification reconnue dans le magasin `cacerts`.

Aucune configuration supplémentaire si le serveur envoie toutes les clés intermédiaires dans la chaîne de certificats. Il vous est demandé de réviser les certificats dans le magasin `cacerts` du JRE du client et de changer le mot de passe dans le magasin `cacerts`

Autres certificats

Stockez les certificats que vous accréditez dans le magasin de clés de confiance au niveau du client. Vous devez stocker au moins un des certificats dans la chaîne de certificats du magasin de clés de confiance. Définissez les paramètres du magasin de clés de confiance dans `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Utilisation d'un gestionnaire d'accréditation

Implémentez un fournisseur d'accréditation et transmettez-lui le nom de l'algorithme utilisé. Définissez le nom de la classe du fournisseur et l'algorithme à utiliser dans `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

Vous pouvez également utiliser les suites de chiffrement SSL pour l'authentification du canal.Voici la liste alphabétique des suites de chiffrement actuellement prises en charge :

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`
- `SSL_KRB5_EXPORT_WITH_RC4_40_SHA`
- `SSL_KRB5_WITH_3DES_EDE_CBC_MD5`
- `SSL_KRB5_WITH_3DES_EDE_CBC_SHA`
- `SSL_KRB5_WITH_DES_CBC_MD5`
- `SSL_KRB5_WITH_DES_CBC_SHA`
- `SSL_KRB5_WITH_RC4_128_MD5`

- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 Si vous envisagez d'utiliser des suites de chiffrement SHA-2, voir «Configuration système requise pour l'utilisation des suites de chiffrement SHA-2 avec les clients MQTT», à la page 175.

Concepts associés

«Configuration du client MQTT pour son authentification par SSL», à la page 106
Pour authentifier le client MQTT à l'aide de SSL, le client se connecte à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port TCP qui correspond au canal de télémétrie qui est configuré pour l'authentification des clients SSL.

Confidentialité de la publication sur les canaux de télémétrie

La confidentialité des publications MQTT transitant dans les canaux de télémétrie est assurée par l'utilisation de SSL pour le chiffrement des transmissions au cours de la connexion.

Les clients MQTT qui se connectent aux canaux de télémétrie utilisent SSL pour assurer la sécurité de la confidentialité des publications transmises sur le canal à l'aide du chiffrement par clé symétrique. Etant donné que les noeuds finaux ne sont pas authentifiés, vous ne pouvez pas accréditer le chiffrement de canal seul. Combinez la sécurisation de la confidentialité avec l'authentification du serveur ou l'authentification mutuelle.

Comme alternative à l'utilisation de SSL, certains types de réseaux privés virtuels (VPN), tels que IPsec, authentifient les noeuds finaux d'une connexion TCP/IP. Le réseau privé virtuel VPN chiffre chaque paquet transitant sur le réseau. Une fois qu'une telle connexion VPN est établie, vous avez établi un réseau sécurisé. Vous pouvez connecter les clients MQTT aux canaux de télémétrie avec TCP/IP sur le réseau VPN.

Pour une configuration classique, qui chiffre le canal et authentifie le serveur, consultez la section «Authentification du canal de télémétrie à l'aide de SSL», à la page 107.

Le chiffrement de connexions SSL sans authentification du serveur expose la connexion à des attaques de type "man in the middle". Bien que les informations que vous échangez soient protégées contre les écoutes clandestines, vous ne savez pas avec qui vous réalisez les échanges. A moins que vous ne contrôliez le réseau, vous êtes exposé à ce que quelqu'un intercepte vos transmissions IP, en se faisant passer pour un noeud final.

Vous pouvez créer une connexion SSL cryptée sans authentifier le serveur, à l'aide d'une spécification de chiffrement d'échange de clé Diffie-Hellman qui prend en charge SSL anonyme. Le secret maître, partagé entre le client et le serveur et utilisé pour chiffrer les transmissions SSL, est établi sans échange de certificat de serveur signé de manière privée.

Etant donné que mes connexions anonymes ne sont pas sécurisées, la plupart des implémentations SSL n'utilisent pas par défaut les spécifications CipherSpecs anonymes. Si une demande client de connexion SSL est acceptée par le canal de télémétrie, ce dernier doit disposer d'un magasin de clés protégé par une phrase passe. Par défaut, étant donné que les implémentations n'utilisent pas de spécification de chiffrement anonyme, le magasin de clés doit contenir un certificat signé de manière privée que le client peut authentifier.

Si vous utilisez des spécifications CipherSpecs anonymes, le magasin de clés du serveur doit exister, mais il ne doit pas forcément contenir des certificats signés de manière privée.

Une autre façon d'établir une connexion cryptée consiste à remplacer le fournisseur d'accréditation côté client par votre propre implémentation. Votre fournisseur d'accréditation n'effectuerait pas l'authentification du certificat serveur, mais la connexion serait cryptée.

Configuration SSL des clients MQTT et des canaux de télémétrie

Les clients MQTT et le service WebSphere MQ Telemetry (MQXR) utilisent JSSE (Java Secure Socket Extension) pour connecter les canaux de télémétrie à l'aide de SSL. Les clients MQTT C et le démon pour dispositifs WebSphere MQ Telemetry ne prennent pas en charge SSL.

Configurez SSL pour authentifier le canal de télémétrie et le client MQTT et chiffrer le transfert de messages entre des clients et le canal de télémétrie.

Comme alternative à l'utilisation de SSL, certains types de réseaux privés virtuels (VPN), tels que IPsec, authentifient les noeuds finaux d'une connexion TCP/IP. Le réseau privé virtuel VPN chiffre chaque paquet transitant sur le réseau. Une fois qu'une telle connexion VPN est établie, vous avez établi un réseau sécurisé. Vous pouvez connecter les clients MQTT aux canaux de télémétrie avec TCP/IP sur le réseau VPN.

Vous pouvez configurer la connexion entre un client Java MQTT et un canal de télémétrie pour utiliser le protocole SSL sur TCP/IP. Les éléments sécurisé dépendent de la manière dont vous avez configuré SSL pour utiliser JSSE. En partant de la configuration la plus sécurisée, vous pouvez configurer trois niveaux différents de sécurité :

1. Autorisez uniquement les clients MQTT dignes de confiance à se connecter. Connectez un client MQTT uniquement à un canal de télémétrie digne de confiance. Chiffrez les messages entre le client et le gestionnaire de files d'attente ; voir «Authentification du client MQTT à l'aide de SSL», à la page 105.
2. Connectez un client MQTT uniquement à un canal de télémétrie digne de confiance. Chiffrez les messages entre le client et le gestionnaire de files d'attente ; voir «Authentification du canal de télémétrie à l'aide de SSL», à la page 107.
3. Chiffrez les messages entre le client et le gestionnaire de files d'attente ; voir «Confidentialité de la publication sur les canaux de télémétrie», à la page 110.

Paramètres de configuration JSSE

Modifiez les paramètres JSSE afin de changer la façon dont une connexion SSL est configurée. Les paramètres de configuration JSSE sont composés de trois ensembles :

1. IBM WebSphere MQ Canal de télémétrie
2. client Java MQTT
3. JRE

Configurez les paramètres du canal de télémétrie à l'aide d'IBM WebSphere MQ Explorer. Définissez les paramètres du client Java MQTT dans l'attribut `MqttConnectionOptions.SSLProperties` . Modifiez les paramètres de sécurité en éditant les fichiers dans le répertoire de sécurité de JRE à la fois sur le client et sur le serveur.

IBM WebSphere MQ Canal de télémétrie

Définissez tous les paramètres SSL de canal de télémétrie à l'aide de WebSphere MQ Explorer.

ChannelName

ChannelName est un paramètre obligatoire sur tous les canaux.

Le nom de canal identifie le canal associé à un numéro de port particulier. Nommez les canaux de manière à vous aider à gérer des ensembles de clients MQTT.

PortNumber

PortNumber est un paramètre facultatif sur tous les canaux. La valeur par défaut est 1883 pour les canaux TCP et 8883 pour les canaux SSL.

Numéro de port TCP/IP associé à ce canal. Les clients MQTT sont connectés à un canal en indiquant le port défini pour le canal. Si le canal dispose des propriétés SSL, le client doit se connecter à l'aide du protocole SSL ; par exemple :

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

KeyFileName

KeyFileName est un paramètre obligatoire pour les canaux SSL. Il doit être omis pour les canaux TCP.

KeyFileName est le chemin d'accès au magasin de clés Java contenant les certificats numériques que vous fournissez. Sur le serveur, utilisez les types de magasin de clés JKS, JCEKS ou PKCS12.

Identifiez le type de magasin de clés à l'aide de l'une des extensions de fichier suivantes :

```
.jks
.jceks
.p12
.pkcs12
```

Un magasin de clés associé à une autre extension est censé être un magasin JKS.

Vous pouvez combiner un magasin de clés d'un type donné se trouvant sur le serveur avec d'autres types de magasins de clés se trouvant sur le client.

Placez le certificat privé du serveur dans le magasin de clés. Le certificat est connu sous le nom de certificat serveur. Ce certificat peut être autosigné ou faire partie d'une chaîne de certificats signée par une autorité de signature.

Si vous utilisez une chaîne de certificats, placez les certificats associés dans le magasin de clés du serveur.

Le certificat du serveur et tous les certificats de la chaîne de certificats sont envoyés aux clients afin d'authentifier l'identité du serveur.

Si vous avez attribué à ClientAuth la valeur Required, le magasin de clés doit contenir tous les certificats nécessaires à l'authentification du client. Le client envoie un certificat autosigné, ou une chaîne de certificats, et est authentifié par la première vérification de cet élément par rapport à un certificat se trouvant dans le magasin de clés. Lorsque vous utilisez une chaîne de certificats, un certificat peut vérifier plusieurs clients, même s'ils sont émis avec des certificats client différents.

PassPhrase

PassPhrase est un paramètre obligatoire pour les canaux SSL. Il doit être omis pour les canaux TCP.

La phrase passe est utilisée pour protéger le magasin de clés.

ClientAuth

`ClientAuth` est un paramètre SSL facultatif. Par défaut, aucune authentification de client n'est effectuée. Il doit être omis pour les canaux TCP.

Définissez `ClientAuth` si vous voulez que le service de télémétrie (MQXR) authentifie le client avant de lui permettre de se connecter au canal de télémétrie.

Si vous définissez `ClientAuth`, le client doit se connecter au serveur à l'aide de SSL et authentifier le serveur. En réponse à la définition de `ClientAuth`, le client envoie son certificat numérique au serveur et tout autre certificat dans son magasin de clés. Son certificat numérique est connu sous le nom de certificat client. Ces certificats sont authentifiés par rapport aux certificats se trouvant dans le magasin de clés du canal et dans le magasin `cacerts`.

CipherSuite

`CipherSuite` est un paramètre SSL facultatif. Par défaut, il essaie toutes les spécifications CipherSpecs activées. Il doit être omis pour les canaux TCP.

Si vous voulez utiliser une spécification CipherSpec particulière, attribuez à `CipherSuite` le nom de la spécification CipherSpec qui doit être utilisée pour établir la connexion SSL.

Le service de télémétrie et client MQTT négocient une spécification CipherSpec commune à partir de tous les CipherSpecs activés à chaque extrémité. Si un CipherSpec spécifique est indiqué à l'une ou aux deux extrémités, il doit correspondre au CipherSpec de l'autre extrémité.

Installez les chiffrements supplémentaires en ajoutant des fournisseurs supplémentaires à JSSE.

FIPS (Federal Information Processing Standards)

FIPS est un paramètre facultatif. Par défaut, il n'est pas défini.

Utilisez le panneau de propriété du gestionnaire de files d'attente ou **runmqsc**, pour définir SSLFIPS. SSLFIPS Indique si seuls les algorithmes certifiés pour FIPS doivent être utilisés.

Revocation namelist

Revocation namelist est un paramètre facultatif. Par défaut, il n'est pas défini.

Utilisez le panneau de propriété du gestionnaire de files d'attente ou **runmqsc**, pour définir SSLCRLNL. SSLCRLNL indique une liste de noms d'objet d'informations d'authentification utilisés pour fournir des emplacements de révocation de certificat.

Aucun autre gestionnaire de files d'attente qui a défini les propriétés SSL n'est utilisé.

Client Java MQTT

Définissez les propriétés SSL pour le client Java dans `MqttConnectionOptions.SSLProperties`; par exemple:

```
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);
```

Les noms et les valeurs des propriétés spécifiques sont décrits dans la documentation d'API pour `MqttConnectOptions`.Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

Protocole

`Protocol` est facultatif.

Le protocole est sélectionné en négociation avec le serveur de télémétrie. Si vous avez besoin d'un protocole spécifique, vous pouvez en sélectionner un. Si le serveur de télémétrie ne prend pas en charge le protocole, la connexion échoue.

ContextProvider

`ContextProvider` est facultatif.

KeyStore

KeyStore est facultatif. Configurez-le si ClientAuth est défini au niveau du serveur pour forcer l'authentification du client.

Placez le certificat numérique du client, signé à l'aide de la clé privée, dans le magasin de clés. Spécifiez le chemin d'accès et le mot de passe du magasin de clés. Le type et le fournisseur sont facultatifs. JKS est le type par défaut et IBMJCE est le fournisseur par défaut.

Indiquez un fournisseur de magasin de clés pour référencer une classe qui ajoute un nouveau fournisseur de magasin de clés. Transmettez le nom de l'algorithme utilisé par le fournisseur de magasin de clés pour instancier KeyManagerFactory en définissant le nom du gestionnaire de clés.

TrustStore

TrustStore est facultatif. Vous pouvez placer tous les certificats que vous accréditez dans le magasin cacerts de JRE.

Configurez le magasin de clés de confiance si vous souhaitez disposer d'un fichier de ce type distinct pour le client. Vous pouvez ne pas configurer le magasin de clés de confiance si le serveur utilise un certificat émis par une autorité de certification connue dont le certificat racine est déjà stocké dans cacerts.

Ajoutez le certificat signé publiquement du serveur ou du certificat racine au magasin de clés de confiance, et indiquez le chemin d'accès et le mot de passe du magasin de clés de confiance. JKS est le type par défaut et IBMJCE est le fournisseur par défaut.

Indiquez un fournisseur de magasin de clés de confiance pour référencer une classe qui ajoute un nouveau fournisseur de magasin de clés de confiance. Transmettez le nom de l'algorithme utilisé par le fournisseur de magasin de clés de confiance pour instancier TrustManagerFactory en définissant le nom du gestionnaire d'accréditation.

JRE

D'autres aspects de la sécurité Java qui affectent le comportement de SSL côté client et côté serveur sont configurés dans le JRE. Les fichiers de configuration sur Windows se trouvent dans *Java Installation Directory*\jre\lib\security. Si vous utilisez le JRE livré avec IBM WebSphere MQ, le chemin est celui qui est indiqué dans le tableau suivant :

Tableau 3. Chemins de fichiers par plateforme pour les fichiers de configuration SSL JRE

Plateforme	**Chemin d'accès au fichier**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux for System x 32 bits	*WMQ Installation Directory*/java/jre/lib/security
Autres plateformes UNIX and Linux	*WMQ Installation Directory*/java/jre64/jre/lib/security

Autorités de certification reconnues

Le fichier cacerts contient les certificats racine des autorités de certification reconnues. Le fichier cacerts est utilisé par défaut, sauf si vous définissez un autre magasin de clés de confiance. Si vous utilisez le fichier cacerts ou que vous n'indiquez aucun magasin de clés de confiance, vous devez vérifier et éditer la liste des signataires du fichier cacerts pour remplir les conditions requises en matière de sécurité.

Vous pouvez ouvrir le fichier cacerts à l'aide de la commande WebSphere MQ strmqikm qui exécutez l'utilitaire IBM Key Management. Ouvrez le fichier cacerts en tant que fichier JKS avec le mot de passe changeit. Modifiez le mot de passe afin de sécuriser le fichier.

Configuration des classes de sécurité

Utilisez le fichier `java.security` pour enregistrer les fournisseurs de sécurité supplémentaires et d'autres propriétés de sécurité par défaut.

Droits d'accès

Utilisez le fichier `java.policy` pour modifier les droits d'accès accordés aux ressources. `javaws.policy` accorde des droits à `javaws.jar`

Puissance de chiffrement

Certains JRE sont livrés avec une puissance de chiffrement réduite. Si vous ne pouvez pas importer les clés dans les magasins de clés, la réduction de la puissance de chiffrement peut en être la cause. Démarrez **ikeyman** à l'aide de la commande **strmqikm**, ou téléchargez des fichiers de juridiction de grande taille mais limités à partir du site IBM developer kits, Security information.

Important : Votre pays d'origine est peut-être assujetti à des restrictions relatives à l'importation, la possession, l'utilisation ou la réexportation de logiciel de chiffrement vers un autre pas. Avant de télécharger ou d'utiliser des fichiers de règles sans restriction, vous devez vérifier les lois applicables dans votre pays. Vérifiez ses règlements et sa politique en matière d'importation, de possession, d'utilisation et de réexportation de logiciel de chiffrement afin de déterminer si cette action est autorisée.

Modifiez le fournisseur d'accréditation pour permettre au client de se connecter à n'importe quel serveur.

L'exemple illustre comment ajouter un fournisseur d'accréditation et le référencer à partir du code du client MQTT. Dans l'exemple aucune authentification du client ou du serveur n'est effectuée. La connexion SSL résultante est chiffrée sans être authentifiée.

Le fragment de code dans la Figure 20, à la page 115 définit le fournisseur d'accréditation `AcceptAllProviders` et le gestionnaire d'accréditation pour le client MQTT.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

Figure 20. Fragment de code pour le client MQTT

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
                AcceptAllTrustManagerFactory.class.getName());
    }
```

Figure 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() };
        }
```

Figure 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);
  }
}
```

Figure 23. AcceptAllX509TrustManager.java

Configuration JAAS du canal de télémétrie

Configurez JAAS pour authentifier le Nom d'`utilisateur` envoyé par le client.

l'administrateur WebSphere MQ configure les canaux MQTT qui nécessitent l'authentification du client à l'aide de JAAS. Indiquez le nom d'une configuration JAAS pour chaque canal qui doit effectuer une authentification JAAS. Les canaux peuvent tous utiliser la même configuration JAAS configuration, ou utiliser des configurations JAAS différentes. Les configurations sont définies dans *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.config.

Le fichier `jaas.config` est organisé par nom de configuration JAAS. Sous chaque nom de configuration figure une liste de configurations de connexion ; voir Figure 24, à la page 117.

JAAS fournit quatre normes de modules de connexion. Les modules de connexion standard NT et UNIX ont une valeur limitée.

JndiLoginModule
Effectue l'authentification sur un service de répertoire configuré sous JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Effectue l'authentification à l'aide des protocoles Kerberos.

NTLoginModule
Effectue l'authentification à l'aide des informations de sécurité NT pour l'utilisateur en cours.

UnixLoginModule
Effectue l'authentification à l'aide des informations de sécurité UNIX pour l'utilisateur en cours.

Lorsque vous utilisez `NTLoginModule` ou `UnixLoginModule` le service de télémétrie (MQXR) s'exécute avec l'identité `mqm`, et non l'identité du canal MQTT. `mqm` est l'identité transmise à `NTLoginModule` ou `UnixLoginModule` pour l'authentification et non pas l'identité du client.

Pour résoudre ce problème, écrivez votre propre module de connexion ou utilisez d'autres modules de connexion standard. Un `JAASLoginModule.java` exemple est fourni avec WebSphere MQ Telemetry. Il s'agit d'une implémentation de l'interface `javax.security.auth.spi.LoginModule`. Utilisez-le pour développer votre propre méthode d'authentification.

Toute nouvelle classe LoginModule que vous fournissez doit figurer dans le chemin d'accès aux classes du service de télémétrie (MQXR). Ne placez pas vos classes dans les répertoires WebSphere MQ qui se trouvent dans le chemin d'accès aux classes. Créez vos propres répertoires et définissez le chemin d'accès complet aux classes pour le service de télémétrie (MQXR).

Vous pouvez accroître le chemin d'accès aux classes utilisé par le service de télémétrie (MQXR) en définissant le chemin d'accès aux classes dans le fichier `service.env`. CLASSPATH doit être indiqué en majuscules et peut uniquement contenir des littéraux. Vous ne pouvez pas utiliser de variables dans CLASSPATH ; par exemple, l'instruction CLASSPATH=%CLASSPATH% est incorrecte. Le service de télémétrie (MQXR) définit son propre chemin d'accès aux classes? La variable CLASSPATH définie dans le fichier `service.env` est ajoutée à ce dernier.

Le service de télémétrie (MQXR) fournit deux rappels qui renvoient le `nom d'utilisateur` et le `Mot de passe` pour le client connecté au canal MQTT. Le `nom d'utilisateur` et le `mot de passe` sont définis dans l'objet `MqttConnectOptions` . Voir la Figure 25, à la page 117 pour un exemple d'accès au `Nom d'utilisateur` et au `Mot de passe`.

Exemples

Modèle de fichier de configuration JAAS avec une configuration nommée, `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";
};
```

Figure 24. Modèle de fichier `jaas.config`

Modèle de module de connexion JAAS codé pour recevoir le `Nom d'utilisateur` et le `Mot de passe` fournis par un client MQTT.

```
  public boolean login()
      throws javax.security.auth.login.LoginException {
    javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
    callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
    callbacks[1] = new javax.security.auth.callback.PasswordCallback(
        "PasswordCallback", false);
    try {
      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // Accept everything.
      if (true) {
        loggedIn = true;
      } else
        throw new javax.security.auth.login.FailedLoginException("Login failed");

      principal= new JAASPrincipal(username);

    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }

    return loggedIn;
  }
```

Figure 25. Modèle de méthode `JAASLoginModule.Login()`

Concepts de programmation du client

Les concepts décrits dans cette section vous aideront à comprendre le client Java pour la version 3.1 du MQTT protocol. Les concepts viennent en complément de la documentation d'API accompagnant le package `com.ibm.micro.client.mqttv3`.

`com.ibm.micro.client.mqttv3` contient les classes qui fournissent les méthodes publiques pour les implémentations Java du protocole MQTT version 3.1 . Le package `com.ibm.micro.client.mqttv3` , ainsi que les packages qui l'accompagnent et qui implémentent le protocole pour Java SE et ME, sont fournis avec l'installation de IBM WebSphere MQ Telemetry.

Pour développer et exécuter un client MQTT, vous devez copier ou installer ces packages sur l'appareil client. Il n'est pas nécessaire d'installer un environnement d'exécution client distinct.

Les conditions de la licence applicables aux clients sont associées au serveur auquel ils sont connectés.

Le client Java est une implémentation de référence de la version 3.1 du MQTT protocol. Vous pouvez implémenter vos propres clients dans différents langages adaptés à différentes plateformes de dispositif. Pour plus de détails, reportez-vous à la rubrique Format et protocole MQ Telemetry Transport.

La documentation des API client du package `com.ibm.micro.client.mqttv3` ne précise pas le serveur auquel le client est connecté. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference. Le comportement du client peut varier légèrement en fonction des serveurs. Les descriptions qui suivent concernent le comportement du client connecté au service de télémétrie (MQXR) de IBM WebSphere MQ.

Le client de messagerie MQTT pour JavaScript et les apps Web

Jusqu'à récemment, la programmation des applications Web et la création d'applications de messagerie étaient des disciplines distinctes. Quelle que soit votre expérience passée, l'utilisation conjointe de JavaScript et de la messagerie présente des avantages non négligeables. Votre application de messagerie, codée comme une application Web, peut être importée et exécutée dans tous les navigateurs récents. Lorsque vous la modifiez, sa dernière version est importée lorsque le navigateur est actualisé. Le navigateur s'occupe de la sécurité et de la fiabilité de la transmission des messages.

Comment l'utilisation des applications Web facilite-t-elle le déploiement des applications ?

Si vous avez de l'expérience dans le développement et le déploiement des apps de messagerie traditionnelles sur (par exemple) IBM WebSphere MQ, vous êtes peut-être familiarisé avec le processus de déploiement suivant :

1. L'administrateur du système installe ou incorpore la bibliothèque client.
2. L'administrateur du système s'arrange pour que l'application de messagerie soit distribuée aux utilisateurs et installé sur leurs systèmes locaux.
3. Lorsque le code change, l'administrateur du système répète les étapes précédentes (la gestion des changements est donc complexe).

Si vous codez votre application de messagerie comme une application Web, le processus de déploiement est le suivant :

1. L'administrateur du système donne accès à l'application Web et à la bibliothèque client par une URL.
2. Le navigateur de l'utilisateur importe en même temps l'application Web et la bibliothèque client.
3. Lorsque le code change, la version mise à jour est importée lorsque le navigateur est actualisé (la gestion des changements est donc simple).

Pourquoi utiliser directement la messagerie du navigateur dans les applications Web ?

Les programmeurs expérimentés d'apps JavaScript peuvent être intéressés par les avantages fournis par les systèmes de messagerie tels que IBM WebSphere MQ :

- Le système de messagerie est responsable de la distribution effective des messages qui transitent par lui.
- Puisque le système de messagerie s'occupe de la distribution, l'application Web peut déclencher la diffusion et la considérer comme autonome. La logique de programmation en est grandement simplifiée. Dans la mesure où la distribution des messages est assurée, votre code n'a plus besoin de la surveiller. L'application n'a plus à gérer les accusés de réception, ni à enregistrer les messages non distribués pour faire une nouvelle tentative de distribution ultérieurement.
- Les systèmes de messagerie sont pilotés par les événements. Après l'envoi d'une demande, il n'est donc plus nécessaire pour l'application client d'interroger continuellement le système pour savoir s'il y a une réponse. A la place, le serveur de messagerie envoie un message au client lorsqu'un événement intéressant se produit. Cela signifie également que l'application client est alertée dès que l'événement se produit, et n'attend pas d'interroger le serveur.
- En outre, la messagerie pilotée par les événements réduit de manière significative la charge sur le périphérique qui héberge votre application client, le trafic sur le réseau entre le navigateur et le serveur de messagerie, et la charge sur ce dernier. Ce point est d'une importance toujours accrue, car un nombre sans cesse plus important de systèmes s'exécutent sur des appareils mobiles et se connectent sur les réseaux sans fil.

Comment les composants s'emboîtent-ils ?

Le client de messagerie MQTT pour JavaScript contient une bibliothèque client et un modèle d'application Web qui l'utilise. Vous pouvez coder vous-même une application Web qui utilise cette bibliothèque. L'application Web et la bibliothèque client sont alors mises à disposition sur une URL de votre choix, par exemple par un gestionnaire de files d'attente MQ (comme dans le diagramme suivant), ou par un serveur d'applications. Le navigateur importe l'app Web et la bibliothèque client, puis l'app Web utilise le navigateur pour se connecter à un serveur MQTT, par exemple IBM WebSphere MQ Telemetry ou IBM MessageSight, et échanger des messages avec lui.

Les flux sont les suivants :

1. Chaque instance du navigateur actualise sa connexion à l'URL sur laquelle l'application Web est disponible, et une version à jour de l'application Web et de la bibliothèque client est chargée dans le navigateur.
2. L'app Web se connecte à un gestionnaire de files d'attente, en utilisant MQTT sur le protocole WebSocket protocol, et s'abonne à un sujet qui la concerne.
3. Le gestionnaire de files d'attente utilise la même connexion pour renvoyer à l'application Web des messages qui correspondent à l'abonnement.

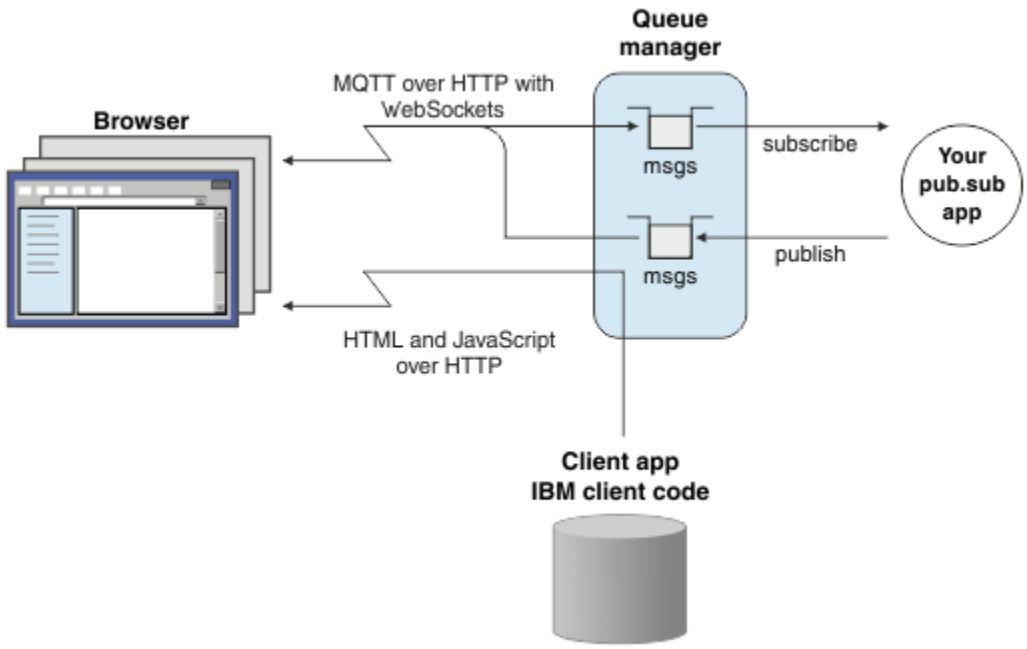


Figure 26. Utilisation du client de messagerie MQTT pour JavaScript avec la messagerie de type publication/abonnement

L'application Web contient la logique de l'application et l'URL du serveur MQTT. Lorsqu'elle est ouverte dans un navigateur, l'application se connecte au serveur MQTT, crée les abonnements dont elle a besoin, puis attend de recevoir des alertes déclenchées par les événements pour réagir.

L'app Web se connecte à l'aide du protocole de transport MQTT sur WebSockets. La plupart des navigateurs modernes sont capables d'établir des connexions WebSockets. En utilisant WebSockets, l'app Web peut transmettre des messages au travers de pare-feu qui acceptent HTTP et WebSocket protocol, et envoyer des paquets de données (aussi appelés "trames") comme si elle utilisait TCP sur IP.

Lorsqu'un message envoyé par l'application Web arrive dans le serveur MQTT, l'application côté serveur le voit simplement comme un message. Il ne sait pas que le message provient d'un navigateur.

Administration et contrôle d'un serveur MQTT

Le serveur MQTT gère la complexité de la messagerie du côté du serveur. Il garantit la distribution des messages qu'il reçoit de l'application Web, et héberge l'application de publication/abonnement qui répond à celle-ci. La procédure qui suit s'applique à tous les serveurs MQTT :

- Créer un serveur.
- Choisissez un port.
- Définissez un nouveau canal MQTT.
- Configurer l'application Web pour qu'elle se connecte au port choisi sur le nouveau canal MQTT.

Vous devez aussi fournir le code JavaScript exécutable au navigateur. Si vous utilisez IBM WebSphere MQ Telemetry, le serveur MQTT le fait lui-même par défaut, en utilisant le canal MQTT déjà utilisé par l'app Web pour se connecter au serveur MQTT. Si vous testez MQTT, cela peut être un moyen rapide de mettre en oeuvre votre solution. En revanche, dans un environnement de production, particulièrement su le débit est important, vous préférerez peut-être servir le JavaScript exécutable de l'app Web sur un canal distinct, à l'aide d'un serveur d'applications dédié tel que WebSphere Application Server.

Remarque : Parce qu'il est conçu pour les environnements à haut débit, IBM MessageSight s'attend à cette configuration.

Par exemple, si vous utilisez IBM WebSphere MQ Telemetry, vous utilisez l'assistant **Nouveau canal de télémétrie** de IBM WebSphere MQ Explorer pour réaliser la procédure suivante :

1. Créer un serveur.
2. Choisir un port (1883 par défaut).
3. Définissez un nouveau canal MQTT.
4. Configurer l'application Web pour qu'elle se connecte au port choisi sur le nouveau canal MQTT.

Le code JavaScript exécutable de l'app Web est (facultativement) servi par le gestionnaire de files d'attente sur le même canal. Ceci n'est possible que si le gestionnaire de files d'attente prend en charge MQTT et HTTP. Si vous disposez déjà d'un gestionnaire de files d'attente configuré pour MQTT, vous pouvez utiliser l'outil de ligne de commande MQSC pour modifier le protocole dans la définition du canal et prendre en charge à la fois MQTT et HTTP. Voir ALTER CHANNEL.

L'app Web et la bibliothèque client du client de messagerie MQTT pour JavaScript sont stockées sur le disque dans une structure qui est définie par votre serveur d'applications ou votre gestionnaire de files d'attente. Si vous utilisez IBM WebSphere MQ Telemetry, l'app Web et la bibliothèque client sont stockées dans la structure de répertoires suivante :

```
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)
```

Le modèle d'application Web et la bibliothèque client sont stockés dans le répertoire *MQINSTALL*/mqxr/SDK/WebContent . Les éléments de ce répertoire sont servis par tous les gestionnaires de files d'attente. Si vous ne souhaitez pas que les utilisateurs puissent voir et utiliser l'ensemble de ces éléments, vous devez créer votre propre version de l'application. Pour rendre cette application, ou votre propre application de remplacement, disponible sur des gestionnaires de files d'attente spécifiques, vous placez l'application dans le répertoire *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent . Pour sélectionner l'application et les classes JavaScript associées à servir à une URL, le gestionnaire de files d'attente doit d'abord se trouver dans son propre répertoire WebContent , puis dans le répertoire WebContent global. Dans l'arborescence de répertoires de l'exemple précédent, le gestionnaire de files d'attente sert *votre app client* et la copie générale des classes JavaScript.

Pour empêcher le gestionnaire de files d'attente de servir les fichiers exécutables de l'application Web, ou modifier le répertoire dans lequel il les recherche, configurez la propriété **webcontentpath** et ajoutez-la au fichier mqxr.properties. Voir MQXR properties.

Concepts associés

«Comment programmer des apps de messagerie dans JavaScript», à la page 122

Tâches associées

«Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

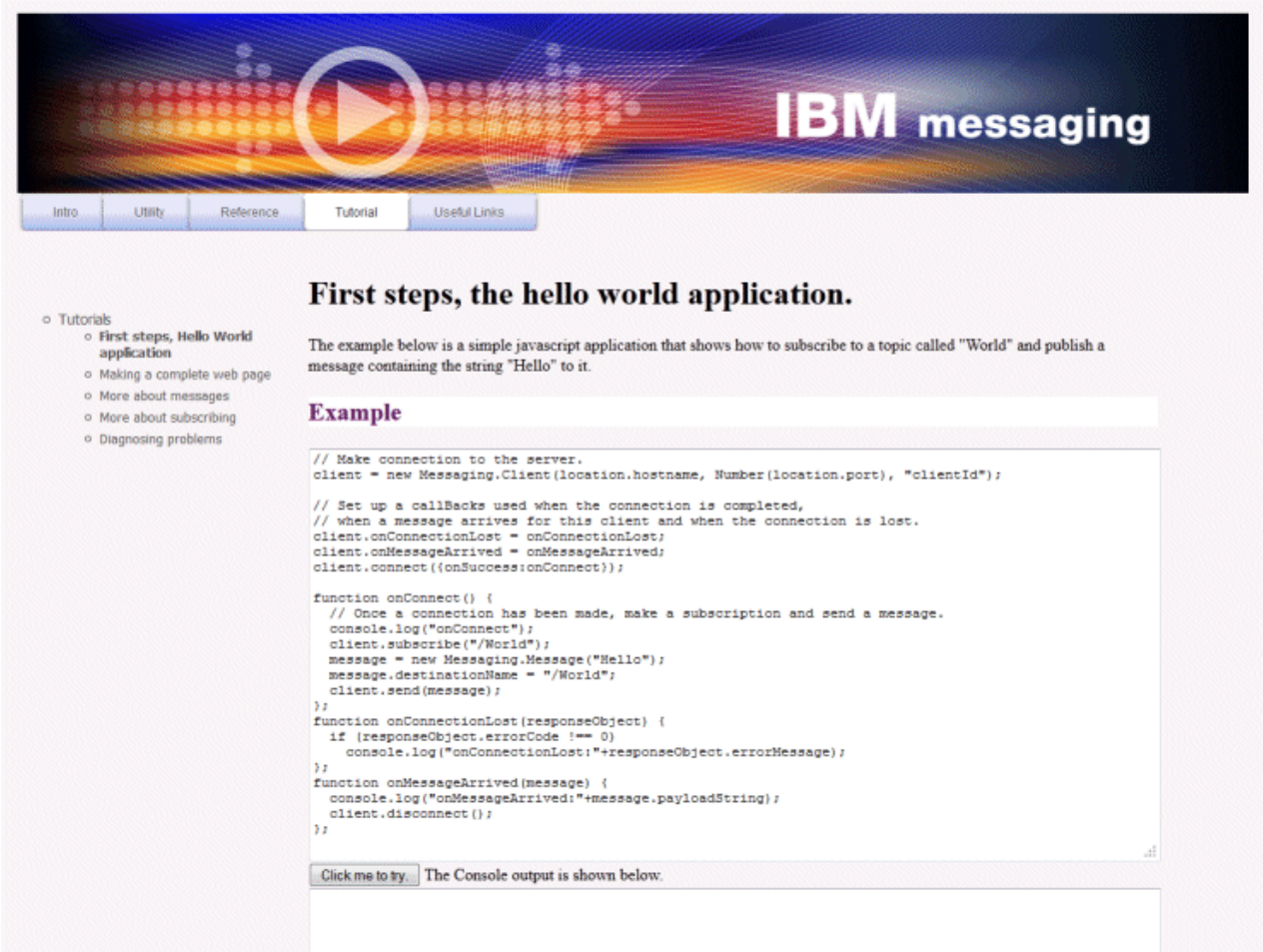
«Initiation au client de messagerie MQTT pour JavaScript», à la page 24

Vous pouvez vous familiariser avec le client de messagerie MQTT pour JavaScript en affichant la page d'accueil du modèle de client de messagerie, et en navigant dans les ressources accessibles par les liens. Pour afficher cette page d'accueil, configurez un serveur MQTT de manière qu'il accepte les connexions du Pages de l'exemple de client de messagerie MQTT JavaScript, puis entrez l'URL que vous avez configurée sur le serveur dans un navigateur Web. Le client de messagerie MQTT pour JavaScript démarre automatiquement sur votre périphérique, et la page d'accueil du modèle de client de messagerie s'affiche. Cette page contient des liens vers des utilitaires, la documentation de l'API, un tutoriel et d'autres informations utiles.

Comment programmer des apps de messagerie dans JavaScript

Le client de messagerie MQTT pour JavaScript contient un tutoriel qui montre comment créer simplement une app Web de publication/abonnement. En analysant le code de l'application "First steps, Hello world", vous pouvez appréhender le mécanisme de la programmation des applications Web de messagerie.

Par ailleurs, si vous avez jusqu'à présent principalement développé et déployé des applications de messagerie traditionnelles, la section «Astuces de programmation JavaScript», à la page 122 peut vous être utile. Si vous être un développeur JavaScript expérimenté, et si vous ne connaissez pas encore la messagerie, vous pouvez consulter une brève introduction à ses principaux concepts dans la section «Concepts de base de la messagerie», à la page 124.



Astuces de programmation JavaScript

Si vous avez l'habitude de développer des applications de messagerie, mais si les applications Web sont une nouveauté pour vous, les astuces suivantes peuvent vous rendre service :

Encapsulation du code de chaque événement dans un rappel onSuccess

Lorsque vous codez une application de messagerie, vous codez les événements suivants dans cet ordre :

1. connect
2. s'abonner
3. Publication
4. Réception des messages

L'API du client de messagerie MQTT pour JavaScript est complètement asynchrone, ce qui signifie que l'unité d'exécution de l'application ne reste pas bloquée en attendant l'aboutissement des appels du type connexion ou abonnement. A la place, ces appels signalent qu'ils ont abouti en appelant un rappel `onSuccess` ou `onFailure`. Pour être certain que chaque événement est terminé avant le déclenchement de l'événement suivant, vous devez encapsuler son code dans un rappel `onSuccess`. Par exemple, l'application JavaScript peut avoir terminé son appel de connexion avant la création effective de celle-ci. Pour être sûr que la connexion est établie avant la création de l'abonnement, vous devez insérer le code d'abonnement dans un rappel `onSuccess` de la connexion.

Le code de l'application "First steps, Hello world" utilise cette méthode.

Imbrication du code de l'application dans des balises HTML

Voici un exemple de page JavaScript :

Voici la source de la page précédente, qui montre comment le code de l'application est imbriqué dans des balises 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>
```

Concepts de base de la messagerie

Voici quelques informations de base pour les développeurs d'applications Web qui débutent dans la programmation d'applications de messagerie :

Messagerie asynchrone et autonome après diffusion

Le protocole MQTT garantit la distribution des messages et des transferts "autonomes après diffusion". Dans le protocole, la distribution des messages est asynchrone : l'application transmet le message à l'API client, et ne fait rien de plus pour s'assurer que le message est distribué. On parle alors d'un transfert *autonome après diffusion*. Les réponses disponibles sont automatiquement envoyées à l'app.

La distribution asynchrone libère l'application des connexions serveur et de l'attente des messages. Le modèle d'interaction est semblable à celui de la messagerie électronique, mais il est optimisé pour la programmation d'application.

Voir aussi la section "Protocole MQTT" de la rubrique «Introduction à MQTT», à la page 5.

Présentation de la messagerie de type publication/abonnement.

Le fournisseur des informations est appelé le *diffuseur de publications*. Le diffuseur de publications fournit des informations sur une rubrique, sans avoir de connaître les applications qu'elles intéressent. Il choisit une *rubrique*, qui est le conteneur des messages relatifs à un thème spécifique. Puis il génère chaque information sur ce thème sous la forme d'un message, appelé une *publication*, et l'envoie sous la rubrique associée.

Le consommateur des informations est l'*abonné*. L'abonné crée un *abonnement* à une rubrique qui l'intéresse. Les nouveaux messages de la rubrique sont envoyés à tous ses abonnés. Les abonnés peuvent souscrire différents abonnements et recevoir des informations de différents diffuseurs de publications.

Voir aussi Introduction à la messagerie de publication / abonnement IBM WebSphere MQ

Correspondance entre les abonnements et les sujets.

Si IBM WebSphere MQ est votre serveur MQTT, vous devez comprendre la manière dont IBM WebSphere MQ spécifie les rubriques. Dans IBM WebSphere MQ, un diffuseur de publications crée un message et le publie avec la chaîne de rubrique qui convient le mieux au thème de la publication. Pour recevoir des publications, un abonné crée un abonnement avec un schéma correspondant à la chaîne de rubrique pour sélectionner les rubriques de publication. Le gestionnaire de files d'attente distribue les publications aux abonnés dont les abonnements correspondent à la rubrique de publication, et qui sont autorisés à recevoir les publications.

Généralement, les rubriques sont classés dans un ordre hiérarchique, dans des arborescences, avec le caractère '/' pour créer des sous-rubriques dans la chaîne de rubrique. Les rubriques sont des noeuds de l'arborescence. Les rubriques sont des noeuds qui ne comportent aucune sous-rubrique, ou des noeuds intermédiaires avec des sous-rubriques. Les abonnés peuvent utiliser des caractères génériques pour s'abonner à plusieurs rubriques en même temps. Par exemple, l'abonnement à `/sport/tennis` ne concerne que les messages de la sous-rubrique tennis, tandis que l'abonnement à `/sport/#` concerne ceux de la sous-rubrique `/sport`.

Voir aussi Rubriques, Arborescence de rubriques et Schémas de caractères génériques.

Concepts associés

«Le client de messagerie MQTT pour JavaScript et les apps Web», à la page 118

Tâches associées

«Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

«Initiation au client de messagerie MQTT pour JavaScript», à la page 24
Vous pouvez vous familiariser avec le client de messagerie MQTT pour JavaScript en affichant la page d'accueil du modèle de client de messagerie, et en navigant dans les ressources accessibles par les liens. Pour afficher cette page d'accueil, configurez un serveur MQTT de manière qu'il accepte les connexions du Pages de l'exemple de client de messagerie MQTT JavaScript, puis entrez l'URL que vous avez configurée sur le serveur dans un navigateur Web. Le client de messagerie MQTT pour JavaScript démarre automatiquement sur votre périphérique, et la page d'accueil du modèle de client de messagerie s'affiche. Cette page contient des liens vers des utilitaires, la documentation de l'API, un tutoriel et d'autres informations utiles.

Rappels et synchronisation dans les applications client MQTT

Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , autant qu'elles le peuvent, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Rappels

L'interface `MqttCallback` est dotée de trois méthodes de rappel ; voir un message d'implémentation dans Callback.java.

`connectionLost(java.lang.Throwable cause)`

`connectionLost` est appelé lorsqu'une erreur de communication provoque la suppression de la connexion. Cette méthode est également appelée si le serveur supprime la connexion à la suite d'une erreur liée au serveur après l'établissement de la connexion. Les erreurs de serveur sont consignées dans le journal des erreurs du gestionnaire de files d'attente. Le serveur supprime la connexion vers le client, et le client appelle `MqttCallback.connectionLost`.

Les seules erreurs distantes émises en tant qu'exceptions sur la même unité d'exécution que l'application client sont les exceptions provenant de `MqttClient.connect`. Les erreurs détectées par le serveur une fois la connexion établie sont signalées à la méthode de rappel `MqttCallback.connectionLost` en tant que `throwables`.

Les erreurs de serveur classiques résultant de `connectionLost` sont des erreurs d'autorisation. Par exemple, le serveur de télémétrie tente de diffuser une publication sur une rubrique au nom d'un client qui n'est pas autorisé à diffuser de publication sur cette rubrique. Toute action entraînant le renvoi d'un code de condition `MQCC_FAIL` au serveur de télémétrie peut provoquer la suppression de la connexion.

`deliveryComplete(MqttDeliveryToken token)`

`deliveryComplete` est appelé par le client MQTT pour renvoyer un jeton de distribution à l'application client ; voir «Jetons de distribution», à la page 130. Lorsque vous utilisez un jeton de distribution, la fonction de rappel peut accéder au message publié avec la méthode `token.getMessage`.

Lorsque le rappel d'application renvoie le contrôle au client MQTT après avoir été appelé par la méthode `deliveryComplete` , la distribution est terminée. Jusqu'à l'achèvement de la distribution, les messages avec la qualité de service QoS 1 ou 2 sont conservés dans la classe de persistance.

L'appel à `deliveryComplete` est un point de synchronisation entre l'application et la classe de persistance. La méthode `deliveryComplete` n'est jamais appelée deux fois pour le même message.

Lorsque le rappel d'application est renvoyé de `deliveryComplete` au client MQTT , le client appelle `MqttClientPersistence.remove` pour les messages avec QoS 1 ou 2. `MqttClientPersistence.remove` supprime la copie stockée en local du message publié.

D'un point de vue du traitement de transaction, l'appel de `deliveryComplete` est une transaction à phase unique qui valide la distribution. Si le traitement échoue au cours du rappel, lors du redémarrage du client, `MqttClientPersistence.remove` est rappelé pour supprimer la copie locale du message publié. Il n'est pas fait appel une nouvelle fois à la procédure de rappel. Si vous utilisez le rappel pour stocker un journal des messages distribués, vous ne pouvez pas synchroniser le journal avec le client MQTT . Si vous voulez stocker un journal de façon fiable, mettez à jour le journal dans la classe `MqttClientPersistence`.

Le jeton de distribution et le message sont référencés par l'unité d'exécution principale de l'application et le client MQTT. Le client MQTT déréférence l'objet `MqttMessage` lorsque la distribution est terminée et l'objet de jeton de distribution lorsque le client se déconnecte. L'objet `MqttMessage` peut faire l'objet d'un nettoyage de mémoire après la fin de la distribution si l'application client le déréférence. Le jeton de distribution peut faire l'objet d'un nettoyage de mémoire une fois la session déconnectée.

Vous pouvez obtenir des attributs `MqttDeliveryToken` et `MqttMessage` une fois que le message a été publié. Si vous tentez de définir des attributs `MqttMessage` après la publication du message, le résultat est indéfini.

Le client MQTT continue à traiter les accusés de réception de la distribution si le client se reconnecte à la session précédente avec le même `identificateur client`. Voir «Sessions propres», à la page 128. L'application client MQTT doit définir `MqttClient.CleanSession` sur `false` pour la session précédente et sur `false` dans la nouvelle session. Le client MQTT crée de nouveaux jetons de distribution et les objets message dans la nouvelle session pour les distributions en attente. Il récupère les objets à l'aide de la classe `MqttClientPersistence`. Si le client d'application fait encore référence aux anciens jetons de distribution et messages, supprimez ces références. La fonction de rappel de l'application est appelée dans la nouvelle session pour toute livraison initiée dans la session précédente et terminé dans cette session.

La fonction de rappel de l'application est appelée après la connexion du client d'application lorsqu'une distribution en attente est terminée. Avant la connexion du client d'application, il peut extraire les distributions en attente à l'aide de la méthode `MqttClient.getPendingDeliveryTokens`.

Remarquez que l'application client a créé au départ l'objet message publié, et le tableau d'octets qui constitue sa charge. Le client MQTT fait référence à ces objets. L'objet message renvoyé par le jeton de distribution dans la méthode `token.getMessage` n'est pas nécessairement le même objet message créé par le client. Si une nouvelle instance de client MQTT recrée le jeton de distribution, la classe `MqttClientPersistence` recrée l'objet `MqttMessage` . Pour plus de cohérence, `token.getMessage` renvoie la valeur `null` si `token.isCompleted` a la valeur `true`, indépendamment du fait que l'objet message a été créé par le client d'application ou par la classe `MqttClientPersistence`.

`messageArrived(MqttTopic topic, MqttMessage message)`

`messageArrived` est appelé lorsqu'une publication arrive pour le client qui correspond à une rubrique d'abonnement. `topic` est la rubrique de publication, et non le filtre d'abonnement. Ces deux éléments peuvent être différents si le filtre contient des caractères génériques.

Si la rubrique correspond à plusieurs abonnements créés par le client, ce dernier reçoit plusieurs copies de la publication. Si un client diffuse une publication sur une rubrique à laquelle il est également abonné, il reçoit une copie de sa propre publication.

Si le message est envoyé avec un QoS de 1 ou 2, il est stocké par la classe `MqttClientPersistence` avant que le client MQTT n'appelle `messageArrived`. `messageArrived` se comporte comme `deliveryComplete` : il est appelé une seule fois pour une publication, et la copie locale de la publication est retirée par `MqttClientPersistence.remove` lorsque `messageArrived` retourne au client MQTT. Le client MQTT supprime ses références à la rubrique et au message lorsque `messageArrived` revient au client MQTT . Les rubriques et les objets message font l'objet d'un nettoyage de mémoire, si le client d'application n'a pas placé une référence sur les objets.

Rappels, unités d'exécution et synchronisation des applications client

Le client MQTT appelle une méthode de rappel sur une unité d'exécution distincte de l'unité d'exécution de l'application principale. L'application client ne crée pas d'unité d'exécution pour le rappel, elle est créée par le client MQTT .

Le client MQTT synchronise les méthodes de rappel. Seule une instance de méthode de rappel peut s'exécuter à la fois. La synchronisation facilite la mise à jour des objets qui pointent les publications qui ont été distribuées. Une seule instance de `MqttCallback.deliveryComplete` s'exécute à la fois, ce qui permet de rendre plus sûre la mise à jour du pointage sans effectuer de synchronisation supplémentaire. Il se peut également qu'une seule publication arrive à la fois. Votre code se trouvant dans la méthode `messageArrived` peut mettre à jour un objet sans le synchroniser. Si vous faites référence au pointage ou que l'objet est en cours de mise à jour, dans une autre unité d'exécution synchronisez le pointage ou l'objet.

Le jeton de distribution fournit un mécanisme de synchronisation entre l'unité d'exécution de l'application principale et la distribution d'une publication. La méthode `token.waitForCompletion` attend la fin

de la distribution d'une publication spécifique ou l'expiration d'un délai d'attente facultatif. Vous pouvez utiliser `token.waitForCompletion` de plusieurs manières différentes afin de traiter une publication à la fois :

1. Pour mettre en pause l'application client jusqu'à la fin de la distribution de la publication, voir PubSync.java.
2. Pour réaliser la synchronisation avec la méthode `MqttCallback.deliveryComplete`. `token.waitForCompletion` ne reprend que lorsque `MqttCallback.deliveryComplete` revient au client MQTT. L'utilisation de ce mécanisme vous permet de synchroniser le code s'exécutant dans `MqttCallback.deliveryComplete` avant que le code ne s'exécute dans l'unité d'exécution de l'application principale.

Que se passe-t-il si vous voulez diffuser une publication sans attendre que chaque publication soit distribuée, mais que vous voulez obtenir la confirmation que toutes les publications ont été distribuées ? Si vous diffusez une publication sur une seule unité d'exécution, la dernière publication à être envoyée est également la dernière à être distribuée.

Synchronisation des demandes envoyées au serveur

Tableau 4. Comportement de synchronisation de méthodes entraînant l'envoi de demandes au serveur.

Ce tableau répertorie les méthodes du client Java MQTT qui envoient une demande au serveur. Pour chaque méthode, le tableau décrit les conditions dans lesquelles la méthode renvoie une valeur ou attend, ainsi que la durée d'attente.

Méthode	Synchronisation	Intervalle de délai d'attente
`MqttClient.Connect`	Attend qu'une connexion soit établie avec le serveur.	30 secondes par défaut, ou valeur définie par un paramètre.
`MqttClient.Disconnect`	Attend que le client MQTT termine le travail qu'il doit effectuer et que la session TCP/IP se déconnecte.	30 secondes par défaut, ou valeur définie par un paramètre.
`MqttClient.Subscribe`	Attend la fin de la demande d'abonnement.	30 secondes par défaut, ou valeur définie par un paramètre.
`MqttClient.UnSubscribe`	Attend la fin de la demande de désabonnement.	30 secondes par défaut, ou valeur définie par un paramètre.
`MqttClient.Publish`	Revient immédiatement à l'unité d'exécution d'application de l'application après avoir transmis la requête au client MQTT.	Aucune.
`MqttDeliveryToken.waitForCompletion`	Attend le retour du jeton de distribution.	Valeur indéfinie par défaut, ou valeur définie par un paramètre.

Sessions propres

Le client MQTT et le service de télémétrie (MQXR) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Lorsque vous connectez une application client MQTT à l'aide de la méthode `MqttClient.connect`, le client identifie la connexion à l'aide de l'identificateur client et de l'adresse du serveur. Le serveur vérifie si les informations de session ont été enregistrées à partir de la précédente connexion au serveur. Si une session précédente existe, et que `cleanSession=true`, alors les précédentes informations de session côté client et côté serveur sont supprimées. Si `cleanSession=false` la session précédente est reprise. Si aucune session précédente n'existe, une nouvelle session est démarrée.

Remarque : L'administrateur WebSphere MQ peut fermer une session ouverte et supprimer toutes les informations de session par la force. Si le client ouvre à nouveau la session avec `cleanSession=false`, une nouvelle session est démarrée.

Publications

Si vous utilisez l'objet par défaut `MqttConnectOptions`, ou que vous attribuez à `MqttConnectOptions.cleanSession` la valeur `true` avant de connecter le client, toutes les distributions de publication en attente pour le client sont supprimées lorsque le client se connecte.

Le paramètre de session propre n'a aucun impact sur les publications envoyées avec la qualité de service `QoS=0`. Pour `QoS=1` et `QoS=2`, l'utilisation de `cleanSession=true` peut entraîner la perte d'une publication.

Abonnements

Si vous utilisez la valeur par défaut `MqttConnectOptions`ou que vous définissez `MqttConnectOptions.cleanSession` sur `true` avant de connecter le client, les anciens abonnements du client sont supprimés lorsque le client se connecte. Les nouveaux abonnements effectués par le client lors de la session sont supprimés lorsque celui-ci se déconnecte.

Si vous définissez `MqttConnectOptions.cleanSession` sur `false` avant de se connecter, tous les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour le client avant sa connexion. Tous les abonnements restent actifs au moment de la déconnexion du client.

Un autre moyen de comprendre l'impact de l'attribut `cleanSession` sur les abonnements consiste à le considérer comme un attribut modal. Dans le cadre de son mode par défaut `cleanSession=true`, le client crée des abonnements et reçoit des publications uniquement dans la portée de la session. En mode alternatif, `cleanSession=false`, les abonnements sont durables. Le client peut se connecter et se déconnecter, et ses abonnements restent actifs. Lorsque le client se reconnecte, il reçoit les publications non distribuées. Lors de la connexion, il peut modifier l'ensemble des abonnements qui sont actifs en son nom.

Vous devez définir le mode `cleanSession` avant de vous connecter ; le mode dure toute la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez de l'utilisation de `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimés.

Identificateur de client

L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . L'identificateur du client doit être unique parmi tous les clients qui se connectent au serveur, et ne doit pas être identique au nom du gestionnaire de files d'attente sur le serveur. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

L'identificateur de client est utilisé dans l'administration d'un système MQTT . Avec potentiellement des centaines de milliers de clients à administrer, vous devez pouvoir identifier rapidement un client particulier. Supposez, par exemple, qu'un dispositif est défaillant et que vous en êtes averti par un client qui appelle le centre d'assistance. Comment le client procède-t-il pour identifier le dispositif, et comment corréler cette identification avec le serveur qui est généralement connecté au client ? Devez-vous consulter une base de données qui mappe chaque dispositif à un identificateur client et à un serveur ? Le nom du dispositif permet-il d'identifier le serveur auquel ce dispositif est connecté ? Lorsque

vous parcourez les connexions client MQTT , chaque connexion est libellée avec l'identificateur client. Devez-vous consulter un tableau pour mapper un identificateur client à un dispositif physique ?

L'identificateur client identifie-t-il un dispositif particulier ou une application s'exécutant sur ce client ? Si un client remplace un dispositif défectueux par un nouveau, le nouveau dispositif possède-t-il le même identificateur client que l'ancien ? Allez vous allouer un nouvel identificateur ? Si vous changez un dispositif physique, mais que vous gardez le même identificateur, les publications en suspens et les publications actives sont automatiquement transférées vers le nouveau dispositif.

Comment vous assurer que les identificateurs client sont uniques ? De même qu'un système génère des identificateurs uniques, vous devez faire appel à un processus fiable pour définir l'identificateur sur le client. Le dispositif client est peut-être une "boîte noire", sans aucune interface utilisateur. Est-ce que vous fabriquez le dispositif avec un identificateur client - tel que l'utilisation de l'adresse MAC ? Ou disposez-vous d'une installation logicielle et d'un processus de configuration qui configure le dispositif avant son activation ?

Vous pouvez créer un identificateur client à partir d'une adresse MAC du dispositif 48 bits, pour que l'identificateur soit court et unique. Si la taille de n'est pas une question critique, vous pouvez utiliser les 17 octets restants afin de faciliter la gestion de l'adresse.

Jetons de distribution

Lorsqu'un client diffuse une publication sur un sujet, un jeton de distribution est créé. Utilisez le jeton de distribution pour surveiller la distribution d'une publication ou pour bloquer l'application client jusqu'à ce que la distribution soit terminée.

Le jeton est un objet `MqttDeliveryToken`. Il est créé en appelant la méthode MqttTopic.publish() et conservé par le client MQTT jusqu'à ce que la session client soit déconnectée et que la distribution soit terminée.

L'utilisation normale d'un jeton consiste à vérifier si la distribution a été effectuée. Bloquez l'app client jusqu'à l'achèvement de la distribution en utilisant le jeton renvoyé pour appeler `token.waitForCompletion`. Vous pouvez également indiquer un gestionnaire `MqttCallBack`. Lorsque le client MQTT a reçu tous les accusés réception, qu'il attend dans le cadre de la distribution de la publication, il appelle `MqttCallBack.deliveryComplete` en envoyant le jeton de distribution sous la forme d'un paramètre.

Jusqu'à l'achèvement de la distribution, vous pouvez inspecter la publication à l'aide du jeton de distribution renvoyé en appelant `token.getMessage`.

Distributions terminées

L'achèvement des distributions est asynchrone et dépend de la qualité de service associée à la publication.

Au plus une fois

`QoS=0`

La distribution est terminée immédiatement en retour de `MqttTopic.publish`. `MqttCallback.deliveryComplete` est appelé immédiatement.

Au moins une fois

`QoS=1`

La distribution est terminée lorsqu'un accusé de réception de la publication a été reçu du gestionnaire de files d'attente. `MqttCallback.deliveryComplete` est appelé lors de la réception de l'accusé de réception. Le message peut être distribué plusieurs fois avant que `MqttCallback.deliveryComplete` ne soit appelé si les communications sont lentes ou peu fiables.

Une seule fois

`QoS=2`

La distribution est achevée lorsque le client reçoit un message d'achèvement indiquant que la publication a été diffusée aux abonnés. `MqttCallback.deliveryComplete` est appelé dès la réception du message de publication. Il n'attend pas le message d'achèvement.

Dans de rares cas, votre application client risque de ne pas revenir au client MQTT à partir de `MqttCallback.deliveryComplete` normalement. Vous savez alors que la distribution est terminée parce que `MqttCallback.deliveryComplete` a été appelé. Si le client redémarre la même session, `MqttCallback.deliveryComplete` n'est pas rappelé.

Distributions incomplètes

Si la distribution est incomplète après la déconnexion de la session du client, vous pouvez connecter le client à nouveau et terminer la distribution. Vous pouvez uniquement terminer la distribution d'un message si ce dernier a été publié dans une session avec l'attribut `MqttConnectionOptions` à la valeur `false`.

Créez le client à l'aide du même identificateur de client et adresse de serveur, puis connectez-vous en attribuant une nouvelle fois à l'attribut `cleanSession` `MqttConnectionOptions` la valeur `false`. Si vous attribuez à `cleanSession` la valeur `true`, les jetons de distribution en attente sont rebutés.

Vous pouvez vérifier s'il existe des distributions en attente en appelant `MqttClient.getPendingDeliveryTokens`. Vous pouvez appeler `MqttClient.getPendingDeliveryTokens` avant de connecter le client.

Publication Last will and testament (dernières volontés et testament)

Si une connexion client MQTT s'arrête de manière inattendue, vous pouvez configurer WebSphere MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Créez une rubrique pour "dernières volontés et testament". Vous pouvez créer une rubrique telle que `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Configurez "dernières volontés et testament" à l'aide de la méthode `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Pensez à créer un horodatage dans le message `lastWillPayload`. Incluez d'autres informations client qui permettent d'identifier le client et les circonstances de la connexion. Transmettez l'objet `MqttConnectionOptions` au constructeur `MqttClient`.

Attribuez à `lastWillQos` la valeur 1 ou 2, afin que le message devienne persistant dans IBM WebSphere MQ, et de garantir la distribution. Pour conserver les dernières informations de connexion perdues, attribuez à `lastWillRetained` la valeur `true`.

La publication de type "dernières volontés et testament" est envoyée aux abonnés si la connexion est interrompue de manière inattendue. Elle est envoyée si la connexion s'arrête sans que le client appelle la méthode `MqttClient.disconnect`.

Pour surveiller les connexions, complétez la publication de type "dernières volontés et testament" avec d'autres publications afin d'enregistrer des connexions et des déconnexions programmées.

Persistance des messages dans les clients MQTT

Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions : pour les publications envoyées au client ou reçues par ce dernier.

Dans MQTT, la persistance de message a deux aspects : la façon dont le message est transféré et sa mise en file d'attente dans le serveur MQTT en tant que message persistant.

1. Le client MQTT combine la persistance des messages et la qualité de service. Selon la qualité de service que vous choisissez pour un message, celui-ci devient persistant. La persistance de message est nécessaire pour implémenter la qualité de service requise.

   Si vous indiquez "au moins une fois", QoS=0, le client supprime le message dès qu'il est publié. S'il se produit un incident lors du traitement en amont du message, ce dernier n'est pas renvoyé. Même si le client reste actif, le message n'est pas renvoyé. Le comportement des messages QoS=0 est identique à celui des messages non persistants rapides IBM WebSphere MQ .

   Si un message est publié par un client avec une qualité de service QoS de 1 ou 2, il devient persistant. Le message est stocké en local et supprimé du client uniquement lorsqu'il n'est plus nécessaire de garantir une distribution "au moins une fois" avec une qualité de service QoS=1 ou "une seule fois", avec une qualité de service QoS=2.
2. Si un message est indiqué avec une qualité de service QoS de 1 ou 2, il est mis en file d'attente comme message persistant. S'il est indiqué comme QoS=0, alors il est mis en file d'attente comme message non persistant. Dans IBM WebSphere MQ , les messages non persistants sont transférés entre les gestionnaires de files d'attente "une seule fois", sauf si le canal de messages possède l'attribut NPMSPEED défini sur FAST.

Une publication persistante est stockée sur le client jusqu'à ce qu'elle soit reçue par une application client. Pour QoS=2, la publication est supprimée du client lorsque le rappel de l'application renvoie le contrôle. Pour QoS=1 l'application peut recevoir une nouvelle fois la publication si un incident se produit. Pour QoS=0, le rappel reçoit la publication une seule fois. Il est possible qu'il ne reçoive pas de publication en cas d'incident ou si le client est déconnecté au moment de la publication.

Lorsque vous vous abonnez à une rubrique, vous pouvez diminuer la qualité de service QoS avec laquelle l'abonné reçoit les messages afin de correspondre à ses aptitudes à la persistance. Les publications créées avec une qualité de service QoS supérieure sont envoyées avec la qualité de service QoS la plus élevée demandée par l'abonné.

Stockage de messages

L'implémentation de stockage de données sur des dispositifs de petites taille varie considérablement. Le modèle d'enregistrement temporaire des messages persistants dans le stockage géré par le client MQTT peut être trop lent ou nécessiter trop de stockage. Sur les périphériques mobiles, le système d'exploitation mobile peut fournir un service de stockage idéal pour les messages MQTT .

Pour vous permettre de répondre aux contraintes des petits périphériques, le client MQTT dispose de deux interfaces de persistance. Ces interfaces définissent les opérations impliquées dans le stockage de messages persistants. Ces interfaces sont décrites dans la documentation d'API du client MQTT pour Java. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference. Vous pouvez implémenter les interfaces afin de les adapter au dispositif. Le client MQTT qui s'exécute sur Java SE possède une implémentation par défaut des interfaces qui stockent les messages persistants dans le système de fichiers. Il utilise le package `java.io`. Le client dispose également d'une implémentation par défaut pour Java ME, `MqttDefaultMIDPPersistence`.

Classes de persistance

`MqttClientPersistence`
: Envoyez une instance de l'implémentation de `MqttClientPersistence` au client MQTT sous la forme d'un paramètre du constructeur `MqttClient`. Si vous omettez le paramètre `MqttClientPersistence` dans le constructeur `MqttClient` , le client MQTT stocke les messages persistants à l'aide de la classe `MqttDefaultFilePersistence` ou `MqttDefaultMIDPPersistence`.

`MqttPersistable`
: `MqttClientPersistence` extrait et insère les objets `MqttPersistable` à l'aide d'une clé de protection. Vous devez fournir une implémentation de `MqttPersistable` ainsi qu'une implémentation de `MqttClientPersistence` si vous n'utilisez pas la classe `MqttDefaultFilePersistence` ou `MqttDefaultMIDPPersistence`.

MqttDefaultFilePersistence

Le client MQTT fournit la classe `MqttDefaultFilePersistence`. Si vous instanciez `MqttDefaultFilePersistence` dans votre application client, vous pouvez indiquer le répertoire dans lequel les messages persistants doivent être stockés en tant que paramètre du constructeur `MqttDefaultFilePersistence`.

Le client MQTT peut aussi instancier `MqttDefaultFilePersistence` et placer les fichiers dans un répertoire par défaut. Le nom du répertoire est *`client identifier`*`-tcp` *`hostname portnumber`*. Les caractères `"\", "\\", "/", ":"` et `" "` sont supprimés de la chaîne de nom de répertoire.

Le chemin d'accès au répertoire est la valeur de la propriété système `rcp.data`. Si `rcp.data` n'est pas défini, le chemin est la valeur de la propriété système `usr.data`.

`rcp.data` est une propriété associée à l'installation d'une initiative OSGi ou d'une application RCP (Rich Client Platform) Eclipse.

`usr.data` est le répertoire dans lequel la commande Java qui a démarré l'application a été lancée.

MqttDefaultMIDPPersistence

`MqttDefaultMIDPPersistence` a un constructeur par défaut et aucun paramètre. Il utilise le package `javax.microedition.rms.RecordStore` pour stocker les messages.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Un `MqttMessage` comporte un tableau d'octets en tant que charge. Faites en sorte que la taille des messages soit la plus petite possible. La longueur maximale des messages autorisée par le protocole MQTT est 250 Mo.

Généralement, un programme client MQTT utilise `java.lang.String` ou `java.lang.StringBuffer` pour manipuler le contenu des messages. Par commodité, la classe `MqttMessage` comprend une méthode `toString` pour convertir sa charge en chaîne. Pour créer une charge de tableau d'octets à partir de `java.lang.String` ou de `java.lang.StringBuffer`, utilisez la méthode `getBytes`.

La méthode `getBytes` convertir une chaîne en jeu de caractères par défaut pour la plateforme. Le jeu de caractères par défaut utilisé est généralement `UTF-8`. Les publications MQTT qui contiennent uniquement du texte sont généralement codées dans `UTF-8`. Utilisez la méthode `getBytes("UTF8")` pour remplacer le jeu de caractères par défaut.

Dans IBM WebSphere MQ, une publication MQTT est reçue en tant que message `jms-bytes`. Le message inclut un dossier `MQRFH2` contenant un dossier `<mqtt>`et un dossier `<mqps>` . Le dossier `<mqtt>` contient `clientId` et `qos`, mais ce contenu peut changer à l'avenir.

Une méthode `MqttMessage` comprend trois attributs supplémentaires : qualité de service, s'il est conservé et s'il est dupliqué. Un indicateur dupliqué est défini uniquement si la qualité de service est "au moins une fois" once" ou "une seule fois". Si le message a été envoyé précédemment et qu'il n'a pas été acquitté assez rapidement par le client MQTT , le message est envoyé à nouveau avec l'attribut en double défini sur `true`.

Publication

Pour créer une publication dans une application client MQTT , créez un `MqttMessage`. Définissez son contenu, la qualité du service et son éventuelle conservation, puis appelez la méthode `MqttTopic.publish(MqttMessage message)`. `MqttDeliveryToken` est renvoyé et la fin de la publication est asynchrone.

Le client MQTT peut également créer un objet de message temporaire pour vous à partir des paramètres de la méthode `MqttTopic.publish(byte [] payload, int qos, boolean retained)` lorsqu'il crée une publication.

Si la publication possède une qualité de service "au moins une fois" ou "une seule fois", QoS=1 ou QoS=2, le client MQTT appelle l'interface `MqttClientPersistence` . Il appelle `MqttClientPersistence` pour stocker le message avant de renvoyer un jeton de distribution à l'application.

L'application peut choisir de se bloquer jusqu'à la distribution du message au serveur, à l'aide de la méthode `MqttDeliveryToken.waitForCompletion`. L'application peut également continuer sans blocage. Si vous voulez vérifier la distribution effective des publications sans créer de blocage, enregistrez une instance d'une classe de rappel implémentant `MqttCallback` auprès du client MQTT. Le client MQTT appelle la méthode `MqttCallback.deliveryComplete` dès que la publication a été distribuée. En fonction de la qualité de service, la distribution peut être presque immédiate pour QoS=0 ou peut prendre un certain temps pour QoS=2.

Utilisez la méthode `MqttDeliveryToken.isComplete` pour vérifier si la distribution est terminée. Alors que la valeur de `MqttDeliveryToken.isComplete` est `false`, vous pouvez appeler `MqttDeliveryToken.getMessage` pour obtenir le contenu du message. Si le résultat de l'appel de `MqttDeliveryToken.isComplete` est `true`, le message a été supprimé et l'appel de `MqttDeliveryToken.getMessage` a envoyé une exception de pointeur Null. Il n'existe pas de synchronisation intégrée entre `MqttDeliveryToken.getMessage` et `MqttDeliveryToken.isComplete`.

Si le client se déconnecte avant de recevoir tous les jetons de distribution en attente, une nouvelle instance du client peut effectuer une requête pour les jetons de distribution en attente avant la connexion. Aucune nouvelle distribution n'est effectuée jusqu'à ce que le client se connecte et il est plus sûr d'appeler la méthode `MqttDeliveryToken.getMessage`. Utilisez la méthode `MqttDeliveryToken.getMessage` pour trouver les publications qui n'ont pas été distribuées. Les jetons de distributions sont supprimés si vous vous connectez avec `MqttConnectOptions.cleanSession` défini à sa valeur par défaut `true`.

Abonnement

Un gestionnaire de file d'attente ou IBM MessageSight est responsable de la création des publications à envoyer à un abonné MQTT. Le gestionnaire de files d'attente vérifie si le filtre de sujet dans un abonnement créé par un client MQTT correspond à la chaîne de sujet dans une publication. La correspondance peut être exacte ou comprendre des caractères génériques. Avant de transférer la publication à l'abonné par le gestionnaire de files d'attente, ce dernier vérifie les attributs associés à la publication. Il suit la procédure de recherche décrite à la rubrique Abonnement à l'aide d'une chaîne de sujet contenant des caractères génériques pour identifier si un objet sujet d'administration accorde à l'utilisateur le droit de s'abonner.

Lorsque le client MQTT reçoit une publication avec une qualité de service "au moins une fois", il appelle la méthode `MqttCallback.messageArrived` pour traiter la publication. Si la qualité de service de la publication est "une seule fois", QoS=2, le client MQTT appelle l'interface `MqttClientPersistence` pour stocker le message lorsqu'il est reçu. Il appelle la méthode `MqttCallback.messageArrived`.

Qualités de service fournies par un client MQTT

Un client MQTT fournit trois qualités de service pour la distribution des publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

La qualité de service d'une publication est un attribut de `MqttMessage`. Elle est définie par la méthode `MqttMessage.setQos`.

La méthode `MqttClient.subscribe` peut réduire la qualité de service appliquée aux publications envoyées à un client sur une rubrique. La qualité de service d'une publication transférée à un abonné peut être différente de la qualité de service de la publication. La plus faible des deux valeurs est utilisée pour transférer une publication.

Au plus une fois
QoS=0

Le message est distribué une fois tout au plus, ou n'est pas distribué du tout. Sa distribution via le réseau n'est pas accompagnée d'un accusé de réception.

Le message n'est pas stocké. Le message peut être perdu si le client est déconnecté ou si le serveur échoue.

`QoS=0` est le mode de transfert le plus rapide. Il est parfois appelé "autonome après diffusion".

Le protocole MQTT ne nécessite pas que les serveurs envoient les publications avec `QoS=0` au client . Si le client est déconnecté au moment où le serveur reçoit la publication, cette dernière peut être supprimée en fonction du serveur. Le service de télémétrie (MQXR) ne supprime pas les messages envoyés avec `QoS=0`. Ils sont stockés en tant que messages non persistants et sont uniquement supprimés en cas d'arrêt du gestionnaire de files d'attente.

Au moins une fois

`QoS=1`

`QoS=1` est le mode de transfert par défaut.

Le message est toujours distribué au moins une fois. Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur DUP défini jusqu'à la réception de l'accusé de réception. En conséquence, le destinataire peut recevoir le même message à plusieurs reprises, et le traiter plusieurs fois.

Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité.

Le message est supprimé du destinataire après son traitement. Si le destinataire est un courtier, le message est publié pour ses abonnés. Si le destinataire est un client, le message est distribué à l'application de l'abonné. Une fois que le message est supprimé, le destinataire envoie un accusé de réception à l'expéditeur.

Le message est supprimé de l'expéditeur une fois qu'il a reçu un accusé de réception de la part du destinataire.

Une seule fois

`QoS=2`

Le message est toujours distribué une seule fois.

Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité.

`QoS=2` est le mode de transfert le plus sûr, mais le plus lent. Il faut au moins deux paires de transmissions entre l'expéditeur et le destinataire avant la suppression du message côté expéditeur. Le message peut être traité côté destinataire après la première transmission.

Dans la première paire de transmissions, l'expéditeur transmet le message et reçoit un accusé de réception du destinataire indiquant qu'il a stocké le message. Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur DUP défini jusqu'à la réception de l'accusé de réception.

Dans le cadre de la seconde paire de transmissions, l'expéditeur dit au destinataire qu'il peut terminer le traitement du message, `"PUBREL"`. Si l'expéditeur ne reçoit pas d'accusé de réception du message `"PUBREL"`, le message `"PUBREL"` est renvoyé jusqu'à la réception de l'accusé de réception. L'expéditeur supprime le message qu'il a enregistré lors de la réception de l'accusé de réception au message `"PUBREL"`

Le destinataire peut traiter le message lors de la première ou de la seconde phase à condition de ne pas traiter à nouveau le message. Si le destinataire est un courtier, il publie le message pour les abonnés. Si le destinataire est un client, il livre le message à l'application de l'abonné. Le destinataire renvoie un message d'achèvement à l'expéditeur indiquant qu'il a terminé le traitement du message.

Publications conservées et clients MQTT

Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

Utilisez la méthode `MqttMessage.setRetained` pour indiquer si la publication sur une rubrique est conservée ou non.

Pour supprimer une publication conservée dans IBM WebSphere MQ, exécutez la commande MQSCCLEAR TOPICSTR**CLEAR TOPICSTR**.

Si vous créez une publication avec une charge utile null, la publication vide est envoyée aux abonnements. Les autres courtiers MQTT peuvent ne pas envoyer de publication vide aux abonnés.

Si vous publiez une publication non conservée dans une rubrique ayant une publication conservée, cette dernière n'est pas affectée. Les abonnés actuels reçoivent la nouvelle publication. Les nouveaux abonnés reçoivent d'abord la publication conservée, puis les nouvelles publications.

Lorsque vous créez ou mettez à jour une publication conservée, envoyez la publication avec un QoS ou 1 ou 2. Si vous l'envoyez avec un QoS de 0, IBM WebSphere MQ crée une publication conservée non permanente. La publication n'est pas conservée en cas d'arrêt du gestionnaire de files d'attente.

Utilisez les publications conservées pour enregistrer la dernière valeur d'une mesure. Les nouveaux abonnés à la rubrique conservée reçoivent immédiatement la valeur de mesure la plus récente. Si aucune mesure n'a été prise depuis le dernier abonnement à la rubrique par l'abonné et que l'abonné s'abonne de nouveau, il reçoit de nouveau la dernière publication conservée dans la rubrique.

Abonnements

Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Créez des abonnement à l'aide des méthodes `MqttClient.subscribe`, et transmettez un ou plusieurs filtres de rubrique et des paramètres de qualité de service. Le paramètre de qualité de service définit la qualité de service maximale à laquelle un abonné est préparé pour recevoir un message. Les messages envoyés à ce client ne peuvent pas être distribués avec la qualité de service la plus élevée. La qualité de service est définie au niveau le plus bas de la valeur d'origine lorsque le message a été publié et que le niveau a été spécifié pour l'abonnement. La qualité de service par défaut pour la réception de messages est `QoS=1`, au moins une fois.

La demande d'abonnement elle-même est envoyée avec la qualité de service `QoS=1`.

Les publications sont reçues par un abonné lorsque le client MQTT appelle la méthode `MqttCallback.messageArrived` . La méthode `messageArrived` transmet également la chaîne de sujet avec laquelle le message a été publié à l'abonné.

Vous pouvez supprimer l'abonnement ou un ensemble d'abonnements à l'aide des méthodes `MqttClient.unsubscribe`.

Une commande WebSphere MQ peut supprimer un abonnement.Répertorie les abonnements à l'aide de WebSphere MQ Explorer ou à l'aide de commandes **runmqsc** ou PCF. Tous les abonnements client MQTT sont nommés. Ils reçoivent un nom au format suivant: *ClientIdentifier*:*Topic name*

Si vous utilisez la valeur par défaut `MqttConnectOptions`ou que vous définissez `MqttConnectOptions.cleanSession` sur `true` avant de connecter le client, les anciens abonnements du client sont supprimés lorsque le client se connecte. Les nouveaux abonnements effectués par le client lors de la session sont supprimés lorsque celui-ci se déconnecte.

Si vous définissez `MqttConnectOptions.cleanSession` sur `false` avant de se connecter, tous les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour le client avant sa connexion. Tous les abonnements restent actifs au moment de la déconnexion du client.

Un autre moyen de comprendre l'impact de l'attribut `cleanSession` sur les abonnements consiste à le considérer comme un attribut modal. Dans le cadre de son mode par défaut `cleanSession=true`, le client crée des abonnements et reçoit des publications uniquement dans la portée de la session. En mode alternatif, `cleanSession=false`, les abonnements sont durables. Le client peut se connecter et se déconnecter, et ses abonnements restent actifs. Lorsque le client se reconnecte, il reçoit les publications

non distribuées. Lors de la connexion, il peut modifier l'ensemble des abonnements qui sont actifs en son nom.

Vous devez définir le mode `cleanSession` avant de vous connecter ; le mode dure toute la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez de l'utilisation de `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimés.

Les publications qui correspondent aux abonnements actifs sont envoyées au client dès leur publication. Si le client est déconnecté, elles sont envoyées au client si ce dernier se reconnecte au même serveur avec le même identificateur client et qu'il est attribué à `MqttConnectOptions.cleanSession` la valeur `false`.

Les abonnements pour un client particulier sont identifiés par l'identificateur client. Vous pouvez vous reconnecter au client à partir d'un dispositif client au même serveur et continuer avec les mêmes abonnements et recevoir des publications non distribuées.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT

Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM WebSphere MQ.

Les chaînes de rubrique sont utilisées pour envoyer des publications aux abonnés. Créez une chaîne de rubrique à l'aide de la méthode `MqttClient.getTopic(java.lang.String topicString)`.

Les filtres de rubrique permettent de s'abonner à des rubriques et de recevoir des publications. Les filtres de rubrique peuvent contenir des caractères génériques. Ces derniers vous permettent de vous abonner à plusieurs rubriques. Créez un filtre de rubrique à l'aide d'une méthode d'abonnement ; par exemple, `MqttClient.subscribe(java.lang.String topicFilter)`.

Chaînes de rubrique

La syntaxe d'une chaîne de rubrique IBM WebSphere MQ est décrite dans Chaînes de rubrique. La syntaxe des chaînes de rubrique MQTT est décrite dans la classe `MqttClient` de la documentation de l'API pour client MQTT pour Java. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference..

La syntaxe de chaque type de chaîne de rubrique est presque identique. Il existe quatre différences mineures :

1. Les chaînes de rubrique envoyées à IBM WebSphere MQ par les clients MQTT doivent suivre la convention de nom des gestionnaires de files d'attente. En particulier, aucune chaîne de rubrique ne peut contenir de trait d'union.
2. La longueur maximale est différente. Les chaînes de rubrique IBM WebSphere MQ sont limitées à 10 240 caractères. Un client MQTT peut créer des chaînes de rubrique pouvant atteindre 65535 octets.
3. Une chaîne de sujet créée par un client MQTT ne peut pas contenir de caractère null.
4. Dans WebSphere Message Broker, un niveau de rubrique null, `'...//...'` n'était pas valide. Les niveaux de rubrique Null sont pris en charge par IBM WebSphere MQ.

Contrairement à la publication/l'abonnement IBM WebSphere MQ, le protocole `mqttv3` n'utilise pas le concept d'objet de rubrique d'administration. Vous ne pouvez pas construire une chaîne de sujet à partir d'un objet de rubrique et d'une chaîne de sujet. Toutefois, une chaîne de sujet est mappée à un sujet d'administration dans WebSphere MQ. Le contrôle d'accès associé au sujet d'administration détermine si une publication est diffusée dans le sujet ou si elle et supprimée. Les attributs qui sont appliqués à la publication lorsqu'elle est transférée aux abonnés sont influencés par les attributs de sujet d'administration.

Filtres de rubrique

La syntaxe d'un filtre de rubrique IBM WebSphere MQ est décrite dans Schéma de caractères génériques (de type rubrique). La syntaxe des filtres de sujet que vous pouvez construire avec un client MQTT est décrite dans la classe `MqttClient` de la documentation d'API du client MQTT pour Java.Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

La syntaxe de chaque type de filtre de rubrique est presque identique. La seule différence réside dans la manière dont différents courtiers MQTT interprètent un filtre de sujet. Dans WebSphere Message Broker V6, un caractère générique multi-niveau pouvait uniquement être utilisé à la fin du filtre de rubrique. Dans WebSphere MQ, un caractère générique à plusieurs niveaux peut être utilisé à n'importe quel niveau de l'arborescence de rubriques ; par exemple, `USA/#/Dutchess County`.

Références pour la programmation client MQTT

Voici des liens vers Module de clientMobile Messaging et M2M, et vers la documentation associée des API client.

Dans Module de clientMobile Messaging et M2M, les bibliothèques client MQTT sont conditionnées avec la documentation de l'API générée. Vous pouvez télécharger le pack client depuis IBM messaging community downloads.

Vous pouvez consulter les copies en ligne de la documentation la plus récente sur les API en suivant ces liens vers le projet Eclipse Paho :

- Classes du client MQTT pour Java
- Bibliothèque client MQTT pour C
- Bibliothèque client MQTT asynchrone pour C

Remarque :

1. Liez les applications MQTT Java au package `org.eclipse.paho.client.mqttv3` plutôt qu'à `com.ibm.micro.client.mqttv3`. créé. Le package `com.ibm.micro.client.mqttv3` est fourni pour prendre en charge les applications MQTT Java existantes.
2. V 7.5.0.1 Liez les applications client MQTT pour C à la bibliothèque `MQTTAsync` plutôt qu'à la bibliothèque `MQTTClient` . `MQTTClient` est fourni pour prendre en charge les applications MQTT existantes pour C.
3. Le client de messagerie MQTT pour JavaScript requiert un serveur MQTT prenant en charge WebSockets. Par exemple, c'est le cas de IBM WebSphere MQ Version 7.5 et versions ultérieures.

Initiation aux serveurs MQTT

Des serveurs de messagerie prenant en charge le protocole de transport MQTT sont disponibles auprès d'IBM et d'autres fournisseurs. Le serveur MQTT le plus simple permet aux apps et aux appareils mobiles, pris en charge par les bibliothèques client MQTT, d'échanger des messages. IBM WebSphere MQ et IBM MessageSight sont des serveurs MQTT d'IBM. Outre leur activité de serveur MQTT de base, ils échangent également les messages entre les apps client MQTT et les apps d'entreprise. Tous les serveurs MQTT d'IBM prennent en charge le protocoleMQTT version 3.1 et MQTT sur WebSocket protocol.

Serveurs MQTT IBM actuels

IBM WebSphere MQ

- IBM WebSphere MQ fournit un système de messagerie au niveau de l'entreprise. Le composant de télémétrie permet aussi à IBM WebSphere MQ d'agir en tant que serveur MQTT.
- Celui-ci prend en charge les apps mobiles, machine-to-machine (M2M) et spécifiques aux appareils, et leur permet également d'échanger des messages avec les apps de messagerie d'entreprise telles que les apps IBM WebSphere MQ et JMS.

- L'installation IBM WebSphere MQ contient une copie du kit de développement de logiciels MQTT d'IBM. Ce kit de développement de logiciels contient des modèles d'apps client MQTT, et des bibliothèques client MQTT qui prennent en charge ces apps.

  Remarque : Pour obtenir la dernière version de ce SDK, téléchargez Module de clientMobile Messaging et M2M. Pour plus d'informations, voir «Initiation aux client MQTT», à la page 11.
- La prise en charge de MQTT a été d'abord incluse dans IBM WebSphere MQ Version 7.0.1. Pour des informations complètes sur chaque édition de IBM WebSphere MQ, voir la documentation du produit suivante :
  - WebSphere MQ Telemetry version 7.5
  - WebSphere MQ Telemetry Version 7.1

  Pour obtenir une introduction à IBM WebSphere MQ, et pour vous familiariser avec le composant IBM WebSphere MQ Telemetry, voir «IBM WebSphere MQ en tant que serveur MQTT», à la page 140.

IBM MessageSight

- IBM MessageSight est un serveur MQTT pour dispositifs, qui permet de connecter simultanément un grand nombre de clients MQTT, et de fournir les performances et l'évolutivité nécessaire pour s'adapter au nombre toujours croissant d'appareils et de détecteurs mobiles. Il prend en charge le protocole MQTT version 3.1 et MQTT sur WebSocket protocol.

- Voici les principaux avantages et inconvénients d'IBM MessageSight en tant que serveur MQTT :
  - Une messagerie haute-performance, fiable et évolutive.
  - Une conception dédiée aux scénarios de machine à machine (M2M) et Internet des objets, en prenant en charge des communautés nombreuses pour les noeuds finaux connectés simultanément.
  - Facilité d'installation et utilisation. Il peut être opérationnel en moins de 30 minutes.
  - La prise en charge des apps mobiles natives pour Android et iOS.
  - Intégration à IBM WebSphere MQ en tant que courtier de publication/abonnement.
- Pour une introduction rapide à IBM MessageSight, voir l'introduction à MessageSight sur YouTube et l'annonce MessageSight. Pour des informations techniques détaillées, voir la documentation du produit MessageSight.

IBM WebSphere MQ Telemetry daemon for devices

- Aussi appelé IBM WebSphere MQ Telemetry advanced client for C. Il s'agit d'un serveur MQTT à faible encombrement qui fonctionne généralement sur les systèmes de localisation par satellite ou les appareils situés à la périphérie du réseau, tels que les boîtiers décodeurs, les unités de télémétrie distantes ou les terminaux point de vente.
- Il sont souvent utilisés pour concentrer un grand nombre de connexions client MQTT, connectés ensuite à IBM WebSphere MQ sur Internet dans une même connexion MQTT. Par exemple, vous pouvez installer un grand nombre de détecteurs dans un bâtiment, les connecter au IBM WebSphere MQ Telemetry daemon for devices, et connecter le démon à IBM WebSphere MQ.

- Le IBM WebSphere MQ Telemetry daemon for devices est inclus avec IBM WebSphere MQ. Une licence distincte est nécessaire pour le connecter à IBM WebSphere MQ. Voir IBM United States Software Announcement 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) est une version du IBM WebSphere MQ Telemetry daemon for devices. La principale différence réside dans son utilisation. RSMB est un petit serveur de test, disponible dans IBM alphaWorks, et destiné à l'évaluation et au test des solutions MQTT. RSMB prend en charge MQTT sur un certain nombre de plateformes Linux, sous Windows XP, Apple Mac OS X Leopard et Unslung (Linksys NSLU12).

Anciens serveurs MQTT IBM

WebSphere Message Broker ***(désormais appelé IBM Integration Bus)***

- WebSphere Message Broker version 6 contenait son propre serveur MQTT. La prise en charge a été remplacée dans WebSphere Message Broker version 7 par le composant de télémétrie de IBM WebSphere MQ.

Autres serveurs MQTT

La page Logiciels du site MQTT.org contient la liste à jour des serveurs et des courtiers MQTT, y compris les serveurs open source.

Tâches associées

«Initiation aux client MQTT», à la page 11
Familiarisez-vous avec le développement d'une app mobile ou machine-to-machine (M2M) en générant et en exécutant un modèle d'app client MQTT qui utilise une bibliothèque client MQTT. Les modèles d'app et les bibliothèques client associées sont disponibles dans Module de clientMobile Messaging et M2M d'IBM. Il existe des versions des applications et des bibliothèques client écrites dans Java, dans JavaScriptet dans C. Vous pouvez exécuter ces applications sur la plupart des plateformes et des appareils, y compris les appareils et les produits Android à partir de Apple.

IBM WebSphere MQ en tant que serveur MQTT

Introduction à l'utilisation du serveur MQTT qui est inclus dans IBM WebSphere MQ MQ.

Pour commencer, suivez les étapes décrites dans les articles suivants :

- «InstallationIBM WebSphere MQ», à la page 141
- «Configuration du service MQTT depuis la ligne de commande», à la page 142
- «Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer», à la page 145

Remarque : Vous pouvez être opérationnel rapidement en utilisant l'exemple avec l'interface de ligne de commande. Cependant, si votre configuration est très différente de celle de l'exemple, il vous faudra plus de connaissances et de compétences pour utiliser l'interface de ligne de commande de façon efficace. L'interface de IBM WebSphere MQ Explorer vous permet de réaliser facilement les tâches de base, ainsi que les tâches de configuration standard.

Des informations conceptuelles sur le composant IBM WebSphere MQ Telemetry sont disponibles dans les articles suivants de la documentation du produit IBM WebSphere MQ :

- Connexion de dispositifs de télémétrie à un gestionnaire de files d'attente
- Service de télémétrie (MQXR)
- Canaux de télémétrie

Information associée

Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Linux et AIX
Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Windows
Configuration des files d'attente réparties en vue de l'envoi de messages aux clients MQTT

InstallationIBM WebSphere MQ

Suivez ces instructions pour obtenir et installer IBM WebSphere MQ et configurez IBM WebSphere MQ Telemetry on Windows ou Linux.

Avant de commencer

Pour connaître les systèmes d'exploitation pris en charge par le service MQTT fonctionnant sous IBM WebSphere MQ, voir IBM WebSphere MQ Telemetry system requirements.

Procurez-vous une copie des éléments d'installation de IBM WebSphere MQ, ainsi que la licence, de l' une des manières suivantes :

1. Demandez les éléments d'installation à votre administrateur IBM WebSphere MQ, et vérifiez que vous pouvez accepter le contrat de licence.
2. Procurez-vous une copie d'évaluation de 90 jours de IBM WebSphere MQ. Voir la rubrique Evaluer: IBM WebSphere MQ.
3. Achetez IBM WebSphere MQ. Voir la rubrique Page du produitIBM WebSphere MQ.

Pourquoi et quand exécuter cette tâche

Installez IBM WebSphere MQ en tant que `root` sous Linux, et en tant qu'administrateur sous Windows. Lors de l'installation, sélectionnez les options supplémentaires `Telemetry Service` et `Telemetry Clients` pour installer le composant IBM WebSphere MQ Telemetry. Créez un ID utilisateur pour administrer IBM WebSphere MQ, et vérifiez que l'ID utilisateur invité est défini. L'ID utilisateur invité est utilisé dans le modèle de configuration du service MQTT pour autoriser l'accès MQTT à IBM WebSphere MQ.

Après l'installation de IBM WebSphere MQ, démarrez le service MQTT en suivant la procédure des rubriques «Configuration du service MQTT depuis la ligne de commande», à la page 142 ou «Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer», à la page 145.

Procédure

1. Connectez-vous en tant que `root` sous Linuxou en tant qu'administrateur sous Windows.
2. Installez IBM WebSphere MQ.

   Suivez les instructions de la rubrique Installation du serveur WebSphere MQ sous Linux ou Installation du serveur WebSphere MQ sous Windows. Sélectionnez `Telemetry Service` et `Telemetry Clients` pour installer le composant IBM WebSphere MQ Telemetry.

   Sous Linux, prenez note de l'instruction contenue à la section "Etapes suivantes" pour définir votre installation comme l'installation principale. Même si elle est la seule installation IBM WebSphere MQ sur votre poste de travail, elle doit être définie comme l'installation principale. Voir Installation unique de WebSphere MQ version 7.1 ou ultérieure en tant qu'installation principale.

   Vous ne pouvez suivre de façon précise les instructions du modèle de configuration que si l'installation est l'installation principale.

   Installations multiples : Si vous voulez utiliser une installation qui n'est pas une installation principale, exécutez la commande setmqenv. Elle configure un environnement IBM WebSphere MQ dans une fenêtre de commande de votre espace de travail. Voir Installations multiples.

   Si vous avez accepté les emplacements d'installation par défaut du programme d'installation, IBM WebSphere MQ est installé dans les répertoires suivants :

   Linux 64 bits

   ```
   /opt/mqm
   ```

Windows 32 bits

```
C:\Program Files\IBM\WebSphere MQ
```

Windows 64 bits

```
C:\Program Files (x86)\IBM\WebSphere MQ
```

Le répertoire d'installation figure sous la forme *MQ_INSTALLATION_PATH*.

3. Facultatif : Ajoutez l'utilisateur avec lequel vous allez administrer IBM WebSphere MQ au groupe `mqm` sur ce poste de travail.

   Cette étape est facultative sous Windows car vous pouvez administrer IBM WebSphere MQ en tant qu'administrateur Windows .Voir Droit d'administration de WebSphere MQ sur les systèmes UNIX et Windows.

   Si votre poste de travail Windows est membre d'un domaine, voir Domaine Windows 2000 avec des droits de sécurité différents de ceux par défaut, ou Windows 2003 et Windows Server 2008 avec les droits de sécurité par défaut.

   Sous Linux, le programme d'installation crée un utilisateur `mqm`, en tant que membre du groupe `mqm`. Donnez à cet utilisateur un mot de passe, ou créez un autre utilisateur avec `mqm` comme groupe principal.
4. Facultatif : Connectez-vous avec l'utilisateur que vous avez affecté au groupe `mqm`.

   Cette étape est facultative sous Windows car vous pouvez administrer IBM WebSphere MQ en tant qu'administrateur Windows .
5. Vérifiez que l'ID utilisateur invité est défini sur votre poste de travail.

   L'ID utilisateur invité est `"guest"` sur Windows et `"nobody"` sur Linux. L'ID utilisateur invité n'a pas besoin de permissions ou de droits du système d'exploitation.

Résultats

Vous avez installé IBM WebSphere MQ sur votre poste de travail en tant qu'installation IBM WebSphere MQ principale et créé le groupe `mqm`. L'installation donne le droit d'administrer IBM WebSphere MQ aux membres du groupe `mqm`. Les membres du groupe d'administrateurs sous Windows disposent également du droit d'administrer IBM WebSphere MQ.

Que faire ensuite

1. Configurez le service MQTT à partir de la ligne de commande ou IBM WebSphere MQ Explorer. Voir «Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer», à la page 145 ou «Configuration du service MQTT depuis la ligne de commande», à la page 142.
2. Testez vos clients MQTT Android, iOS, WebSockets, Java et "C".
3. Une fois le test terminé, supprimez le gestionnaire de files d'attente et le service MQTT en exécutant la commande *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` sous Windows et *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` sous Linux.

Information associée

Installation de WebSphere MQ Telemetry

Installation du serveur WebSphere MQ sous Linux

Installation du serveur WebSphere MQ sous Windows

Configuration du service MQTT depuis la ligne de commande

Suivez ces instructions pour configurer IBM WebSphere MQ à l'aide de la ligne de commande en exécutant les modèles d'application IBM WebSphere MQ Telemetry. Les étapes vous montrent comment exécuter un script pour créer un service MQTT sur un nouveau gestionnaire de files d'attente appelé `MQXR_SAMPLE_QM`.

Avant de commencer

Vous devez disposer des droits d'administration sur un gestionnaire de file d'attente IBM WebSphere MQ pour configurer le service MQTT. Il existe plusieurs manières d'accéder à un gestionnaire de file d'attente :

1. Procurez-vous une copie d'IBM WebSphere MQ et créez un gestionnaire de file d'attente sur votre propre poste de travail Linux ou Windows. Suivez les instructions de la rubrique «InstallationIBM WebSphere MQ», à la page 141 pour obtenir et installer IBM WebSphere MQ. Notez que vous devez également sélectionner `Telemetry Service` et `Telemetry Clients` lors de l'installation. Vous pouvez aussi modifier une installation existante pour ajouter ces options.
2. Demandez à un administrateur d'IBM WebSphere MQ de vous autoriser l'accès administrateur à un gestionnaire de files d'attente sur un serveur sur lequel IBM WebSphere MQ Telemetry est installé en tant qu'option. V7.5.0.1 Outre le nom du gestionnaire de file d'attente, il vous faut aussi deux ports TCP/IP pour MQTT et pour MQTTsur WebSockets. Si vous prévoyez de connecter des clients sécurisés, il vous faut au moins deux ports supplémentaires.

Pour suivre les étapes exactement comme elles sont décrites, vous devez pouvoir créer un gestionnaire de file d'attente nommé `MQXR_SAMPLE_QM`, et le port TCP/IP 1883 doit être libre.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous exécutez un script qui crée un gestionnaire de files d'attente, puis vous configurez le service MQTT pour qu'il écoute les connexions client MQTT V3.1 sur le port 1883.La configuration autorise tout le monde à publier et à diffuser des publications et à s'abonner à n'importe quel sujet. La configuration de la sécurité et du contrôle d'accès est minimale et prévue uniquement pour un gestionnaire de files d'attente sur un réseau sécurisé dont l'accès est restreint. Pour exécuter IBM WebSphere MQ et MQTT dans un environnement non sécurisé, vous devez configurer la sécurité. Pour configurer la sécurité de IBM WebSphere MQ et MQTT, voir les liens connexes à la fin de cette tâche.

Procédure

1. Connectez-vous avec un ID utilisateur disposant des droits d'administration sur IBM WebSphere MQ.

   Pour définir un ID utilisateur avec des droits d'administration sur IBM WebSphere MQ, voir l'étape 3 dans «InstallationIBM WebSphere MQ», à la page 141.
2. Ouvrez une fenêtre de commande et exécutez le modèle de script de commandes pour créer et démarrer le modèle de gestionnaire de files d'attente nomméMQXR_SAMPLE_QM et le service MQTT.

   Le chemin d'accès à l'exemple de script est `%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat` sous Windows et *MQ_INSTALLATION_PATH*`/mqxr/samples/SampleMQM.sh` sous Linux.

   Entrez la commande suivante pour créer et configurer le gestionnaire de files d'attente :

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Résultats

L'exemple crée un canal MQTT appelé `PlainText` avec les propriétés suivantes sur 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
```

Les propriétés du canal sous Linux sont les mêmes que sous Windows, sauf `com.ibm.mq.MQXR.UserName=nobody`.

Les MQTT V3.1 qui se connectent au port 1883 accèdent à IBM WebSphere MQ avec l'ID utilisateur défini dans la variable *com.ibm.mq.MQXR.UserName*. Le modèle de script autorise l'ID utilisateur avec les commandes suivantes 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
```

La première commande donne à l'utilisateur le droit de diffuser des publications et de créer des abonnements sur les sujets qui hérite des droits du sujet de base. La deuxième commande donne à l'utilisateur le droit d'insérer les messages dans les files d'attente de transmission `SYSTEM.MQTT.TRANSMIT.QUEUE`. Le service MQTT envoie des messages à `SYSTEM.MQTT.TRANSMIT.QUEUE` en tant que publications aux abonnés MQTT.

Le script démarre le service MQTT sur le gestionnaire de files d'attente pour écouter les connexions sur le port 1883.

Que faire ensuite

Suivez la procédure suivante pour tester la connexion en exécutant le modèle d'application Java MQTT V3.1.

La source de l'exemple d'application Java se trouve dans le fichier `MQTTV3Sample.java` .

Deux fenêtres de commande sont nécessaires pour exécuter le modèle. Exécutez le modèle en tant qu'abonné dans une fenêtre, et en tant que diffuseur de publications dans l'autre.

- Windows Pour démarrer l'abonné, exécutez la commande

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  Pour publier, entrez la commande

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux Pour démarrer l'abonné, exécutez la commande

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  Pour publier, entrez la commande

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

Le diffuseur de publications et l'abonné écrivent leur sortie dans leur fenêtre de commande :

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Figure 27. Sortie du diffuseur de publications

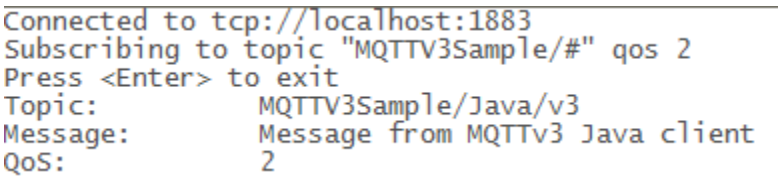


Figure 28. Sortie de l'abonné

Le serveur est maintenant prêt pour le test de votre application MQTT V3.1 .

Tâches associées

Configuration du service MQTT avec WebSphere MQ Explorer
Suivez les instructions ci-dessous pour configurer IBM WebSphere MQ en utilisant IBM WebSphere MQ Explorer pour exécuter les modèles de client IBM WebSphere MQ Telemetry. Les étapes vous montrent comment créer un service MQTT en exécutant l'assistant de configuration `Define sample` .

Information associée

WebSphere MQ Telemetry
Développement d'applications pour WebSphere MQ Telemetry
Administration de WebSphere MQ Telemetry
Sécurité de WebSphere MQ Telemetry

Configuration du service MQTT à l'aide de IBM WebSphere MQ Explorer

Suivez les instructions ci-dessous pour configurer IBM WebSphere MQ en utilisant IBM WebSphere MQ Explorer pour exécuter les modèles de client IBM WebSphere MQ Telemetry. Les étapes vous montrent comment créer un service MQTT en exécutant l'assistant de configuration `Define sample` .

Avant de commencer

Vous devez disposer des droits d'administration sur un gestionnaire de file d'attente IBM WebSphere MQ pour configurer le service MQTT. Il existe plusieurs manières d'accéder à un gestionnaire de file d'attente :

1. Procurez-vous une copie d'IBM WebSphere MQ et créez un gestionnaire de file d'attente sur votre propre poste de travail Linux ou Windows. Suivez les instructions de la rubrique «InstallationIBM WebSphere MQ», à la page 141 pour obtenir et installer IBM WebSphere MQ. Notez que vous devez également sélectionner `Telemetry Service` et `Telemetry Clients` lors de l'installation. Vous pouvez aussi modifier une installation existante pour ajouter ces options.
2. Demandez à un administrateur d'IBM WebSphere MQ de vous autoriser l'accès administrateur à un gestionnaire de files d'attente sur un serveur sur lequel IBM WebSphere MQ Telemetry est installé en tant qu'option. V 7.5.0.1 Outre le nom du gestionnaire de file d'attente, il vous faut aussi deux ports TCP/IP pour MQTT et pour MQTTsur WebSockets. Si vous prévoyez de connecter des clients sécurisés, il vous faut au moins deux ports supplémentaires.

Pour suivre les étapes exactement comme elles sont décrites, vous devez pouvoir créer un gestionnaire de file d'attente nommé `MQXR_SAMPLE_QM`, et le port TCP/IP 1883 doit être libre.

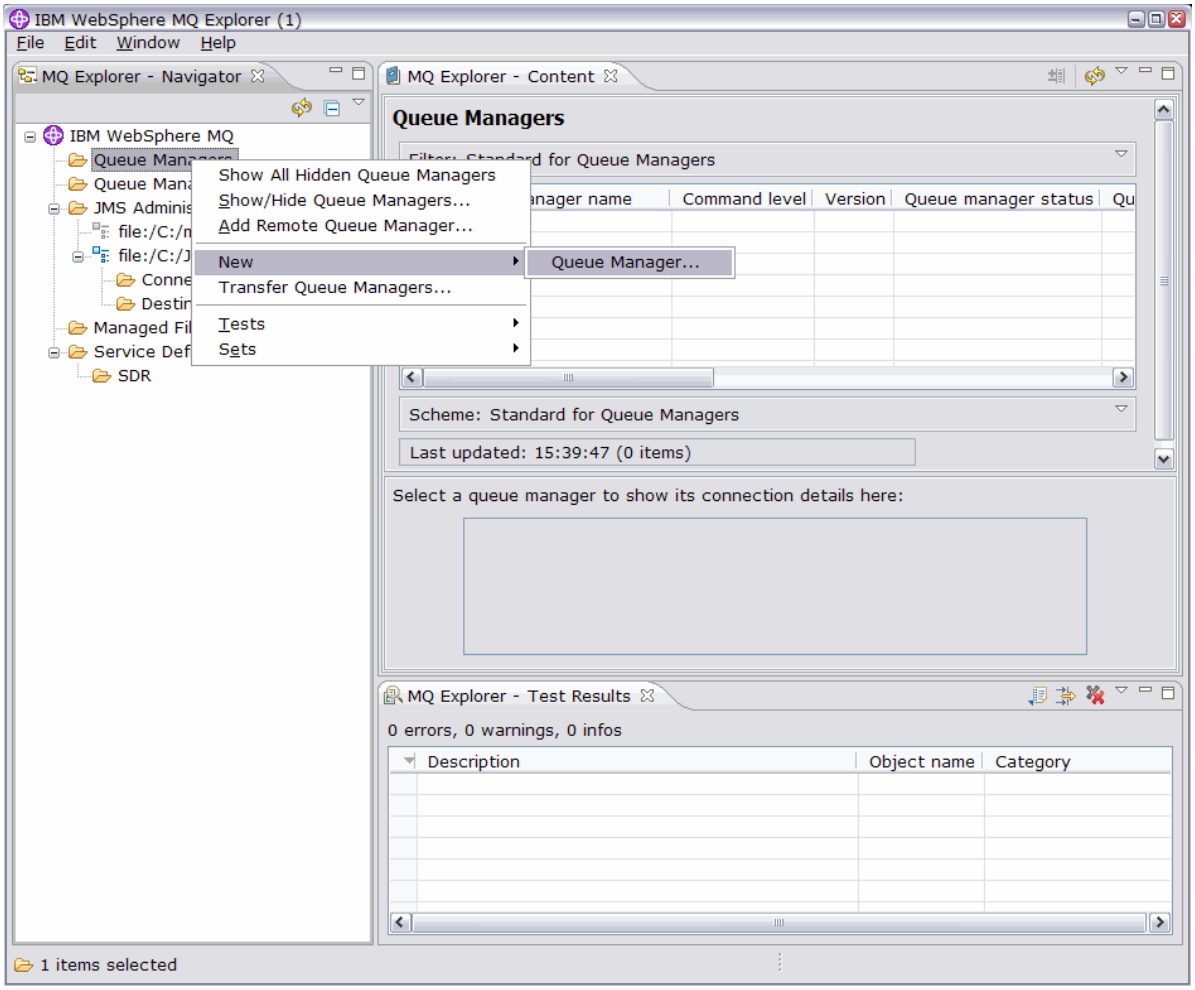
Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous exécutez l'assistant de configuration de IBM WebSphere MQ Explorer `Define sample` pour créer un service MQTT à l'écoute des connexions client MQTT V3.1 sur le port 1883. La configuration autorise tout le monde à publier et à diffuser des publications et à s'abonner à n'importe quel sujet. La configuration de la sécurité et du contrôle d'accès est minimale et prévue uniquement pour un gestionnaire de files d'attente sur un réseau sécurisé dont l'accès est restreint. Pour exécuter IBM WebSphere MQ et MQTT dans un environnement non sécurisé, vous devez configurer la sécurité. Pour configurer la sécurité de IBM WebSphere MQ et MQTT, voir les liens connexes à la fin de cette tâche.

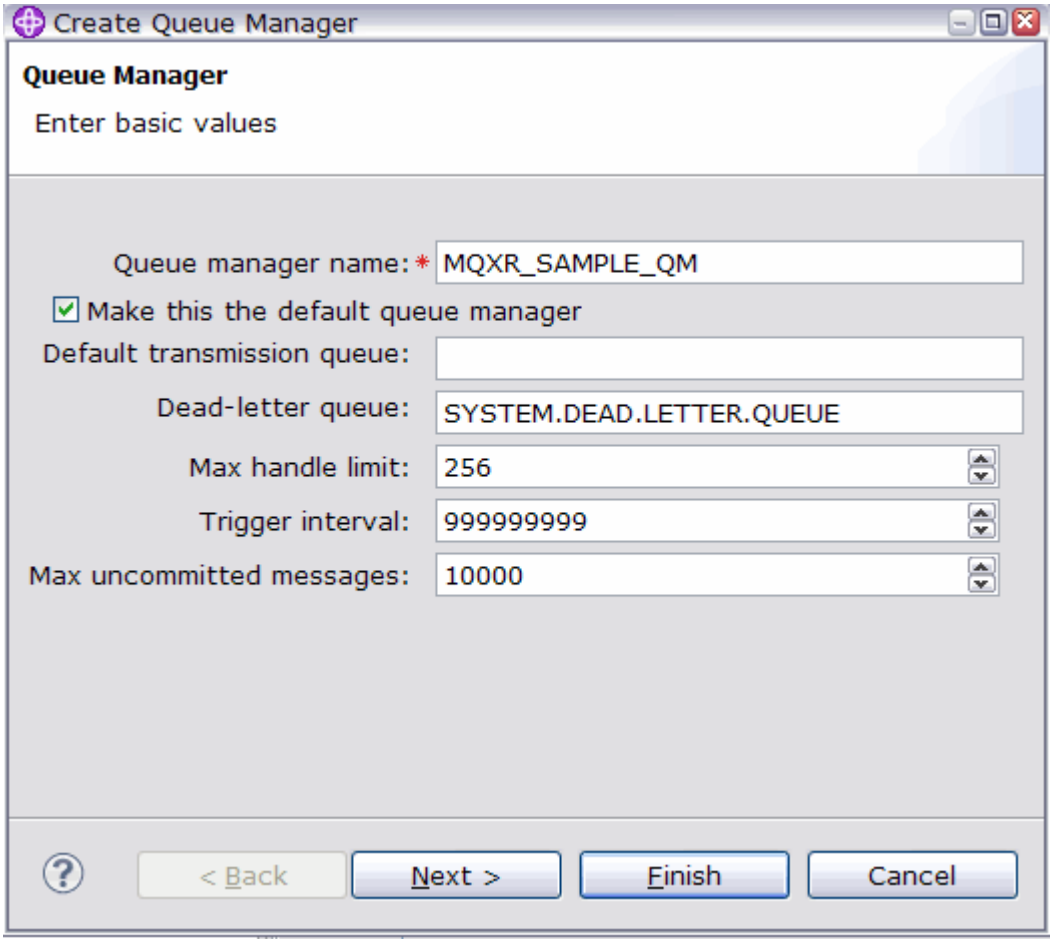
Procédure

1. Connectez-vous avec un ID utilisateur disposant des droits d'administration sur IBM WebSphere MQ.

   Pour définir un ID utilisateur avec des droits d'administration sur IBM WebSphere MQ, voir l'étape 3 dans «InstallationIBM WebSphere MQ», à la page 141.
2. Ouvrez une fenêtre de commande et exécutez la commande IBM WebSphere MQ Explorer **strmqcfg** pour démarrer IBM WebSphere MQ Explorer.
3. Création d'un gestionnaire de files d'attente
   a) Démarrez l'assistant **Nouveau gestionnaire de files d'attente**.

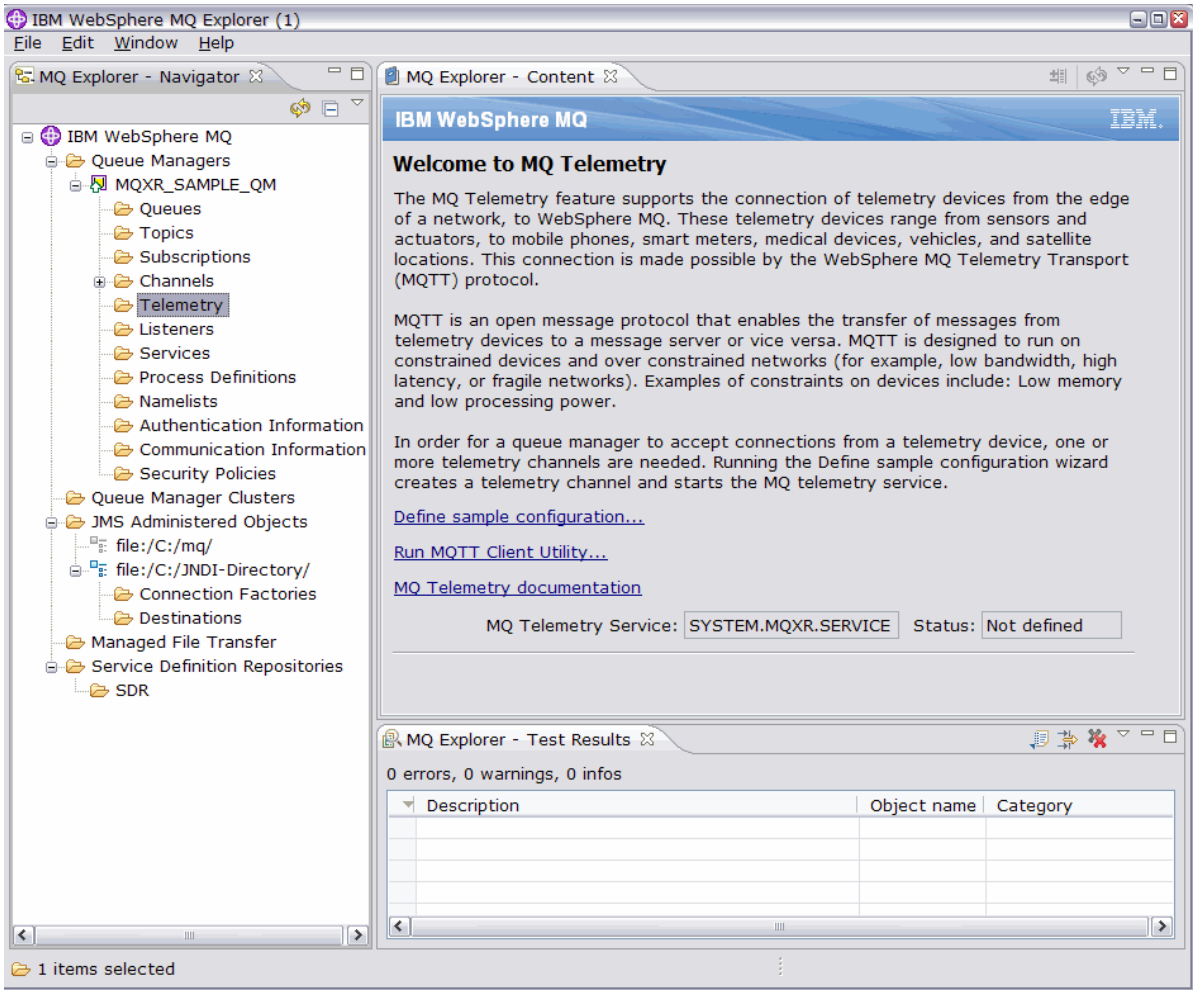


   b) Entrez le nom du **gestionnaire de files d'attente**, et celui de la **file d'attente de rebut**. Pour plus de facilité, faite de ce gestionnaire de files d'attente le gestionnaire par défaut. Cliquez sur **Terminer**.



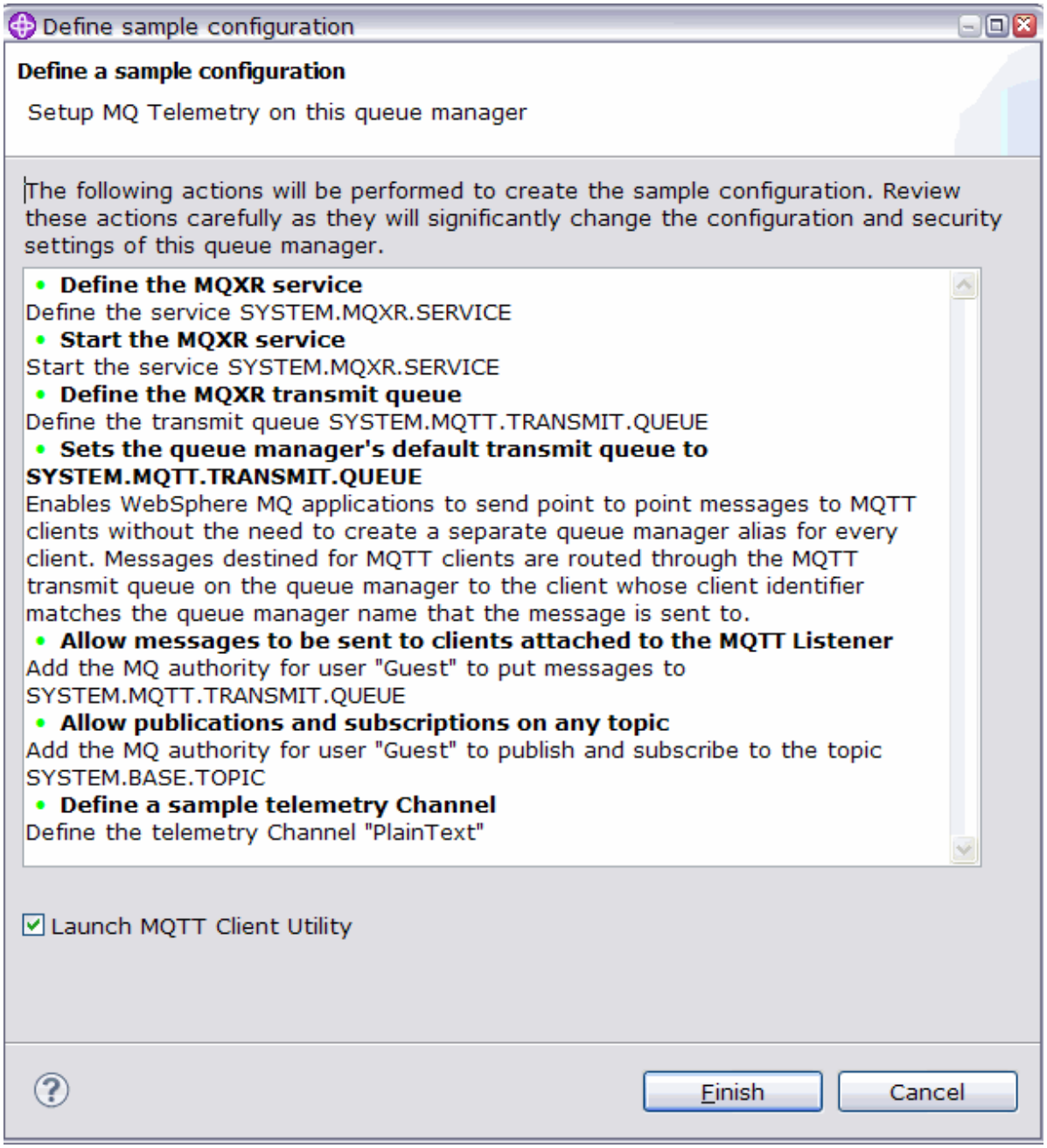
IBM WebSphere MQ Explorer crée le gestionnaire de files d'attente et le démarre.

4. Exécutez l'assistant **Définition du modèle de configuration** de Telemetry.
   a) Ouvrez le dossier Telemetry du gestionnaire de configuration.



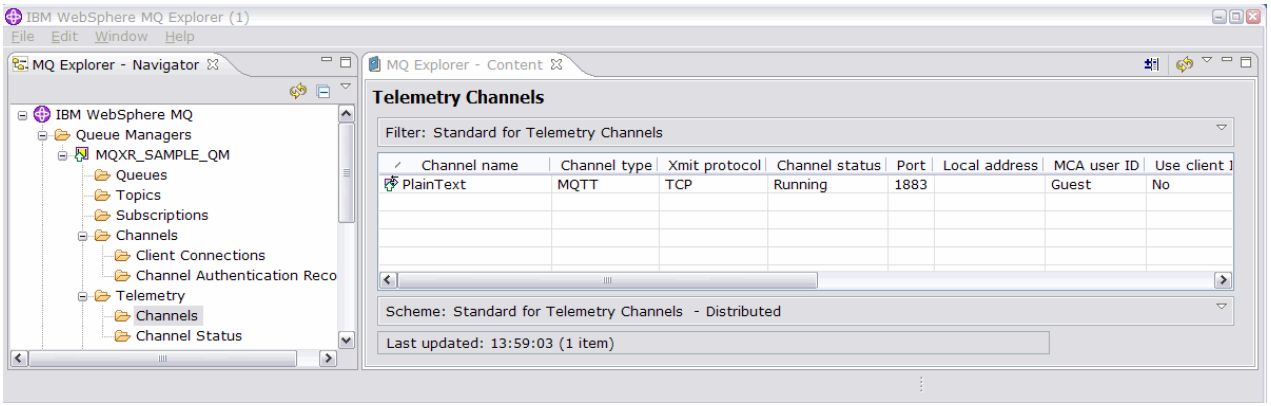
b) Cliquez sur **Définition du modèle de configuration** pour démarrer l'assistant.

c) Cliquez sur **Terminer** pour créer le service de télémétrie, et lancez MQTT Client Utility.



Résultats

Ouvrez le dossier Canaux de télémétrie pour lister les modèles de canaux.



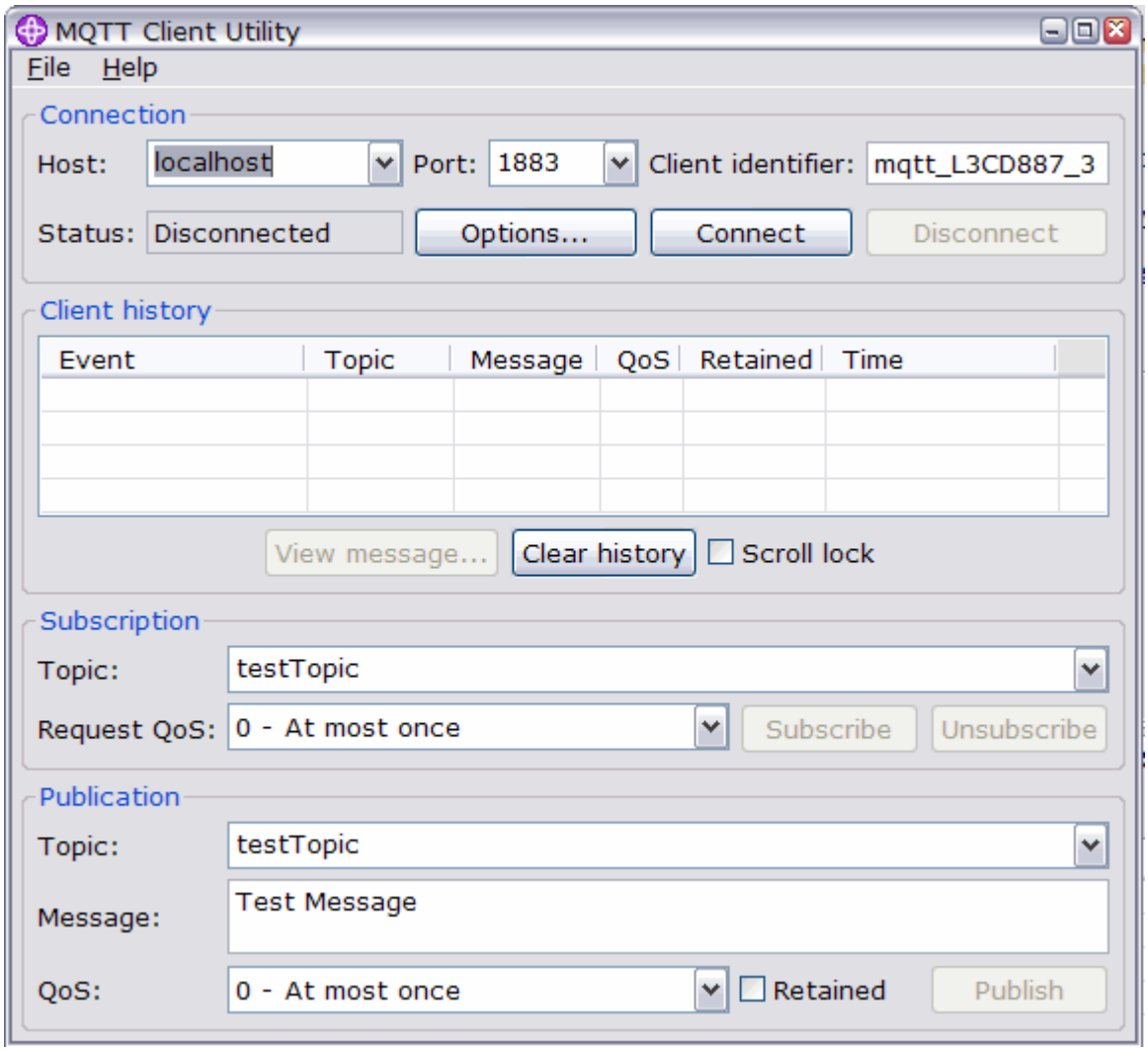
Vous pouvez modifier les propriétés de ce canal, et ajouter et supprimer des canaux dans la fenêtre.

Que faire ensuite

Testez la connexion à l'aide de MQTT Client Utility.

1. Pour démarre l'utilitaire client, ouvrez le dossier **Telemetry** et cliquez deux fois sur **Exécuter l'utilitaire client MQTT**.

   deux fenêtres **Utilitaire client MQTT** s'ouvrent, identiques mais pour des identificateurs client différents.



2. Cliquez sur **Connect** dans les deux fenêtres.
3. Cliquez sur **Subscribe** dans les deux fenêtres.
4. Cliquez sur **Publish** dans l'une des fenêtres. Les résultats sont présentés à la Figure 29, à la page 151.

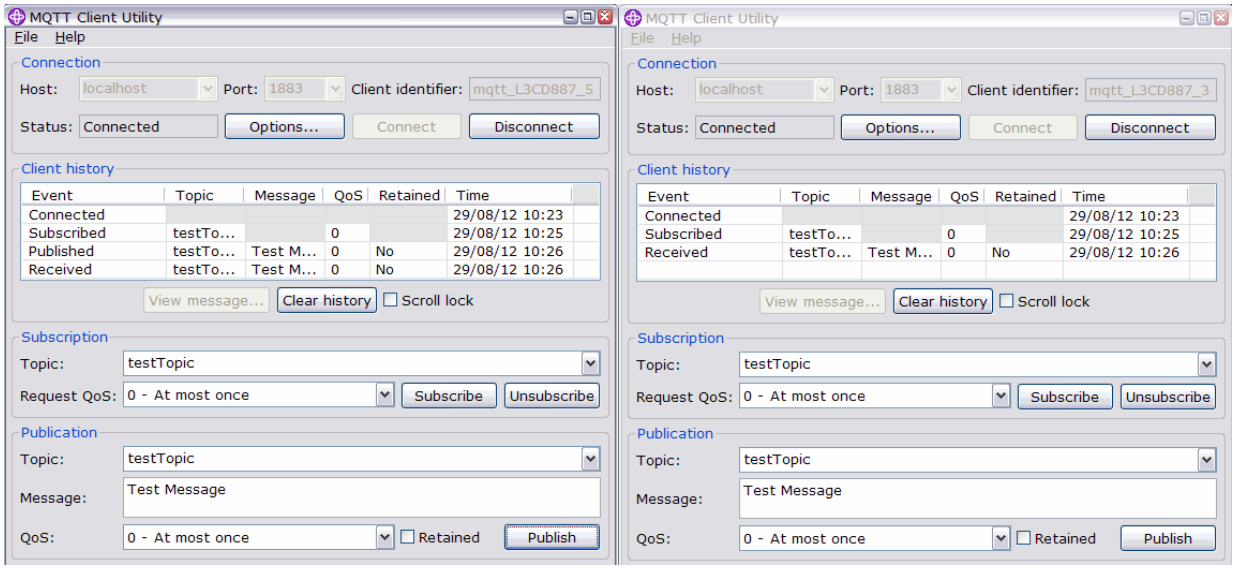


Figure 29. Résultats

5. Cliquez sur **Disconnect** dans les deux fenêtres.

Le serveur est maintenant prêt pour le test de votre application MQTT V3.1 .

Tâches associées

Configuration du service MQTT depuis la ligne de commande

Suivez ces instructions pour configurer IBM WebSphere MQ à l'aide de la ligne de commande en exécutant les modèles d'application IBM WebSphere MQ Telemetry. Les étapes vous montrent comment exécuter un script pour créer un service MQTT sur un nouveau gestionnaire de files d'attente appelé `MQXR_SAMPLE_QM`.

Administration de WebSphere MQ Telemetry

Information associée

WebSphere MQ Telemetry

Administration de WebSphere MQ Telemetry avec WebSphere MQ Explorer

Développement d'applications pour WebSphere MQ Telemetry

Sécurité

Sécurité de WebSphere MQ Telemetry

Concepts du démon pour périphériques IBM WebSphere MQ Telemetry

Le démon pour dispositifs IBM WebSphere MQ Telemetry est une app client avancée de MQTT V3. Utilisez-le pour stocker et transférer des messages à d'autres clients MQTT. Il se connecte à IBM WebSphere MQ comme un client MQTT, mais vous pouvez également y connecter d'autres clients MQTT.

Le démon est un courtier de publication/abonnement. Les clients MQTT V3 se connectent à lui pour publier sur des sujets ou s'y abonner, en utilisant des chaînes de sujet pour la publication, et des filtres de sujet pour les abonnements. La chaîne de sujet est hiérarchique, et les niveaux sont séparés par `/`. Les filtres de sujet sont des chaînes qui peuvent contenir des caractères génériques correspondant à un niveau +, et des caractères génériques correspondant à plusieurs niveaux # dans leur dernière portion.

Remarque : Les caractères génériques dans le démon suivent les règles plus restrictives de WebSphere Message Broker v6. IBM WebSphere MQ est différent. Il accepte plusieurs caractères génériques multiniveaux. Les caractères génériques peuvent représenter un nombre indéfini de niveaux à tout endroit de la chaîne de sujet.

Plusieurs clients MQTT v3 peuvent se connecter au démon à l'aide d'un port d'écoute. Le port d'écoute par défaut est le modifiable. Vous pouvez définir plusieurs ports d'écoute et leur affecter des espaces de noms distincts. Voir «Ports d'écoute du démon pour dispositifs WebSphere MQ Telemetry», à la page 159. Le démon est lui-même un client MQTT v3. Configurez une connexion de pont pour connecter le démon au port d'écoute d'un autre démon, ou à un service WebSphere MQ Telemetry (MQXR).

Vous pouvez configurer plusieurs ponts pour le démon pour dispositifs WebSphere MQ Telemetry. Utilisez les ponts pour connecter un réseau de démons échangeant des publications.

Chaque pont peut diffuser des publications et s'abonner aux sujets sur son démon local. Il peut aussi le faire sur un autre démon, un courtier de publication/abonnement WebSphere MQ, ou un autre courtier MQTT v3 auquel il est connecté. A l'aide d'un filtre de sujet, vous pouvez sélectionner les publications à propager d'un courtier à un autre. Vous pouvez propager les publications dans les deux sens. Vous pouvez propager les publications du démon local vers chacun de ses courtiers distants connectés, ou des courtiers connectés vers le démon local. Voir «Ponts du démon pour périphériques IBM WebSphere MQ Telemetry», à la page 152.

Ponts du démon pour périphériques IBM WebSphere MQ Telemetry

Un pont du démon pour dispositifs IBM WebSphere MQ Telemetry connecte deux courtiers de publication/abonnement à l'aide du protocole MQTT v3. Le pont propage les publications d'un courtier à un autre, dans n'importe quel sens. A l'une des extrémités se trouve la connexion de type pont du démon pour dispositifs WebSphere MQ Telemetry, et à l'autre extrémité peut se trouver un gestionnaire de files d'attente ou un autre démon. Un gestionnaire de files d'attente est connecté à la connexion de type pont par un canal de télémétrie. Un démon est connecté à la connexion de type pont à l'aide d'un programme d'écoute.

Le démon pour dispositifs IBM WebSphere MQ Telemetry prend en charge une ou plusieurs connexions simultanées à d'autres courtiers. Les connexions du démon s'appellent des ponts et sont définies par des entrées dans son fichier de configuration. Les connexions à IBM WebSphere MQ se font à l'aide de canaux de télémétrie IBM WebSphere MQ, comme le montre la figure qui suit :

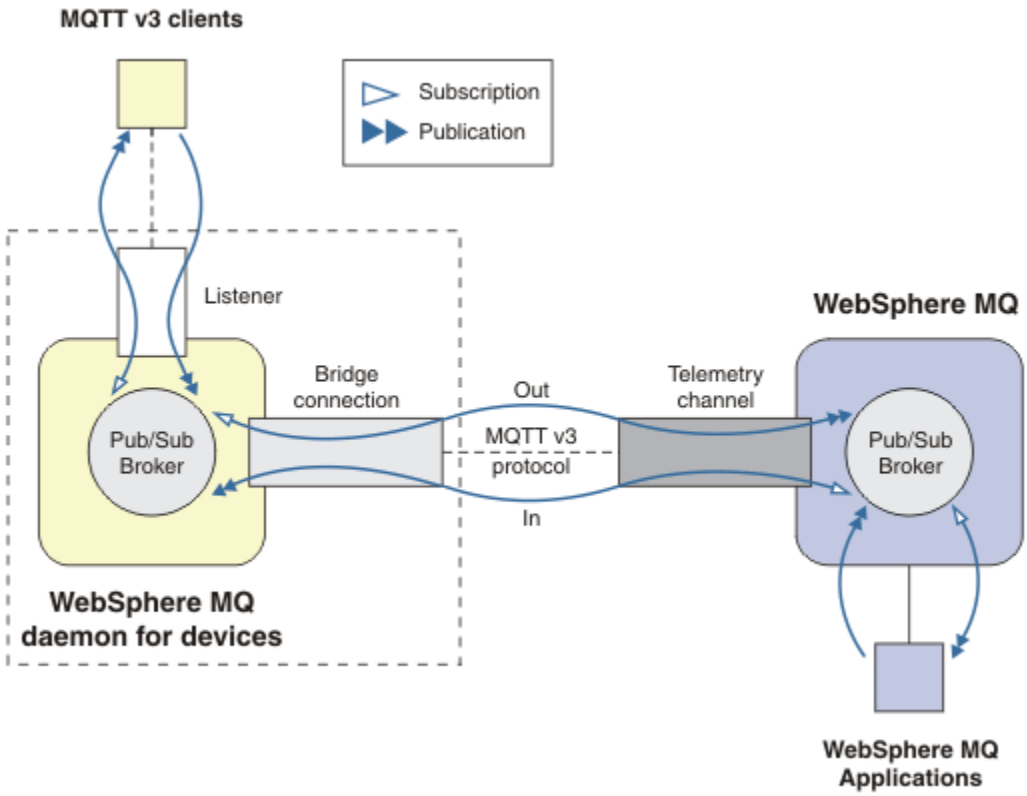


Figure 30. Connexion de IBM WebSphere MQ Telemetry daemon for devices à IBM WebSphere MQ

Un pont connecte le démon à un autre courtier en tant que client MQTT v3. Les paramètres du pont reproduisent les attributs d'un client MQTT v3.

Un pont est plus qu'une connexion. Il agit comme un agent de publication/abonnement situé entre deux courtiers de publication/abonnement. Le courtier local est le démon pour dispositifs IBM WebSphere MQ Telemetry et le courtier distant peut être n'importe quel courtier de publication/abonnement prenant en charge le protocole MQTT v3. Généralement, le courtier distant est un autre démon ou IBM WebSphere MQ.

Le rôle du pont est de propager les publications entre les deux courtiers. Le pont est bidirectionnel. Il propage les publications dans les deux directions. La figure Figure 30, à la page 152 illustre la manière dont le pont connecte le démon pour dispositifs IBM WebSphere MQ Telemetry à IBM WebSphere MQ. La rubrique «Exemples de paramétrage des sujets dans le pont», à la page 153 utilise des exemples pour illustrer l'utilisation du paramètre topic pour configurer le pont.

Les flèches In et Out de la Figure 30, à la page 152 indiquent la bidirectionnalité du pont. A une extrémité de la flèche, un abonnement est créé. Les publications qui lui correspondent sont publiées sur le courtier situé à l'extrémité opposée de la flèche. Le libellé de la flèche dépend du flux des publications. Les publications entrent dans le démon et en sortent. Les libellés sont importants car ils sont utilisés dans la syntaxe de commande. Gardez en mémoire qu'Entrée (In) et Sortie (Out) font référence au flux des publications, et non à la destination de l'abonnement.

D'autres clients, applications ou courtiers peuvent être connectés à IBM WebSphere MQ ou au démon pour dispositifs WebSphere MQ Telemetry. Ils diffusent des publications et s'abonnent aux sujets dans le courtier auquel ils sont connectés. Si le courtier est IBM WebSphere MQ, les sujets peuvent être groupés ou répartis, et non définis de façon explicite au niveau du gestionnaire de files d'attente local.

Utilisation des ponts

Connectez les démons entre eux à l'aide de connexions de type pont et de programmes d'écoute. Connectez les démons et les gestionnaires de files d'attente à l'aide de connexions de type pont et de canaux de télémétrie. Lorsque vous connectez plusieurs courtiers les uns aux autres, il est possible de créer des boucles. Prenez garde : les publications peuvent circuler sans fin sans être détectées le long d'une boucle de courtiers.

La connexion de démons à IBM WebSphere MQ via un pont peut être utilisée pour les raisons suivantes :

Réduire le nombre de connexions de clients MQTT à WebSphere MQ
En utilisant une hiérarchie de démons vous pouvez connecter de nombreux clients à WebSphere MQ. Plus que vous n'en pouvez connecter simultanément avec un unique gestionnaire de files d'attente.

Stocker et retransmettre les messages entre les clients MQTT et WebSphere MQ
Vous pouvez utiliser la fonction de stockage et de retransmission pour éviter de devoir conserver des connexions continues entre les clients et IBM WebSphere MQ, si les clients ne disposent pas de leur propre stockage. Vous pouvez utiliser plusieurs types de connexion entre le client MQTT et WebSphere MQ. Voir Concepts et scénarios de télémétrie pour la surveillance et le contrôle.

Filtrer les publications échangées entre les clients MQTT et WebSphere MQ
Généralement, les publications sont constituée en partie de messages traités en local, tandis que les autres messages impliquent d'autres applications. Les publications locales peuvent inclure des flux de contrôle entre les détecteurs et les actionneurs, et les publications distantes comprennent les demandes de lecture, les statuts et les commandes de configuration.

Modifier les espaces de sujet des publications
Il s'agit d'éviter les collisions entre les chaînes de sujet des clients connectés à différents ports d'écoute. Dans l'exemple, le démon affecte un libellé aux relevés de compteurs provenant des différents bâtiments. Voir Séparation des espaces de sujet des clients connectés à des démons différents.

Exemples de paramétrage des sujets dans le pont

Diffusion de toutes les publications au courtier distant - valeurs par défaut

Le sens par défaut est out (sortie), et le pont publie les sujets sur le courtier distant. Le paramètre topic contrôle les sujets qui sont propagés à l'aide de filtres.

Dans la Figure 31, à la page 154, le pont utilise le paramètre topic pour s'abonner à tout ce qui est publié sur le démon local par les clients MQTT, ou par d'autres courtiers. Il publie les sujets sur le courtier distant connecté par le pont.

```
connection Daemon1
topic #
```

Figure 31. Diffusion de toutes les publications au courtier distant

Diffusion de toutes les publications au courtier distant - valeurs explicites

Le paramètre topic dans le fragment de code suivant donne le même résultat que les valeurs par défaut. Le seule différence est que le paramètre **direction** est explicite. Utilisez le sens `out` (sortie) pour souscrire un abonnement au courtier local (le démon), et diffuser des publications sur le courtier distant. Les publications créées sur le démon local auquel le pont est abonné sont publiées sur le courtier distant.

```
connection Daemon1
topic # out
```

Figure 32. Diffusion de toutes les publications au courtier distant - valeurs explicites

Diffusion de toutes les publications au courtier local

Au lieu d'utiliser le sens `out` (sortie), vous pouvez définir le sens opposé, `in` (entrée). Le fragment de code suivant configure l'abonnement du pont à tout ce qui est publié sur le courtier distant connecté par le pont. Le pont publie les sujets sur le courtier local (le démon).

```
connection Daemon1
topic # in
```

Figure 33. Diffusion de toutes les publications au courtier local

Diffusion de toutes les publications du sujet export du courtier local dans le sujet import du courtier distant

Deux paramètres supplémentaires, **local_prefix** et **remote_prefix**, permettent de modifier le filtre de sujets, `#` dans les exemples suivants. L'un des paramètres sert à modifier le filtre de sujets utilisé dans l'abonnement, l'autre sert à modifier le sujet dans lequel la publication est diffusée. Le résultat recherché est de remplacer le début de la chaîne de sujet utilisée dans un courtier par une chaîne de sujet différente dans un autre courtier.

Selon le sens de la commande topic, la signification de **local_prefix** et de **remote_prefix** s'inverse. Si le sens est `out`, le paramètre par défaut **local_prefix** est utilisé pour l'abonnement au sujet, tandis que **remote_prefix** remplace le paramètre **local_prefix** de la chaîne de sujet dans la publication distante. Si le sens est `in`, **remote_prefix** est inséré dans l'abonnement distant, et **local_prefix** remplace le paramètre **remote_prefix** dans la chaîne de sujet.

La portion initiale d'une chaîne de sujet est souvent considérée comme l'élément de définition d'un espace de sujet. Les paramètres supplémentaires permettent de modifier l'espace dans lequel le sujet est publié. Vous pouvez ainsi empêcher les collisions potentielles entre le sujet propagé et un autre sujet sur le courtier cible, ou pour retirer une chaîne de sujet d'un point de montage.

Par exemple, dans le fragment de code suivant, toutes les publications diffusées dans la chaîne de sujet `export/#` dans le démon, sont republiées dans `import/#` dans le courtier distant.

```
topic # out export/ import/
```

Figure 34. Diffusion de toutes les publications du sujet export du courtier local dans le sujet import du courtier distant

Diffusion de toutes les publications du sujet export du courtier distant dans le sujet import du courtier local

Le fragment de code suivant montre la configuration inverse. Le pont s'abonne à tout ce qui est publié sur le serveur distant avec la chaîne de sujet `export/#`, et le publie sur le courtier local dans le sujet `import/#`.

```
connection Daemon1
topic # in import/ export/
```

Figure 35. Diffusion de toutes les publications du sujet export du courtier distant dans le sujet import du courtier local

Diffusion dans le courtier distant de toutes les publications du point de montage 1884/ avec les chaînes de sujet d'origine

Dans le fragment de code qui suit, le pont s'abonne à tout ce qui est publié par les clients connectés au point de montage 1884/ dans le démon local. Il diffuse au courtier distant toutes les publications du point de montage. La chaîne de point de montage 1884/ est retirée des sujets publiés sur le serveur distant. Le paramètre *local_prefix* est identique à la chaîne de point de montage 1884/, et *remote_prefix* est une chaîne vide.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Figure 36. Diffusion dans le courtier distant de toutes les publications du point de montage 1884/ avec les chaînes de sujet d'origine

Séparation des espaces de sujet des clients connectés à des démons différents

Supposons qu'une application publie les relevés du compteur électrique d'un bâtiment. Les relevés sont diffusés à l'aide de clients MQTT sur un démon hébergé dans le même bâtiment. Le sujet sélectionné pour la publications est `power`. La même application est déployée dans plusieurs bâtiments d'un complexe. Pour la surveillance du site et le stockage des données, les relevés de tous les bâtiments sont rassemblés à l'aide de connexions de type pont. Les connexions relient les démons des bâtiments à WebSphere MQ dans un site central.

Une application client identique est utilisée dans tous les bâtiments. Cette application publie dans le sujet `power`. Toutefois, les données doivent être identifiées par bâtiment. Cette opération est effectuée par le démon pour chaque bâtiment, ajoutant le numéro de bâtiment comme préfixe au nom du sujet. Le pont du premier bâtiment du complexe utilise le préfixe `meters/building01/`, celui du deuxième utilise le préfixe `meters/building02/`. Les relevés des autres bâtiments suivent le même modèle. WebSphere MQ reçoit donc les relevés avec des rubriques telles que `meters/building01/power`.

Le fichier de configuration de chaque démon contient une instruction de rubrique qui suit le modèle du fragment de code suivant :

```
connection Daemon1
topic power out "" meters/building01/
```

Figure 37. Séparation des espaces de sujet des clients connectés à des démons différents

Dans le fragment de code précédent, la chaîne vide est une marque de réservation pour le paramètre non utilisé local_prefix.

Remarque : Cet exemple est fictif et sert uniquement d'illustration. En pratique, l'espace de sujet dans lequel l'application diffuse ses publications est généralement configurable.

Séparation des espaces de sujet des clients connectés au même démon

Supposons qu'un seul démon soit utilisé pour connecter tous les compteurs. Si l'application peut être configurée pour se connecter à différents ports, vous pouvez identifier les bâtiments en connectant le compteur de chaque bâtiment à un port d'écoute distinct, comme dans le fragment de code suivant. L'exemple factice illustre la manière dont des points de montage peuvent être utilisés.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Figure 38. Séparation des espaces de sujet des clients connectés au même démon

Remappage des différents sujets pour les publications transitant dans les deux sens

Dans la configuration du fragment de code suivant, le pont s'abonne à la rubrique unique b sur le courtier distant et transmet les publications relatives à b au démon local, en changeant la rubrique en a. Le pont s'abonne également à la rubrique unique x au niveau du courtier local et transfère les publications relatives à x au courtier distant, en remplaçant la rubrique par y.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Figure 39. Remappage des différents sujets pour les publications transitant dans les deux sens

Il est à noter dans cet exemple que sur les deux courtiers, plusieurs sujets font l'objet d'un abonnement et reçoivent des publications. Sur les deux courtiers, les espaces de sujet sont disjoints.

Remappage des mêmes sujets pour les publications transitant dans les deux sens (boucle)

Contrairement à l'exemple précédent, la configuration de la Figure 40, à la page 157 produit généralement une boucle. Dans l'instruction topic `topic "" in a b`, le pont s'abonne à distance à b, et publie en local sur a. Dans l'autre instruction topic, le pont s'abonne en local à a, et publie à distance sur b. La même configuration peut s'écrire comme à la Figure 41, à la page 157.

Le résultat général est que si un client publie dans b à distance, la publication est transférée au démon local en tant que publication sur la rubrique a. Toutefois, lors de sa publication par le pont sur le démon local sur la rubrique a, la publication correspond à l'abonnement effectué par le pont sur la rubrique locale a. L'abonnement est `topic "" out a b`. Par conséquent, la publication est retransférée au courtier distant en tant que publication dans la rubrique b. Le pont est désormais abonné à la rubrique distante bet le cycle recommence.

Certains courtiers sont dotés d'un mécanisme de détection de boucle pour éviter ces situations. Mais ce mécanisme doit fonctionner lorsque différents types de courtiers sont interconnectés par

des ponts. La détection de boucle ne fonctionne pas si WebSphere MQ est connecté par un pont au démon pour dispositifs WebSphere MQ Telemetry. En revanche, elle fonctionne si deux démons pour dispositifs IBM WebSphere MQ Telemetry sont connectés par un pont. La détection de boucle est activée par défaut. Voir la rubrique relative au paramètre try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Figure 40. !Remappage des mêmes rubriques pour les publications circulant dans les deux sens

```
connection Daemon1
topic "" both a b
```

Figure 41. !Remappez les mêmes rubriques pour les publications circulant dans les deux sens, à l'aide de `both`.

La configuration de la Figure 39, à la page 156 est identique à celle de la Figure 40, à la page 157.

Disponibilité des connexions de type pont du IBM WebSphere MQ Telemetry daemon for devices

Configurez plusieurs adresses de connexion de pont pour le IBM WebSphere MQ Telemetry daemon for devices, pour lui permettre de se connecter au premier courtier distant disponible. Si le courtier est un gestionnaire de files d'attente multi-instance, entrez ses deux adresses TCP/IP. Configurez une connexion principale pour le connecter ou le reconnecter au serveur principal, s'il est disponible.

La paramètre de la connexion de pont addresses est une liste d'adresses de connecteur TCP/IP. Le pont tente de se connecter successivement à chaque adresse, jusqu'à ce qu'il réussisse à établir une connexion. Les paramètres de connexion round_robin et start_type contrôlent la manière dont les adresses sont utilisées une fois que la connexion a été établie.

Si la valeur de start_type est auto, manual ou lazy, le pont tente de se reconnecter en cas de défaillance de la connexion. Il tente d'établir la connexion en suivant la séquence d'adresses, avec un délai de 20 secondes entre chaque tentative. Si la valeur de `start_type` est once, le pont ne tente pas de se reconnecter automatiquement si la connexion tombe.

Si la valeur de round_robin est true, le pont commence ses tentatives de connexion à la première adresse de la liste, puis les poursuit selon l'ordre de la liste. Lorsqu'il arrive à la fin de la liste, il recommence à la première adresse. Si la liste contient une seule adresse, il fait une tentative sur celle-ci toutes les 20 secondes.

Si la valeur de round_robin est false, la première adresse de la liste, qui est appelée le serveur principal, a la préférence. En cas d'échec de la première tentative de connexion au serveur principal, le pont continue d'essayer de s'y connecter en arrière-plan. En même temps, il tente de se connecter avec les autres adresses de la liste. Lorsque la tentative de connexion au serveur principal aboutit, le pont se déconnecte de la connexion active et passe sur la connexion au serveur principal.

Après une déconnexion volontaire, par exemple avec la commande **`connection_stop`**, le pont tente de réutiliser la même adresse au redémarrage de la connexion. Si la connexion est interrompue parce que le pont n'arrive pas à se connecter ou que le courtier distant perd la connexion, le pont attend 20 secondes. Il tente ensuite de se connecter à l'adresse qui suit dans la liste, ou à la même adresse si la liste n'en contient qu'une.

Connexion à un gestionnaire de files d'attente multi-instance

Dans une configuration de gestionnaire de files d'attente multi-instance, le gestionnaire de files d'attente s'exécute sur deux serveurs différents avec des adresses IP distinctes. Les canaux de télémétrie sont

généralement configurés sans adresse IP spécifique. Ils ne sont configurés qu'avec un numéro de port. Au démarrage du canal de télémétrie, celui-ci sélectionne par défaut la première adresse réseau disponible sur le serveur local.

Configurez le paramètre addresses de la connexion de pont avec les deux adresses IP utilisées par le gestionnaire de files d'attente. Affectez à round_robin la valeur true.

En cas de défaillance de l'instance active du gestionnaire de files d'attente, le gestionnaire de files d'attente bascule sur l'instance de secours. Le démon détecte l'arrêt de la connexion à l'instance active et tente de se reconnecter à l'instance de secours. Il utilise la deuxième adresse Ip dans la liste des adresses configurées pour la connexion de pont.

Le pont est toujours connecté au même gestionnaire de files d'attente. Le gestionnaire de files d'attente récupère son état. Si cleansession a la valeur false, la session de la connexion de pont est restaurée à l'état qu'elle avait avant la défaillance. La connexion est rétablie au bout d'un certain temps. Des messages avec la qualité de service "au moins une fois" ou "au plus une fois" ne sont pas perdus et les abonnements continuent à fonctionner.

La durée de la reconnexion dépend du nombre de canaux et de clients qui redémarrent au démarrage de l'instance de secours, et au nombre de messages en cours. Le pont peut faire un certain nombre de tentatives sur les deux adresse IP avant de réussir à rétablir la connexion.

Ne configurez pas le canal de télémétrie d'un gestionnaire de files d'attente multi-instance avec une adresse IP spécifique. L'adresse IP n'est valide que sur un serveur.

La configuration d'un canal de télémétrie avec une adresse IP spécifique peut être souhaitable si vous utilisez une autre solution à haute disponibilité qui gère l'adresse IP.

cleansession

Une connexion de pont est une session du client MQTT v3. Vous pouvez choisir si la connexion démarre une nouvelle session ou si elle en restaure une existante. Si elle restaure une session existante, la connexion de pont préserve les abonnements et les publications conservées de la session précédente.

N'affectez pas à cleansession la valeur si addresses contient plusieurs adresses IP, et si les adresses IP établissent des connexions à des canaux de télémétrie hébergés par différents gestionnaires de files d'attente, ou à différents démons de télémétrie. L'état de la session n'est pas transféré entre les gestionnaires de files d'attente et les démons. Une tentative de redémarrage d'une session existante sur un autre gestionnaire de files d'attente ou un autre démon aboutit au lancement d'une nouvelle session. Les messages en attente de validation sont perdus, et les abonnements peuvent se comporter de façon inattendue.

notifications

Les notifications permettent aux applications de savoir si la connexion de pont est active. Une notification est une publication dont la valeur est `1` (connecté), ou `0` (déconnecté). Elle est publiée dans la *chaîne de sujet* définie par le paramètre `notification_topic`. La valeur par défaut de *topicString* est $SYS/`broker/connection/` *clientIdentifier*`/state`. La *chaîne de sujet* par défaut contient le préfixe $SYS. Créez un abonnement aux sujets commençant par $SYS en définissant un filtre de sujet commençant par $SYS. Le sujet de filtre # crée sur le démon un abonnement à tout, sauf aux sujets commençant par $SYS. Voyez $SYS comme définissant un espace de sujet particulier, réservé au système et distinct de l'espace de sujet de l'application.

Les notifications permettent à IBM WebSphere MQ Telemetry daemon for devices de notifier les clients MQTT lorsqu'un pont est connecté ou déconnecté.

keepalive_interval

Le paramètre de connexion de pont keepalive_interval définit l'intervalle entre les commandes ping TCP/IP envoyées par le pont au serveur distant. L'intervalle par défaut est 60 secondes. La commande ping empêche le serveur distant ou un pare-feu détectant une période d'inactivité de fermer la session TCP/IP.

clientid

Une connexion de pont est une session client MQTT v3 dont le paramètre `clientIdentifier` est défini par le paramètre de connexion de pont clientid. Si vous souhaitez que les reconnexions reprennent une session précédente en affectant au paramètre cleansession la valeur false, l'attribut `clientIdentifier` utilisé dans chaque session doit être le même. La valeur par défaut de clientid est *hostname.connectionName* et reste la même.

Installation, vérification, configuration et contrôle du démon pour dispositifs WebSphere MQ Telemetry

L'installation, la vérification, la configuration et le contrôle du démon se font à partir de fichiers.

Installez le démon en copiant le kit de développement de logiciels sur l'appareil sur lequel il doit être exécuté.

Par exemple, exécutez MQTT Client Utility et connecte-vous au démon pour dispositifs WebSphere MQ Telemetry en tant que courtier de publication/abonnement. Voir Utilisation du démon pour dispositifs WebSphere MQ Telemetry en tant que le courtier de publication/abonnement.

Configurez le démon en créant un fichier de configuration. Voir Fichier de configuration du démon pour dispositifs WebSphere MQ.

Contrôlez un démon actif en créant des commandes dans le fichier `amqtdd.upd`. Toutes les 5 secondes, le démon lit le fichier, exécute les commandes et supprime le fichier. Voir Fichier de commandes du démon pour dispositifs WebSphere MQ Telemetry.

Ports d'écoute du démon pour dispositifs WebSphere MQ Telemetry

Connectez les clients MQTT V3 au démon pour dispositifs WebSphere MQ Telemetry à l'aide des ports d'écoute. Un point de montage et un nombre maximal de connexions peuvent être définis pour les ports d'écoute.

Un port d'écoute doit correspondre au numéro de port défini dans la méthode `connect(serverURI)` du client MQTT qui s'y connecte. Sur le client, comme sur le démon, sa valeur par défaut est 1883.

Vous pouvez modifier le port par défaut du démon en définissant le paramètre global `port` dans son fichier de configuration. Vous pouvez définir des ports spécifiques en ajoutant une définition `listener` au fichier de configuration du démon.

Pour chaque port d'écoute autre que le port par défaut, vous pouvez définir un point de montage pour isoler les clients. Les clients connectés à un port avec un point de montage sont isolés des autres clients. Voir «Points de montage du démon pour dispositifs WebSphere MQ Telemetry», à la page 160.

Vous pouvez limiter le nombre de clients qui peuvent se connecter à un port. Utilisez la définition globale `max_connections` pour limiter les connexions au port par défaut, ou qualifier caque port d'écoute à l'aide de `max_connections`.

Exemple

Exemple de fichier de configuration qui remplace le port par défaut 1883 par 1880 et limite à 10000 le nombre de connexions au port 1880. Le nombre de connexions au port 1884 est limité à 1000. Les clients connectés au port 1884 sont isolés de ceux qui sont connectés aux autres ports.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Points de montage du démon pour dispositifs WebSphere MQ Telemetry

Vous pouvez associer un point de montage à un port d'écoute utilisé par les clients MQTT pour les connecter à un démon pour dispositifs WebSphere MQ Telemetry. Un point de montage isole les publications et les abonnements échangés par les clients MQTT utilisant un port d'écoute des clients MQTT connectés à un autre port d'écoute.

Les clients connectés à un port d'écoute avec un point de montage ne peuvent jamais échanger directement des sujets avec les clients connectés à d'autres ports d'écoute. Les clients connectés à un port d'écoute sans point de montage peuvent diffuser des publications ou créer des abonnements sur les sujets de tous les clients. Les clients ne détectent pas qu'ils sont connectés par un point de montage. Cela n'a pas d'incidence sur les chaînes de sujet qu'ils créent.

Un point de montage est une chaîne de texte qui est ajoutée en préfixe aux chaînes de sujet des publications et des abonnements. Il est ajouté en préfixe à toutes les chaînes de sujet créées par les clients connectés au port d'écoute avec un point de montage. Cette chaîne est retirée de toutes les chaînes de sujet envoyées aux clients connectés au port d'écoute.

Si un port d'écoute n'a pas de point de montage, les chaînes de sujet des publications et des abonnements créés et reçus par les clients connectés au port ne sont pas modifiées.

Ajoutez / à la fin des chaînes de point de montage que vous créez. De cette manière, le point de montage est le sujet parent de l'arborescence de sujets pour le point de montage.

Exemple

Un fichier de configuration contient les ports d'écoute suivants :

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Un client, connecté au port 1883, crée un abonnement à `MyTopic`. Le démon enregistre l'abonnement comme `1883/MyTopic`. Un autre client connecté au port 1883 publie un message sur le sujet `MyTopic`. Le démon remplace la chaîne de sujet par `1883/MyTopic` et recherche des abonnements correspondants. L'abonné du port 1883 reçoit la publication avec sa chaîne de sujet d'origine `MyTopic`. Le démon a retiré le préfixe du point de montage de la chaîne de sujet.

Un autre client, connecté au port 1884, publie également sur le sujet `MyTopic`. Cette fois, le démon enregistre le sujet comme `1884/MyTopic`. L'abonné du port 1883 ne reçoit pas la publication, car différents points de montage aboutissent à un abonnement avec des chaînes de sujet différentes.

Un client, connecté au port 1885, publie sur le sujet `1883/MyTopic`. Le démon ne modifie pas la chaîne de sujet. L'abonné du port 1883 reçoit la publication sur le sujet `MyTopic`.

Qualité du service du démon pour dispositifs WebSphere MQ Telemetry, abonnements durables et publications conservées

Les paramètres de qualité du service s'appliquent à un seul démon en cours d'exécution. Si le démon s'arrête, soit de manière contrôlée, ou à cause d'une défaillance, l'état des messages en cours est perdu. La distribution d'un message au moins une fois, ou au plus une fois, ne peut pas être garantie en cas d'arrêt du démon. Le démon pour dispositifs WebSphere MQ Telemetry prend en charge une persistance limitée. Pour enregistrer les publications conservées et les abonnements en cas d'arrêt du démon, définissez le paramètre de configuration **`retained_persistence`**.

Contrairement à WebSphere MQ, le démon pour dispositifs WebSphere MQ Telemetry ne consigne pas les données persistantes. Ni l'état de la session et des messages, ni les publications conservées ne sont enregistrés dans les transactions. Par défaut, le démon supprime toutes les données lorsqu'il s'arrête. Une option permet de faire des points de contrôle réguliers sur les abonnements et les publications conservées. Le statut des messages est toujours perdu à l'arrêt du démon. Toutes les publications non conservées le sont aussi.

Définissez l'option de configuration du démon `Retained_persistence` avec la valeur `true` pour enregistrer périodiquement dans un fichier les publications conservées. Au redémarrage du démon, le dernier enregistrement automatique des publications conservées est rétabli. Par défaut, les messages conservés créés par les clients ne sont pas rétablis au redémarrage du démon.

Définissez l'option de configuration du démon `Retained_persistence` avec la valeur `true` pour enregistrer périodiquement dans un fichier les abonnements créés dans une session persistante. Si `Retained_persistence` a la valeur `true`, les abonnements créés par les clients dans une session dans laquelle `CleanSession` a la valeur `false`, donc une "session persistante", sont restaurés. Le démon restaure les abonnements lorsqu'il redémarre, ce qui déclenche la réception des publications. Le client reçoit les publications lorsqu'il redémarre avec `CleanSession` et la valeur `false`. Par défaut, l'état de la session client n'est pas enregistré à l'arrêt d'un démon. Les abonnements ne sont donc pas restaurés, même si le client définit `CleanSession` à la valeur `false`.

`Retained_persistence` est un mécanisme d'enregistrement automatique. Il n'enregistre pas forcément les dernières publications conservées ou les derniers abonnements. Vous pouvez modifier la fréquence d'enregistrement des publications conservées et des abonnements. Définissez l'intervalle ou le nombre de modifications entre les enregistrements, à l'aide des options de configuration `autosave_on_changes` et `autosave_interval`.

Exemple de configuration de la persistance

```
# 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
```

Sécurité du démon pour dispositifs WebSphere MQ Telemetry

Le démon pour dispositifs WebSphere MQ Telemetry peut authentifier les clients qui se connectent à lui, utiliser des données d'identification pour se connecter à d'autres courtiers et contrôler l'accès aux sujets. La sécurité fournie par le démon est limitée par le fait qu'elle est construite à l'aide du client WebSphere MQ Telemetry pour C, qui ne prend pas en charge SSL. En conséquence, les connexions en entrée et en sortie du démon ne sont pas chiffrées, et ne peuvent pas être authentifiées à l'aide de certificats.

Par défaut, aucune sécurité n'est activée.

Authentification des clients

Les clients MQTT peuvent définir un nom d'utilisateur et un mot de passe à l'aide des méthodes `MqttConnectOptions.setUserName` et `MqttConnectOptions.setPassword`.

Authentifiez un client qui se connecte au démon en vérifiant le nom d'utilisateur et le mot de passe qu'il fournit par rapport aux entrées du fichier de mots de passe. Pour activer l'authentification, créez un fichier de mots de passe et définissez le paramètre `password_file` dans le fichier de configuration du démon. Voir password_file.

Définissez la paramètre `allow_anonymous` dans le fichier de configuration du démon pour permettre aux clients qui se connectent sans nom d'utilisateur et mot de passe de se connecter à un démon qui vérifie l'authentification. Voirallow_anonymous. Lorsqu'un client fournit un nom d'utilisateur et un mot de passe, celui-ci est toujours vérifié dans le fichier de mots de passe si le paramètre `password_file` est défini.

Définissez le paramètre `clientid_prefixes` dans le fichier de configuration du démon pour limiter la connexion à des clients spécifiques. Les clients doivent avoir un `clientIdentifiers` commençant par l'un des préfixes listés dans le paramètre `clientid_prefixes`. Voir clientid_prefixes.

Sécurité de la connexion de pont

Chaque connexion de type pont du démon pour dispositifs WebSphere MQ Telemetry est un client MQTT V3. Vous pouvez définir le nom d'utilisateur et le mot de passe de chaque connexion de pont en tant que paramètre dans le fichier de configuration du démon. Voir username etpassword. Le pont peut alors s'authentifier auprès du courtier.

Contrôle d'accès des sujets

Si les clients sont authentifiés, le démon peut également fournir le contrôle d'accès aux sujets pour chaque utilisateur. Il accorde l'accès sur la base de la correspondance entre le sujet sur lequel le client diffuse une publication ou crée un abonnement et une chaîne de sujet du fichier de contrôle d'accès. Voir acl_file.

La liste de contrôle d'accès contient deux parties. La première partie contrôle l'accès de tous les clients, y compris celui des clients anonymes. La seconde partie contient une section réservée aux utilisateurs du fichier de mots de passe. Elle liste les accès spécifiques des utilisateurs individuels.

Exemple

L'exemple qui suit montre les paramètres de sécurité.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

Figure 42. Fichier de configuration du démon

```
Fred:Fredpassword
Barney:Barneypassword
```

Figure 43. Fichier de mot de passe 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/#
```

Figure 44. Fichier de contrôle d'accès acl.txt

Traitement des incidents liés aux clients MQTT

Cherchez une tâche de traitement des incidents pour vous aider à résoudre un problème lié à l'exécution des clients MQTT.

Tâches associées

«Traçage et débogage du client Java MQTT (Paho)», à la page 171
Le consignateur par défaut utilise la fonction de journalisation Java standard appelée `java.util.logging` (JSR47). Vous pouvez la configurer à l'aide d'un fichier de configuration ou à l'aide d'un programme.

«Traçage du client JavaScript MQTT», à la page 174

Vous pouvez utiliser le client JavaScript pour collecter une trace en modifiant l'application Web client pour appeler des méthodes sur l'objet client connecté.

«Traçage du service de télémétrie (MQXR)», à la page 167
Suivez ces instructions pour démarrer le traçage du service de télémétrie, définir les paramètres qui contrôlent la trace et trouver la sortie de trace.

«Traçage du client Java MQTT v3», à la page 168
Suivez ces instructions pour créer une trace du client Java MQTT Java et contrôler sa sortie.

«Traçage du client MQTT pour C», à la page 170
Définissez la variable d'environnement `MQTT_C_CLIENT_TRACE` pour tracer une app C client MQTT.

«Résolution des problèmes : Le client MQTT ne se connecte pas», à la page 182
Identifiez et corrigez la raison pour laquelle un programme client MQTT ne se connecte pas au service de télémétrie (MQXR).

«Résolution des problèmes : La connexion du client MQTT a été supprimée», à la page 184
Identifiez la raison pour laquelle un client envoie des exceptions `ConnectionLost` inattendues après s'être connecté et exécuté pendant une durée courte ou longue.

«Résolution des problèmes : Messages perdus dans une application MQTT», à la page 185
Résolution du problème de la perte de message. Le message est-il non persistant, a-t-il été envoyé au mauvais emplacement, n'a-t-il jamais été envoyé ? Un programme client codé de manière erronée peut perdre des messages.

«Résolution des problèmes : Le service de télémétrie (MQXR) ne démarre pas», à la page 186
Identifiez et corrigez la raison pour laquelle le service de télémétrie (MQXR) ne démarre pas. Vérifiez l'installation WebSphere MQ Telemetry. Vérifiez notamment qu'aucun fichier ne manque, n'a été déplacé ou ne dispose de droits non appropriés. Vérifiez les chemins utilisés par le service de télémétrie (MQXR) pour localiser ses programmes.

«Résolution des problèmes : Le module de connexion JAAS n'est pas appelé par le service de télémétrie», à la page 188
Vérifiez si le module de connexion JAAS n'est pas appelé par le service de télémétrie (MQXR), et configurez JAAS pour corriger le problème.

«Résolution des problèmes : Démarrage ou exécution du démon», à la page 191
Consultez le journal de la console du démon pour dispositifsIBM WebSphere MQ Telemetry, activez la trace ou utilisez le tableau des symptômes de la présente rubrique pour traiter les incidents liés au démon.

«Résolution des problèmes : Les clients MQTT ne se connectent pas au démon», à la page 191
Les clients ne se connectent pas au démon, ou le démon ne se connecte pas aux autres démons ou à un canal de télémétrie WebSphere MQ.

Référence associée

«Emplacement des journaux de télémétrie, journaux des erreurs et fichiers de configuration», à la page 164
Localisez l'emplacement des journaux de télémétrie, des journaux des erreurs et des fichiers de configuration utilisés par IBM WebSphere MQ Telemetry.

«Codes anomalie du client Java MQTT v3», à la page 166
Recherchez les causes des codes anomalie dans une exception de client Java MQTT v3 ou une exception throwable.

«Configuration système requise pour l'utilisation des suites de chiffrement SHA-2 avec les clients MQTT», à la page 175
Pour Java 6 à partir de IBM, SR13 et versions ultérieures, vous pouvez utiliser des suites de chiffrement SHA-2 pour sécuriser vos canaux MQTT et vos applications client. Toutefois, les suites de chiffrement SHA-2 ne sont pas activées par défaut jusqu'à Java 7 à partir de IBM, SR4 . Par conséquent, dans les versions antérieures, vous devez spécifier la suite requise. Si vous exécutez un client MQTT avec votre propre JRE, vous devez vous assurer qu'il prend en charge les suites de chiffrement SHA-2. Pour que vos applications client utilisent les suites de chiffrement SHA-2, le client doit aussi associer le contexte SSL à une valeur qui prend en charge Transport Layer Security (TLS) version 1.2.

«Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL», à la page 176
Les différents navigateurs, combinés aux différentes plateformes, présentent des fonctionnalités différentes. La compréhension de ces différences vous aidera à configurer vos applications, autorités de certification et certificats client pour une connexion à l'aide du client de messagerie MQTT pour JavaScript sur SSL et WebSockets.

Emplacement des journaux de télémétrie, journaux des erreurs et fichiers de configuration

Localisez l'emplacement des journaux de télémétrie, des journaux des erreurs et des fichiers de configuration utilisés par IBM WebSphere MQ Telemetry.

Remarque : Les exemples sont codés pour Windows. Modifiez la syntaxe pour exécuter les exemples sur Linux

Journaux côté serveur

L'assistant d'installation d'IBM WebSphere MQ Telemetry écrit des messages dans son journal d'installation :

```
WMQ program directory\mqxr
```

Le service de télémétrie (MQXR) écrit des messages dans un journal des erreurs de gestionnaire de files d'attente WebSphere MQ et les fichiers FDC, dans le répertoire des erreurs d'IBM WebSphere MQ :

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

Il écrit également un journal pour le service de télémétrie (MQXR). Le journal affiche les propriétés du service avec lequel il a démarré et les erreurs qu'il a trouvées en agissant comme un proxy pour un client MQTT. Par exemple, l'annulation d'un abonnement que le client n'a pas créé. Le chemin d'accès au journal est :

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

Le modèle de configuration de télémétrie IBM WebSphere MQ créé par IBM WebSphere MQ Explorer démarre le service de télémétrie à l'aide de la commande **runMQXRService**. **runMQXRService** se trouve dans *WMQ Telemetry install directory*\bin. Il écrit dans :

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Modifiez **runMQXRService** pour afficher les chemins configurés pour le service de télémétrie (MQXR) ou pour répercuter l'initialisation avant le démarrage du service de télémétrie (MQXR).

Fichiers de configuration côté serveur

Canaux de télémétrie et service de télémétrie (MQXR)

Restriction : Le format, l'emplacement, le contenu et l'interprétation du fichier de configuration de télémétrie peut changer dans les éditions à venir. Vous devez utiliser IBM WebSphere MQ Explorer pour configurer les canaux de télémétrie.

IBM WebSphere MQ Explorer sauvegarde les configurations de télémétrie dans le fichier `mqxr_win.properties` sous Windowset dans le fichier `mqxr_unix.properties` sous Linux. Les fichiers de propriétés sont sauvegardés dans le répertoire de configuration de la télémétrie :

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Figure 45. Répertoire de configuration de la télémétrie sous Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figure 46. Répertoire de configuration de la télémétrie sous Linux

Machine virtuelle Java

Définissez les propriétés Java qui sont transmises en tant qu'arguments au service de télémétrie (MQXR) dans le fichier `java.properties`. Ces propriétés sont transmises directement à la machine virtuelle Java qui exécute le service de télémétrie (MQXR). Elles sont transmises en tant que propriétés JVM supplémentaires sur la ligne de commande Java. Les propriétés définies via la ligne de commande sont prioritaires sur celles qui ont été ajoutées à partir du fichier `java.properties`.

Recherchez le fichier `java.properties` dans le même dossier que les configurations de télémétrie. Voir Figure 45, à la page 164 et Figure 46, à la page 165.

Modifiez le fichier `java.properties` en définissant chaque propriété sur une ligne distincte. Formatez chaque propriété exactement comme si vous souhaitiez la transmettre en tant qu'argument à la machine virtuelle. Par exemple :

```
-Xmx1024m
-Xms1024m
```

JAAS

Le fichier de configuration JAAS est décrit dans la section Configuration JAAS du canal de télémétrie, qui comprend le modèle de fichier de configuration JAAS JAAS.config livré avec IBM WebSphere MQ Telemetry.

Si vous configurez un service JAAS, vous allez probablement devoir écrire une classe afin d'authentifier les utilisateurs pour remplacer les procédures d'authentification JAAS standard.

Pour inclure votre classe `Login` au chemin d'accès aux classes utilisé par le service de télémétrie (MQXR), indiquez un fichier de configuration WebSphere MQ `service.env`.

Définissez le chemin d'accès aux classes pour votre `LoginModule` JAAS dans `service.env`. Vous ne pouvez pas utiliser la variable `%classpath%` dans `service.env`. Le chemin d'accès aux classes de `service.env` est ajouté au chemin d'accès aux classes déjà défini dans la définition du service de télémétrie (MQXR).

Affichez les chemins d'accès aux classes utilisés par le service de télémétrie (MQXR) en ajoutant `echo set classpath` au fichier `runMQXRService.bat`. La sortie est envoyée à `mqxr.stdout`.

L'emplacement par défaut du fichier `service.env` est :

```
WMQ data directory\service.env
```

Redéfinissez ces paramètres en utilisant un fichier `service.env` pour chaque gestionnaire de files d'attente dans :

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

La Figure 47, à la page 165 montre un exemple de fichier `service.env` pour utiliser l'exemple `LoginModule.class`.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Remarque : `service.env` ne doit pas contenir de variables. Remplacez la valeur réelle de *`WMQ Install Directory`*.

Figure 47. Exemple `service.env` pour Windows

Fonction de trace
Un ingénieur service IBM peut vous demander de configurer la fonction de trace ; voir «Traçage du service de télémétrie (MQXR)», à la page 167. Les paramètres de configuration de la fonction de trace sont stockés dans deux fichiers :

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Fichier journaux côté client

La classe de persistance fichier par défaut dans le client Java SE MQTT fourni avec IBM WebSphere MQ Telemetry crée un dossier portant le nom: *clientIdentifier*-tcp*hostNameport* ou *clientIdentifier*-ssl*hostNameport* dans le répertoire de travail du client. Le nom du dossier indique le nom d'hôte et le port utilisés lors de la tentative de connexion.: Le dossier contient les messages qui ont été stockés par la classe de persistance. Les messages sont supprimés lorsque leur distribution a abouti.

Le dossier est supprimé lorsqu'un client, avec une session propre, se termine.

Si la fonction de trace du client est activée, le journal non formaté est, par défaut, stocké dans le répertoire de travail du client. Le fichier de trace est appelé mqtt-*n*.trc

Fichier de configuration côté client

Définissez les propriétés de trace et SSL pour le client Java MQTT à l'aide des fichiers de propriétés Java ou définissez les propriétés à l'aide d'un programme. Transmettez les propriétés au client Java MQTT à l'aide du commutateur -D de la machine virtuelle Java: par exemple,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Voir «Traçage du client Java MQTT v3», à la page 168.Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

Codes anomalie du client Java MQTT v3

Recherchez les causes des codes anomalie dans une exception de client Java MQTT v3 ou une exception throwable.

Tableau 5. Codes anomalie du client Java MQTT v3

Code raison	Valeur	Raison
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	Le client est déjà connecté.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	Le client est déjà déconnecté.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Emis lorsqu'une tentative d'appel de MqttClient.disconnect a été effectuée au sein d'une méthode sur MqttCallback.
REASON_CODE_CLIENT_DISCONNECTING	32102	Le client est en cours de déconnexion et ne peut pas accepter de nouveau travail.
REASON_CODE_CLIENT_EXCEPTION	0	Le client a rencontré une exception.

Tableau 5. Codes anomalie du client Java MQTT v3 (suite)		
Code raison	**Valeur**	**Raison**
REASON_CODE_CLIENT_NOT_CONNECTED	32104	Le client n'est pas connecté au serveur.
REASON_CODE_CLIENT_TIMEOUT	32000	Le client a dépassé le délai d'attente alors qu'il attendait une réponse du serveur
REASON_CODE_FAILED_AUTHENTICATION	4	L'authentification avec le serveur a échoué en raison d'un nom d'utilisateur ou d'un mot de passe incorrect.
REASON_CODE_INVALID_CLIENT_ID	2	Le serveur a rejeté l'ID fourni par le client
REASON_CODE_INVALID_PROTOCOL_VERSION	1	La version de protocole demandée n'est pas prise en charge par le serveur.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Erreur interne provoquée par la non disponibilité de nouvel ID message.
REASON_CODE_NOT_AUTHORIZED	5	Non autorisé à effectuer cette opération de demande.
REASON_CODE_SERVER_CONNECT_ERROR	32103	Impossible d'établir la connexion au serveur.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	L'URI du serveur et le SocketFactory fourni ne correspondent pas.
REASON_CODE_SSL_CONFIG_ERROR	32106	Erreur de configuration SSL.
REASON_CODE_UNEXPECTED_ERROR	6	Une erreur inattendue s'est produite.

Traçage du service de télémétrie (MQXR)

Suivez ces instructions pour démarrer le traçage du service de télémétrie, définir les paramètres qui contrôlent la trace et trouver la sortie de trace.

Avant de commencer

Le traçage est une fonction de prise en charge. Suivez ces instructions si un ingénieur service IBM vous demande de tracer votre service de télémétrie (MQXR). La documentation du produit ne fournit pas d'informations sur le format du fichier de trace, ni sur son utilisation pour déboguer un client.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser les commandes IBM WebSphere MQ **strmqtrc** et **endmqtrc** pour démarrer et arrêter la trace IBM WebSphere MQ . La commande **strmqtrc** capture la trace pour le service de télémétrie (MQXR). Lorsque vous utilisez la commande **strmqtrc**, un délai pouvant atteindre deux secondes s'écoule avant le démarrage de la trace de ce service. Pour plus d'information sur la trace IBM WebSphere MQ, voir Utilisation de la trace. Vous pouvez également tracer le service de télémétrie (MQXR) à l'aide de la procédure suivante :

Procédure

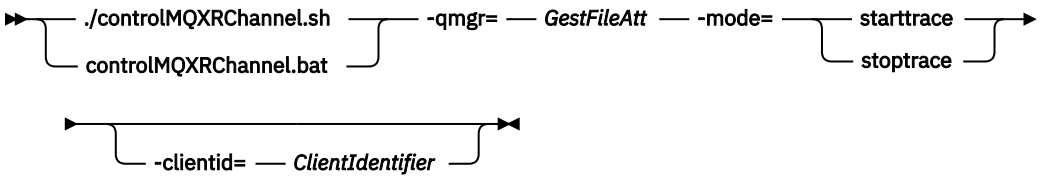
1. Définissez les options de trace destinées à contrôler le nombre de détails et la taille de la trace. Ces options s'appliquent à une trace démarrée via la commande **strmqtrc** ou la commande **controlMQXRChannel**.

   Définissez les options de trace dans les fichiers suivants :

   ```
   mqxrtrace.properties
   trace.config
   ```

   Les fichiers se trouvent dans ce répertoire :

   - Sous Windows, *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr.
   - Sous Linux, var/mqm/qmgrs/ *qMgrName*/mqxr.
2. Ouvrez une fenêtre de commande dans le répertoire suivant :
   - Sur les systèmes Windows , *WebSphere MQ installation directory*\mqxr\bin .
   - Sur les systèmes Linux /opt/mqm/mqxr/bin.
3. Exécutez la commande suivante pour démarrer une trace SYSTEM.MQXR.SERVICE :



   Paramètres obligatoires

   qmgr=*qmgrName*
   NomGestFileAtt est le nom du gestionnaire de files d'attente

   mode=starttrace| stoptrace
   Définissez starttrace pour démarrer le traçage ou stoptrace pour l'arrêter.

   Paramètres optionnels

   clientid=*ClientIdentifier*
   Associez l'*identificateur client* à l'identificateur client d'un client. clientid filtre la trace sur un client unique. Pour tracer plusieurs clients, exécutez la commande de trace plusieurs fois.

   Exemple :

   ```
   /opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
   problemclient
   ```

Résultats

Pour afficher la sortie de la trace, passez dans le répertoire suivant :

- Sous Windows , *WebSphere MQ data directory*\trace.
- Sous Linux : /var/mqm/trace.

Les fichiers de trace s'appellent mqxr_PPPPP.trc, où PPPPP est l'ID de processus.

Référence associée

strmqtrc

Traçage du client Java MQTT v3

Suivez ces instructions pour créer une trace du client Java MQTT Java et contrôler sa sortie.

Avant de commencer

Cette rubrique est uniquement applicable à IBM WebSphere MQ version 7.5.0.0. Pour obtenir des informations sur le traçage du client Java pour les versions ultérieures, voir «Traçage et débogage du client Java MQTT (Paho)», à la page 171.

Le traçage est une fonction de prise en charge. Suivez ces instructions si un ingénieur service IBM vous demande de tracer votre client Java MQTT. La documentation du produit ne fournit pas d'informations sur le format du fichier de trace, ni sur son utilisation pour déboguer un client.

Le traçage fonctionne uniquement pour le client Java WebSphere MQ Telemetry.

Pourquoi et quand exécuter cette tâche

Remarque : Les exemples sont codés pour Windows. Modifiez la syntaxe pour exécuter les exemples sur Linux[2].

Procédure

1. Créez un fichier de propriétés Java contenant la configuration de la trace.

   Dans le fichier de propriétés, spécifiez les propriétés facultatives suivantes. Si la clé de la propriété est spécifiée plusieurs fois, la dernière occurrence définit la propriété.

   a) `com.ibm.micro.client.mqttv3.trace.outputName`

      Répertoire dans lequel le fichier de trace doit être écrit. Par défaut, il s'agit du répertoire de travail du client. Le fichier de trace est appelé `mqtt-`*`n`*`.trc`.

      ```
      java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
      ```

   b) `com.ibm.micro.client.mqttv3.trace.count`

      Nombre de fichiers de trace à écrire. Par défaut, un fichier de taille illimitée.

      ```
      java com.ibm.micro.client.mqttv3.trace.count=5
      ```

   c) `com.ibm.micro.client.mqttv3.trace.limit`

      Taille maximale du fichier à écrire ; la valeur par défaut est 500000. La limite s'applique uniquement si plusieurs fichiers de trace sont demandés.

      ```
      java com.ibm.micro.client.mqttv3.trace.limit=100000
      ```

   d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

      Activez ou désactivez la trace par client. Si *`clientIdentifier`*`=*`, la trace est activée ou désactivée pour tous les clients. Par défaut, la trace est désactivées pour tous les clients.

      ```
      java com.ibm.micro.client.mqttv3.trace.client.*.status=on
      ```

      ```
      java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
      ```

2. Transmettez le fichier de propriétés de trace à la machine virtuelle Java utilisant une propriété système.

   ```
   java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
   ```

3. Exécutez le client.
4. Convertissez le fichier de trace d'un codage binaire à un codage texte ou `.html`. Utilisez la commande suivante :

[2] Java utilise le délimiteur de chemin correct. Vous pouvez coder le délimiteur dans un fichier de propriétés en tant que `'/'` ou `'\\'`; `'\'` est le caractère d'échappement

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

dans laquelle les arguments sont les suivants :

-?
Affiche l'aide

-i traceFile
Obligatoire. Permet d'accéder au fichier d'entrée (par exemple, `mqtt-0.trc`).

-o outputFile
Obligatoire. Définit le fichier de sortie (par exemple, `mqtt-0.trc.html` ou `mqtt-0.trc.txt`).

-h
Sortie HTML. L'extension des fichiers de sortie doit être `.html`. Si elle n'est pas spécifiée, la sortie s'affiche en texte brut.

-d time
Indente une ligne avec * si la différence de temps en millisecondes est supérieure ou égale à (> =) temps. Cet argument ne s'applique pas à la sortie HTML.

L'exemple ci-dessous présente une sortie HTML du fichier de trace.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

Le deuxième exemple présente une sortie en texte brut du fichier de trace, avec un retrait utilisant l'astérisque (*) lorsque des horodatages consécutifs comportent un décalage de 50 millisecondes ou plus.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

Le dernier exemple présente une sortie en texte brut du fichier de trace :

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Traçage du client MQTT pour C

Définissez la variable d'environnement `MQTT_C_CLIENT_TRACE` pour tracer une app C client MQTT.

Avant de commencer

La trace du client MQTT pour C est disponible pour les bibliothèques préconfigurées Windows et Linux du client MQTT pour C, ainsi que pour les bibliothèques iOS que vous avez générées.

Pourquoi et quand exécuter cette tâche

Définissez comme valeur de la variable d'environnement `MQTT_C_CLIENT_TRACE` le chemin du fichier qui doit contenir la sortie de la trace. La sortie de la trace est enregistrée dans ce fichier.

Procédure

Définissez `MQTT_C_CLIENT_TRACE=mqttcclient.log` avant d'exécuter votre app C client MQTT.

a) Par exemple, modifiez le modèle de script dans «Initiation au client MQTT pour C», à la page 26 :

```
@echo off
setlocal
set MQTT_C_CLIENT_TRACE=mqttcclient.log
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
cl  /nologo /D "WIN32" /I "..\include" "MQTTV3Sample.c" /link /
nologo ..\windows_ia32\mqttv3c.lib
set path=%path%;..\windows_ia32;
```

```
start "MQTT Subscriber" MQTTV3Sample -a subscribe -b localhost -p 1883
@rem Sleep for 2 seconds
ping -n 2 127.0.0.1 > NUL 2>&1
MQTTV3Sample -b localhost -p 1883
pause
endlocal
```

b) Exécutez le script du répertoire `%sdkroot%/sdk/client/c/samples`.

Résultats

Les fichiers de sortie de trace commencent par les lignes suivantes :

```
============================================================
                   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
```

Tâches associées

«Initiation au client MQTT pour C», à la page 26
Familiarisez-vous avec le modèle de client MQTT pour C sur toutes les plateformes sur lesquelles vous pouvez compiler le code source C. Vérifiez que vous pouvez exécuter le modèle de client MQTT pour C avec IBM MessageSight ou IBM WebSphere MQ en tant que serveur MQTT.

Traçage et débogage du client Java MQTT (Paho)

Le consignateur par défaut utilise la fonction de journalisation Java standard appelée `java.util.logging` (JSR47). Vous pouvez la configurer à l'aide d'un fichier de configuration ou à l'aide d'un programme.

Pourquoi et quand exécuter cette tâche

Remarque : Le client Java Paho s'applique uniquement aux versions de IBM WebSphere MQ versions 7.5.0.1 et ultérieures. Pour plus d'informations sur le traçage du client Java dans IBM WebSphere MQ version 7.5.0.0, voir «Traçage du client Java MQTT v3», à la page 168.

Remarque : Le traçage est une fonction de prise en charge. Suivez ces instructions si un technicien de maintenance IBM vous demande de tracer votre client Java MQTT . La documentation du produit ne

fournit pas d'informations sur le format du fichier de trace, ni sur son utilisation pour déboguer un client. La fonction de trace fonctionne uniquement pour le client Java IBM WebSphere MQ Telemetry.

La méthode la plus simple pour utiliser un fichier de configuration consiste à spécifier son nom dans la propriété `java.util.logging.config.file`.

Un fichier de propriétés de travail `jsr47min.properties` est fourni dans le package `org.eclipse.paho.client.mqttv3.logging`

La fonction de consignation JSR47 peut être utilisée de plusieurs manières :

- Pour collecter des messages d'un ensemble sélectionné de packages
- Pour collecter des messages à partir et en-dessous d'un niveau de journalisation
- Pour choisir plusieurs destinations pour les messages de journal
- En fournissant un consignateur intégré qui écrit dans un fichier et contrôle la taille et le nombre de fichiers utilisés
- En fournissant un consignateur intégré qui écrit en mémoire et autorise l'écriture des messages en mémoire en fonction d'un déclencheur
- Si l'application qui utilise la bibliothèque client MQTT est également instrumentée par l'utilisation de JSR47, les messages de l'application et de la bibliothèque client se mélangent.

Une classe utilitaire est fournie pour aider à collecter des informations de débogage. Cette classe inclut les messages de journal et de trace décrits précédemment, mais peut collecter des informations telles que les propriétés système Java et la valeur des variables à partir du client Paho.

La fonction de débogage est fournie dans la classe publique Debug, qui fait partie du package `org.eclipse.paho.client.mqttv3.util`. Une instance de Debug peut être obtenue à l'aide de la méthode getDebug() à la fois sur les objets client MQTT asynchrones et synchrones.

Exemple :

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

La méthode dumpClientDebug() vide le volume maximal d'informations de débogage. La fonction de journalisation doit être activée pour capturer l'intégralité des informations de débogage qui sont écrites. Pour capturer toutes les informations de débogage, appelez une méthode de vidage lorsque le problème est connu, par exemple à la suite d'une exception particulière.

Procédure

1. Créez un fichier de configuration ou utilisez le fichier jsr47min.properties fourni.

   Si vous utilisez le fichier de propriétés fourni, vérifiez que le déclencheur de commande push est défini sur le niveau d'erreur approprié. Par défaut, il est défini sur un niveau d'erreur grave mais il peut s'avérer nécessaire d'enregistrer la trace en continu dans le fichier plutôt que de la conserver en mémoire jusqu'à ce qu'une erreur se produise. Pour ce faire, changez :

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   à

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. Transmettez le fichier de configuration de trace à la machine virtuelle Java à l'aide d'une propriété système.

   Si vous utilisez le fichier jsr4min.properties :

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Exécutez le client.

Résultats

Lorsqu'une exception ou un problème se produit, la classe Paho Debug enregistre la trace en mémoire dans le fichier cible configuré.

La trace n'est pas automatiquement écrite dans le fichier lorsqu'elle est générée, cela se produit uniquement lorsque le déclencheur de commande push est actionné ou que la classe de débogage provoque l'enregistrement de la trace. Ce dernier cas peut nécessiter la modification du code d'application.

Chaque ligne est écrite dans le gestionnaire de fichiers lorsqu'il est créé. Vous pouvez contrôler le format dans lequel les messages sont enregistrés en configurant un gestionnaire de fichiers. Un gestionnaire de fichiers personnalisé est fourni avec Paho ; qui enregistre plus que le gestionnaire simple et moins que le gestionnaire XMLHandler fourni avec l'environnement JRE. Les enregistrements de trace qui utilisent le formateur de journal Paho sont au format suivant :

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Exemple

Un fichier de propriétés de travail `jsr47min.properties` est fourni. Ce fichier contient une suggestion de configuration pour la collecte de la trace qui aide à résoudre les problèmes relatifs au client Paho MQTT. Il configure la trace pour qu'elle soit collectée en mémoire en continu avec un minimum d'impact sur la performance. Lorsque le déclencheur de commande push s'exécute ou qu'une demande spécifique de placement est effectuée, la trace en mémoire est placée dans le gestionnaire cible configuré. Le déclencheur de commande push par défaut est un message de niveau Grave qui correspond à une connexion interrompue. Par défaut, la trace qui est collectée en mémoire est alors enregistrée dans le fichier spécifié. Par défaut, ce fichier est le fichier `java.util.logging.FileHandler` standard. Vous pouvez utiliser la classe Paho Debug pour placer la trace de mémoire dans sa cible.

Des détails complets sur JSR47 sont disponibles dans le document Javadoc du package 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
```

Que faire ensuite

Pour collecter une trace à l'aide d'un programme, une classe utilitaire est fournie pour aider à collecter les informations de débogage. Cette classe inclut les messages de journal et de trace décrits précédemment, mais peut collecter des informations telles que les propriétés système Java et la valeur des variables à partir du client Paho.

La fonction de débogage est fournie dans la classe publique `Debug`, qui fait partie du package `org.eclipse.paho.client.mqttv3.util`. Une instance de Debug peut être obtenue à l'aide de la méthode getDebug() à la fois sur les objets client MQTT asynchrones et synchrones.

Exemple :

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

La méthode dumpClientDebug() vide le volume maximal d'informations de débogage. La fonction de journalisation doit être activée pour capturer l'intégralité des informations de débogage qui sont écrites. Pour capturer toutes les informations de débogage, appelez une méthode de vidage lorsque le problème est connu, par exemple à la suite d'une exception particulière.

Traçage du client JavaScript MQTT

Vous pouvez utiliser le client JavaScript pour collecter une trace en modifiant l'application Web client pour appeler des méthodes sur l'objet client connecté.

Pourquoi et quand exécuter cette tâche

Vous pouvez collecter une trace à l'aide des méthodes suivantes :

- `client.startTrace()` démarre le traçage pour le client.
- `client.stopTrace()` arrête la trace pour le client.
- `client.getTraceLog()` renvoie la mémoire tampon de trace en cours.

Vous pouvez sortir la mémoire tampon de trace pour l'envoyer au service de support logiciel IBM. Pour ce faire, vous pouvez suivre plusieurs méthodes. L'exemple illustre la trace en cours de démarrage, puis la sortie envoyée à la console et à une adresse e-mail spécifiée, et enfin la trace en cours d'arrêt.

```
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();
 };
```

Exemple de sortie :

```
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 Configuration système requise pour l'utilisation des suites de chiffrement SHA-2 avec les clients MQTT

Pour Java 6 à partir de IBM, SR13 et versions ultérieures, vous pouvez utiliser des suites de chiffrement SHA-2 pour sécuriser vos canaux MQTT et vos applications client. Toutefois, les suites de chiffrement SHA-2 ne sont pas activées par défaut jusqu'à Java 7 à partir de IBM, SR4 . Par conséquent, dans les versions antérieures, vous devez spécifier la suite requise. Si vous exécutez un client MQTT avec votre propre JRE, vous devez vous assurer qu'il prend en charge les suites de chiffrement SHA-2. Pour que vos applications client utilisent les suites de chiffrement SHA-2, le client doit aussi associer le contexte SSL à une valeur qui prend en charge Transport Layer Security (TLS) version 1.2.

Pour Java 7 à partir de IBM, à partir de SR4 , les suites de chiffrement SHA-2 sont activées par défaut. Pour Java 6 depuis IBM, SR13 et les éditions de service ultérieures, si vous définissez un canal MQTT sans spécifier de suite de chiffrement, le canal n'acceptera pas les connexions d'un client utilisant une suite de chiffrement SHA-2 . Pour utiliser les suites de chiffrement SHA-2, vous devez définir la suite requise dans la définition du canal. Ainsi, le serveur MQTT active la suite avant l'établissement des connexions. Cela signifie également que seules les applications client utilisant la suite définie peuvent se connecter au canal.

Une limitation similaire existe pour le client MQTT pour Java. Si le code client s'exécute sur un environnement d'exécution Java 1.6 depuis IBM, les suites de chiffrement SHA-2 requises doivent être explicitement activées. En outre, le client ne peut utiliser ces suites que si son contexte SSL prend en charge la version 1.2 du protocole Transport Layer Security (TLS). Exemple :

```
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);
```

En juin 2013, Internet Explorer 10 est le seul navigateur qui fonctionne avec client de messagerie MQTT pour JavaScript et prend également en charge le protocole TLS 1.2 . Il s'agit donc du seul navigateur que vous pouvez utiliser si vous souhaitez établir des connexions SHA-2 avec le client JavaScript .

Consultez les liens connexes pour connaître la liste des suites de chiffrement actuellement prises en charge.

Concepts associés

«Configuration du client MQTT pour son authentification par SSL», à la page 106
Pour authentifier le client MQTT à l'aide de SSL, le client se connecte à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port TCP qui correspond au canal de télémétrie qui est configuré pour l'authentification des clients SSL.

«Configuration du client MQTT l'authentification du canal par SSL», à la page 108
Pour authentifier le canal de télémétrie à l'aide de SSL, le client doit se connecter à un canal de télémétrie à l'aide de SSL. Il doit spécifier un port qui correspond au canal de télémétrie qui est configuré pour SSL. La configuration doit comprendre un magasin de clés protégé par phrase passe contenant un certificat numérique signé du serveur de manière privée.

V 7.5.0.1 Restrictions concernant la prise en charge des navigateurs pour les applications Web de messagerie mobile sur SSL

Les différents navigateurs, combinés aux différentes plateformes, présentent des fonctionnalités différentes. La compréhension de ces différences vous aidera à configurer vos applications, autorités de certification et certificats client pour une connexion à l'aide du client de messagerie MQTT pour JavaScript sur SSL et WebSockets.

La messagerie mobile à l'aide de JavaScript sur SSL est relativement nouvelle ; il n'est donc pas surprenant que les diverses combinaisons navigateur/plateforme aient implémenté cette fonction de manières légèrement différentes, et avec une portée différente. Le tableau suivant présente ce qui fonctionne ou non à l'heure actuelle pour chaque combinaison de navigateur (Firefox, Chrome, Internet Explorer et Safari) et plateforme (Windows, Linux, Mac, iOS et Android).

Tableau 6. Prise en charge SSL par plateforme et navigateur. Pour chaque combinaison navigateur/plateforme, le tableau indique si les connexions SSL anonymes et non anonymes sont prises en charge, et dans quelle mesure le navigateur fonctionne avec tous les certificats client et autorités de certification.

Navigateur	**Prise en charge SSL (O/N)**	**Fonctionnement de SSL avec toute autorité de certification (O/N)**	**Informations supplémentaires**
Bureau Firefox.	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Oui	Ajoutez l'autorité de certification et le certificat client au navigateur. Firefox utilise son propre magasin de certificats. Pour importer un certificat de l'autorité de certification, cliquez sur **Outils** > **Options** > **Avancé** > **Chiffrement** > **Afficher les certificats** > **Autorités** > **Importer** Pour importer un certificat client, cliquez sur **Outils** > **Options** > **Avancé** > **Chiffrement** > **Afficher les certificats** > **Vos certificats** > **Importer** Pour activer une connexion sécurisée, spécifiez `https://` dans l'URL. Firefox peut sélectionner un certificat automatiquement ou vous demander l'autorisation à chaque fois. Firefox vous permet également d'utiliser SSL 3.0 ou TLS 1.0 ; vérifiez que les deux sont sélectionnés.

Tableau 6. Prise en charge SSL par plateforme et navigateur. Pour chaque combinaison navigateur/plateforme, le tableau indique si les connexions SSL anonymes et non anonymes sont prises en charge, et dans quelle mesure le navigateur fonctionne avec tous les certificats client et autorités de certification. *(suite)*

Navigateur	**Prise en charge SSL (O/N)**	**Fonctionnement de SSL avec toute autorité de certification (O/N)**	**Informations supplémentaires**
Bureau Chrome.	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Oui	Utilisez le navigateur pour ajouter l'autorité de certification et le certificat client au magasin de certificats du système d'exploitation, qui est partagé avec d'autres logiciels. Pour importer un certificat de l'autorité de certification, cliquez sur **Paramètres** > **Afficher les paramètres avancés** > **Gérer les certificats** > **Autorités de certification racines de confiance** > **Importer** Pour importer un certificat client, cliquez sur **Paramètres** > **Afficher les paramètres avancés** > **Gérer les certificats** > **Personnel** > **Importer** Pour activer une connexion sécurisée, spécifiez `https://` dans l'URL. Chrome vous donne plusieurs possibilités ; sélectionnez celle qui convient selon que vous configurez une connexion anonyme ou non anonyme.

Tableau 6. Prise en charge SSL par plateforme et navigateur. Pour chaque combinaison navigateur/plateforme, le tableau indique si les connexions SSL anonymes et non anonymes sont prises en charge, et dans quelle mesure le navigateur fonctionne avec tous les certificats client et autorités de certification. *(suite)*

Navigateur	**Prise en charge SSL (O/N)**	**Fonctionnement de SSL avec toute autorité de certification (O/N)**	**Informations supplémentaires**
Internet Explorer.	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Oui	Lorsque vous établissez une connexion SSL non anonyme, vous êtes invité à sélectionner le certificat client approprié. Internet Explorer utilise le magasin de certificats Windows, qui est partagé avec d'autres logiciels. Pour importer un certificat de l'autorité de certification, cliquez sur **Outils** > **Options Internet** > **Contenu** > **Certificats** > **Autorités de certification racines de confiance** > **Importer** Pour importer un certificat client, cliquez sur **Outils** > **Options Internet** > **Contenu** > **Certificats** > **Personnel** > **Importer**
Bureau Safari.	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Oui	Utilisez le navigateur pour ajouter l'autorité de certification et le certificat client au magasin de certificats du système d'exploitation, qui est partagé avec d'autres logiciels.

Tableau 6. Prise en charge SSL par plateforme et navigateur. Pour chaque combinaison navigateur/plateforme, le tableau indique si les connexions SSL anonymes et non anonymes sont prises en charge, et dans quelle mesure le navigateur fonctionne avec tous les certificats client et autorités de certification. *(suite)*

Navigateur	**Prise en charge SSL (O/N)**	**Fonctionnement de SSL avec toute autorité de certification (O/N)**	**Informations supplémentaires**
Firefox sous Android	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Non	Non anonyme : Les certificats client ne fonctionnent pas car vous ne pouvez pas ajouter votre autorité de certification à la liste dans Firefox. Pour importer un certificat client, cliquez sur **Paramètres** > **Sécurité** > **Stockage des données d'identification**. Si votre certificat est signé par une autorité de certification sécurisée dans la liste, vous pouvez établir une connexion sécurisée.
Chrome sous Android	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Non	Non anonyme : Les certificats client ne fonctionnent pas car vous ne pouvez pas ajouter votre autorité de certification à la liste dans Chrome. **Remarque :** Google prévoie cette prise en charge dans la version 27 de Chrome. Il s'agit d'un défaut apparent depuis la version 18. Pour importer un certificat client, cliquez sur **Paramètres** > **Sécurité** > **Stockage des données d'identification**. Si votre certificat est signé par une autorité de certification sécurisée dans la liste, vous pouvez établir une connexion sécurisée.

Tableau 6. Prise en charge SSL par plateforme et navigateur. Pour chaque combinaison navigateur/plateforme, le tableau indique si les connexions SSL anonymes et non anonymes sont prises en charge, et dans quelle mesure le navigateur fonctionne avec tous les certificats client et autorités de certification. *(suite)*

Navigateur	**Prise en charge SSL (O/N)**	**Fonctionnement de SSL avec toute autorité de certification (O/N)**	**Informations supplémentaires**
Safari sous iOS	SSL Anonyme - Oui SSL Non anonyme - Oui	SSL Anonyme - Oui SSL Non anonyme - Non	Non anonyme : Le périphérique ne fait pas confiance au certificat client, même lorsque le certificat de l'autorité de certification est installé en même temps. Safari utilise le magasin de certificats du périphérique. Pour effectuer une importation dans ce magasin, cliquez sur **Paramètres** > **Général** > **Profil**et servez le certificat de l'autorité de certification ou du client à partir d'une page Web ou envoyez-le par courrier électronique à vous-même.
Chrome sous iOS	SSL Anonyme - Oui SSL Non anonyme - Non	SSL Anonyme - Non SSL Non anonyme - Non	Anonyme : Seules les applications Apple ont accès au magasin racine du système iOS. Par conséquent, Chrome doit utiliser sa propre liste d'autorités de certification, à laquelle vous ne pouvez rien ajouter. Non anonyme : Les certificats client ne fonctionnent pas car vous ne pouvez pas ajouter votre autorité de certification à la liste.

Tâches associées

«Connexion du client de messagerie MQTT pour JavaScript sur SSL et WebSockets», à la page 78
Connectez votre app Web de manière sécurisée à IBM WebSphere MQ en utilisant les pages HTML du modèle de client de messagerie MQTT pour JavaScript avec SSL et le WebSocket protocol.

Information associée

Mozilla: (SSL) Does Firefox use the Android CA storage or its own?
Chromium: Issue 134418 - Implement client certificate support
Unable to open https site with not trusted certificate on ie10

Résolution des problèmes : Le client MQTT ne se connecte pas

Identifiez et corrigez la raison pour laquelle un programme client MQTT ne se connecte pas au service de télémétrie (MQXR).

Avant de commencer

Le problème est-il lié au serveur, au client ou à la connexion ? Avez-vous écrit votre propre client de gestion de protocole MQTT v3 ou une application client MQTT utilisant les clients MQTT C ou Java WebSphere ?

Exécutez l'application de vérification fournie avec WebSphere MQ Telemetry sur le serveur, et vérifiez que le canal et le service de télémétrie (MQXR) fonctionnent correctement. Transférez ensuite l'application de vérification au client et exécutez-là au niveau du client.

Pourquoi et quand exécuter cette tâche

Il existe plusieurs raisons pour lesquelles un client MQTT ne peut pas se connecter ou pour lesquelles vous pouvez conclure qu'il ne s'est pas connecté au service de télémétrie.

Procédure

1. Pensez aux déductions qui peuvent être tirées du code d'anomalie renvoyé par le service de télémétrie (MQXR) à `MqttClient.Connect`. De quel type d'incident de connexion s'agit-il ?

Option	Description
REASON_CODE_INVALID_PROTOCOL_VERSION	Assurez-vous que l'adresse de connecteur correspond à un canal de télémétrie et que vous n'avez pas utilisé la même adresse de connecteur pour un autre courtier.
REASON_CODE_INVALID_CLIENT_ID	Vérifiez que l'identificateur client ne dépasse pas 23 octets et qu'il ne contient que des caractères de la plage: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	Vérifiez que l'identificateur de client n'est pas identique au nom de gestionnaire de files d'attente.
REASON_CODE_SERVER_CONNECT_ERROR	Vérifiez que le service de télémétrie (MQXR) et le gestionnaire de files d'attente fonctionnent normalement. Utilisez **netstat** pour vérifier que l'adresse de connecteur n'est pas allouée à une autre application.

   Si vous avez écrit une bibliothèque client MQTT au lieu d'utiliser l'une des bibliothèques fournies par IBM WebSphere MQ Telemetry, reportez-vous au code retour `CONNACK`.

   De ces trois erreurs, vous pouvez déduire que le client s'est connecté au service de télémétrie (MQXR), mais que celui-ci a détecté une erreur.
2. Pensez aux déductions qui peuvent être tirées des codes d'anomalie renvoyés par le client lorsque le service de télémétrie (MQXR) ne répond pas :

Option	Description
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	Recherchez un fichier FDC au niveau du serveur ; voir «Journaux côté serveur», à la page 164. Lorsque le service de télémétrie (MQXR) détecte que le client a dépassé son délai d'attente, il enregistre un fichier de diagnostic de premier niveau (FDC). Il écrit un fichier FDC chaque fois que la connexion est interrompue de manière inattendue.

Le service de télémétrie (MQXR) n'a peut-être pas répondu au client, et celui-ci a dépassé son délai d'attente. Le client WebSphere MQ Telemetry Java se bloque uniquement si l'application a défini un délai d'attente indéfini. Le client envoie l'une des ces exceptions après que le délai d'attente défini pour `MqttClient.Connect` expire avec un problème de connexion non diagnostiqué.

A moins que vous n'ayez trouvé un fichier FDC qui soit en corrélation avec la défaillance de connexion, vous ne pouvez pas déduire que le client a tenté de se connecter au serveur :

a) Confirmez que le client a envoyé une demande de connexion.

Vérifiez la demande TCPIP avec un outil tel que **tcpmon**, disponible à l'adresse https://java.net/projects/tcpmon

b) L'adresse de connecteur distant utilisée par le client correspond-elle à l'adresse de connecteur définie pour la canal de télémétrie ?

La classe de persistance fichier par défaut dans le client Java SE MQTT fourni avec IBM WebSphere MQ Telemetry crée un dossier portant le nom: *clientIdentifier*`-tcp`*hostNameport* ou *clientIdentifier*`-ssl`*hostNameport* dans le répertoire de travail du client. Le nom du dossier indique le `nom d'hôte` et le `port` utilisés lors de la tentative de connexion. ; voir «Fichier journaux côté client», à la page 166.

c) Pouvez-vous effectuer un ping vers le serveur distant ?

d) La commande **netstat** sur le serveur montre-t-elle que le canal de télémétrie s'exécute sur le port auquel le client est connecté ?

3. Vérifiez si le service de télémétrie (MQXR) a détecté un problème dans la demande du client.

Le service de télémétrie (MQXR) inscrit les erreurs qu'il détecte dans `mqxr.log`, et le gestionnaire de files d'attente dans `AMQERR01.LOG`.

4. Tentez d'identifier le problème en exécutant un autre client.

- Exécutez le modèle d'application à l'aide du même canal de télémétrie.
- Exécutez le client de l'interface graphique **wmqttSample** pour vérifier la connexion. Obtenez **wmqttSample** en téléchargeant SupportPac IA92.

Remarque : Les anciennes versions de IA92 n'incluent pas la bibliothèque client Java MQTT v3 .

Exécutez les programmes exemples sur la plateforme du serveur afin d'éliminer les incertitudes concernant la connexion au serveur, puis exécutez les exemples sur la plateforme client.

5. Autres points à vérifier :

a) Est-ce que des dizaines et des milliers de clients MQTT tentent de se connecter simultanément ?

Les canaux de télémétrie disposent d'une file d'attente pour mettre en mémoire tampon les files de connexion en attente de connexions entrantes. Les connexions sont traitées en excès de 10 000 par seconde. La taille de la mémoire tampon de commandes en attente peut être configurée à l'aide de l'assistant de canal de télémétrie dans IBM WebSphere MQExplorer. Sa taille par défaut est 4096. Vérifiez que la file de connexion n'a pas été configurée avec une valeur inférieure.

b) Le service de télémétrie (MQXR) et le gestionnaire de files d'attente sont-ils toujours opérationnels ?

c) Le client est-il connecté à un gestionnaire de files d'attente à haute disponibilité qui a commuté son adresse TCPIP ?

d) Le pare-feu filtre-t-il de manière sélective les paquets de données sortants ou renvoyés ?

Résolution des problèmes : La connexion du client MQTT a été supprimée

Identifiez la raison pour laquelle un client envoie des exceptions `ConnectionLost` inattendues après s'être connecté et exécuté pendant une durée courte ou longue.

Avant de commencer

La connexion du client MQTT a abouti. Le client peut être en fonction pendant une longue durée. Si les clients démarrent avec un court intervalle entre eux, le temps s'écoulant entre la connexion et la suppression de la connexion risque d'être court.

Il n'est pas difficile de distinguer une connexion supprimée d'une connexion ayant abouti, puis supprimée ultérieurement. Une connexion supprimée est définie par le client MQTT appelant la méthode `MqttCallback.ConnectionLost` . La méthode est appelée uniquement après que la connexion a abouti. Le symptôme est différent de celui de `MqttClient.Connect` envoyant une exception après avoir reçu un accusé de réception négatif ou un délai d'attente dépassé.

Si l'app client MQTT n'utilise pas les bibliothèques client MQTT fournies par IBM WebSphere MQ, le symptôme dépend du client. Dans le protocole MQTT v3, le symptôme est une absence de réponse ou une réponse tardive à une demande au serveur, ou l'échec d'une connexion TCP/IP.

Pourquoi et quand exécuter cette tâche

Le client MQTT appelle `MqttCallback.ConnectionLost` avec une exception throwable en réponse à tout problème rencontré côté serveur après avoir reçu un accusé de réception positif. Lorsqu'un client MQTT reprend le contrôle après `MqttTopic.publish` et `MqttClient.subscribe`, la demande est transférée à une unité d'exécution de client MQTT responsable de l'envoi et de la réception de messages. Les erreurs se produisant côté serveur sont signalées de manière asynchrone en transmettant une exception throwable à la méthode de rappel `ConnectionLost`.

Le service de télémétrie (MQXR) enregistre toujours un fichier de diagnostic de premier niveau (FDC) lorsqu'il perd la connexion.

Procédure

1. Un autre client utilisant le même `ClientIdentifier` a-t-il démarré ?

   Si un deuxième client ou le même client a démarré ou redémarré avec le même `ClientIdentifier`, la première connexion au premier client est annulée.
2. Le client MQTT a-t-il accédé à une rubrique pour laquelle il n'a pas d'autorisation de publication ou d'abonnement ?

   Toute action pouvant être prise par le service de télémétrie au nom du client qui renvoie `MQCC_FAIL` a comme conséquence la suppression de la connexion client.

   Le code anomalie n'est pas renvoyé au client.

   - Recherchez les messages de journal dans les fichiers `mqxr.log` et `AMQERR01.LOG` du gestionnaire de files d'attente auquel le client est connecté ; voir «Journaux côté serveur», à la page 164.
3. La connexion TCP/IP a-t-elle été supprimée ?

   La valeur du paramètre de délai d'attente d'un pare-feu indiquant qu'une connexion TCPIP doit être inactive est trop faible et a entraîné la suppression de la connexion.

   - Réduisez la durée de la connexion TCPIP inactive à l'aide de `MqttConnectOptions.setKeepAliveInterval`.

Résolution des problèmes : Messages perdus dans une application MQTT

Résolution du problème de la perte de message. Le message est-il non persistant, a-t-il été envoyé au mauvais emplacement, n'a-t-il jamais été envoyé ? Un programme client codé de manière erronée peut perdre des messages.

Avant de commencer

Etes-vous vraiment sûr que le message que vous avez envoyé n'est pas perdu ? Pouvez-vous déduire qu'un message est perdu parce qu'il n'a pas été reçu ? S'il s'agit d'une publication, quel message a été perdu : le message envoyé par le diffuseur de publications ou celui envoyé par l'abonné ? Ou l'abonnement a-t-il été perdu, ou le courtier n'envoie-t-il pas de publications à l'abonné pour cet abonnement ?

Si la solution implique une publication/abonnement distribué à l'aide de clusters ou de structures hiérarchiques de publication/abonnement, il existe de nombreux problèmes de configuration pouvant entraîner la perte d'un message.

Si vous envoyez un message avec une qualité de service "Au moins une fois" ou "Au plus une fois", il est possible que ce message que vous croyiez perdu n'ait pas été distribué comme vous vous y attendiez. Il est peu probable que le message ait été supprimé par erreur du système. Il n'est peut-être pas parvenu à créer la publication ou l'abonnement que vous souhaitiez.

L'étape la plus importante de la détermination du problème de message perdu consiste à confirmer que le message a bien été perdu. Recréez le scénario pour perdre davantage de messages. Utilisez la qualité de service "Au moins une fois" ou "Au plus une fois" pour éliminer tous les cas où le système pourrait supprimer des messages.

Pourquoi et quand exécuter cette tâche

Il existe quatre étapes pour diagnostiquer un message perdu.

1. Les messages "autonomes après diffusion" s'exécutent comme prévu. Les messages "autonomes après distribution" sont parfois supprimés du système.
2. Configuration : la configuration de publication/abonnement avec les droits d'accès appropriés dans un environnement distribué n'est pas directe.
3. Erreurs de programmation client : la responsabilité de la distribution des messages ne relève pas de la seule responsabilité du code écrit par IBM.
4. Si vous avez épuisé toutes ces possibilités, vous pouvez alors décider de faire appel au service IBM.

Procédure

1. Si le message perdu à une qualité de service "Autonome après diffusion", indiquez une qualité de service "Au moins une fois" ou "Au plus une fois". Tentez à nouveau de perdre le message.
   - Les messages envoyés avec la qualité de service "Autonome après diffusion" sont éliminés par IBM WebSphere MQ dans plusieurs circonstances :
     - Communications perdues et canal arrêté.
     - Gestionnaire de files d'attente arrêté.
     - Nombre de messages excessif.
   - La livraison des messages de type "Autonome après diffusion" dépend de la fiabilité de TCP/IP. TCP/IP continue d'envoyer des paquets de données jusqu'à l'obtention de l'accusé de réception de la distribution. Si la session TCP/IP est interrompue, les messages dont la qualité de service est "Autonome après diffusion" sont perdus. La session peut être interrompue par la fermeture de session du client ou du serveur, des problèmes de communication ou la déconnexion de la session par un pare-feu.
2. Vérifiez que le client redémarre la session précédente, afin d'envoyer à nouveau les messages non distribués avec la qualité de service "Au moins une fois" ou "Au plus une fois".

a) Si l'application client utilise le client Java SE MQTT, vérifiez qu'elle définit `MqttClient.CleanSession` sur `false`

b) Si vous utilisez des bibliothèques client différentes, vérifiez qu'une session est redémarrée correctement.

3. Vérifiez que l'application client redémarre la même session, et n'en démarre pas une autre par erreur.

   Pour redémarrer la même session, `cleanSession = false`, et `Mqttclient.clientIdentifier` et aussi `MqttClient.serverURI` doivent avoir les mêmes valeurs que dans la session précédente.

4. Si une session se ferme prématurément, vérifiez que le message est disponible dans le magasin de persistance au niveau du client pour effectuer un nouvel envoi.

   a) Si l'application client utilise le client Java SE MQTT, vérifiez que le message est sauvegardé dans le dossier de persistance ; voir «Fichier journaux côté client», à la page 166

   b) Si vous utilisez des bibliothèques client différentes, ou que vous avez implémenté votre propre mécanisme de persistance, vérifiez qu'il fonctionne correctement.

5. Vérifiez que personne n'a supprimé le message avant sa distribution.

   Les messages non distribués en attente de distribution aux clients MQTT sont stockés dans `SYSTEM.MQTT.TRANSMIT.QUEUE`. Les messages en attente de distribution au serveur de télémétrie sont stockés par le mécanisme de persistance du client. Voir Persistance des messages dans les clients MQTT.

6. Vérifiez que le client dispose d'un abonnement à la publication qu'il doit recevoir.

   Répertorie les abonnements à l'aide de WebSphere MQ Explorer ou à l'aide de commandes **runmqsc** ou PCF. Tous les abonnements client MQTT sont nommés. Ils reçoivent un nom au format suivant: *ClientIdentifier*:*Topic name*

7. Vérifiez que le diffuseur de publications dispose des droits d'accès pour publier et que l'abonné à les droits pour souscrire un abonnement à la rubrique de la publication.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   Dans un système de publication/abonnement en cluster, l'abonné doit disposer des droits d'accès sur la rubrique sur le gestionnaire de files d'attente auquel l'abonné est connecté. Il n'est pas nécessaire pour l'abonné d'être autorisé à s'abonner à la rubrique du gestionnaire de files d'attente où la publication est diffusée. Les canaux entre les gestionnaires de files d'attente doivent disposer des droits appropriés pour la transmission sur l'abonnement du proxy et le réacheminement de la publication.

   Créez le même abonnement et diffusez-y une publication à l'aide d'IBM WebSphere MQ Explorer. Faites une simulation de publication et d'abonnement de votre client d'application en utilisant l'utilitaire client. Démarrez l'utilitaire depuis IBM WebSphere MQ Explorer et modifiez son ID utilisateur pour qu'il corresponde a celui qui a été adopté par votre application client.

8. Vérifiez que l'abonné a les droits nécessaire pour insérer la publication dans la file `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

9. Vérifiez que l'application IBM WebSphere MQ point-à-point est autorisée à insérer son message sur le `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

   Voir "Envoi d'un message à un client directement" dans Configuration des files d'attente réparties en vue de l'envoi de messages aux clients MQTT.

Résolution des problèmes : Le service de télémétrie (MQXR) ne démarre pas

Identifiez et corrigez la raison pour laquelle le service de télémétrie (MQXR) ne démarre pas. Vérifiez l'installation WebSphere MQ Telemetry. Vérifiez notamment qu'aucun fichier ne manque, n'a été déplacé

ou ne dispose de droits non appropriés. Vérifiez les chemins utilisés par le service de télémétrie (MQXR) pour localiser ses programmes.

Avant de commencer

La fonction WebSphere MQ Telemetry est installée. IBM WebSphere MQ Explorer dispose d'un dossier Telemetry dans **IBM WebSphere MQ > Gestionnaires de files d'attente > *qMgrNom* > Telemetry**. Si le dossier n'existe pas, l'installation échoue.

Le service de télémétrie (MQXR) ne peut démarrer que s'il a été créé. Si le service de télémétrie (MQXR) n'a pas été créé, exécutez la commande **Define sample configuration ...** dans le dossier `Telemetry` .

Si le service de télémétrie (MQXR) a déjà été créé, des dossiers **Canaux** et **Statut de canal** supplémentaires sont créés sous le dossier `Telemetry`. Le service de télémétrie `SYSTEM.MQXR.SERVICE` est dans le dossier **Services**. Il est visible si vous avez cliqué sur le bouton d'option pour afficher les objets système.

Cliquez avec le bouton droit de la souris sur `SYSTEM.MQXR.SERVICE` pour démarrer et arrêter le service, afficher son statut et indiquer si votre ID utilisateur est autorisé à démarrer le service.

Pourquoi et quand exécuter cette tâche

Le service de télémétrie (MQXR) `SYSTEM.MQXR.SERVICE` ne démarre pas. L'échec du démarrage se manifeste de deux manières :

1. La commande de démarrage échoue immédiatement.
2. La commande de démarrage s'exécute normalement mais est immédiatement suivie par l'arrêt du service.

Procédure

1. Démarrez le service

   Résultat

   Le service s'arrête immédiatement. Une fenêtre affiche un message d'erreur ; par exemple :

   ```
   WebSphere MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Motif

   Les fichiers sont manquants de l'installation ou les droits sur les fichiers installés n'ont pas été correctement définis.

   La fonction IBM WebSphere MQ Telemetry est installée sur un seul ou deux gestionnaires de files d'attente hautement disponibles. Si l'instance du gestionnaire de files d'attente bascule sur une instance de secours, elle tente de démarrer `SYSTEM.MQXR.SERVICE`. La commande de démarrage du service échoue car le service de télémétrie (MQXR) n'est pas installé sur l'instance de secours.

   Analyse

   Consultez le journal des erreurs ; voir «Journaux côté serveur», à la page 164.

   Actions

   Installez ou désinstallez et réinstallez la fonction WebSphere MQ Telemetry.
2. Démarrez le service, attendez 30 secondes, régénérez l'explorateur et vérifiez le statut du service.

   Résultat

   Le service démarre, puis s'arrête.

   Motif

   `SYSTEM.MQXR.SERVICE` a lancé la commande **`runMQXRService`**, mais la commande a échoué.

Analyse

Consultez le journal des erreurs ; voir «Journaux côté serveur», à la page 164.

Regardez si le problème se produit lorsque seul le modèle de canal est défini. Sauvegardez et effacez le contenu du répertoire *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ . Exécutez l'assistant de modèle de configuration et tentez de démarrer le service.

Actions

Vérifiez les droits d'accès et les problèmes de chemin.

Résolution des problèmes : Le module de connexion JAAS n'est pas appelé par le service de télémétrie

Vérifiez si le module de connexion JAAS n'est pas appelé par le service de télémétrie (MQXR), et configurez JAAS pour corriger le problème.

Avant de commencer

Vous avez modifié *WMQ installation directory*\mqxr\samples\LoginModule.java pour créer votre propre classe d'authentification *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. Vous avez également écrit vos propres classes d'authentification JAAS et les avez placées dans le répertoire de votre choix. Après les tests initiaux avec le service de télémétrie (MQXR), vous pensez que votre classe d'authentification n'est pas appelée par le service de télémétrie (MQXR).

Remarque : Faites en sorte que vos classe d'authentification ne puissent pas être remplacées par des opérations de maintenance sur WebSphere MQ. Utilisez votre propre chemin pour ces classe, plutôt qu'un chemin de l'arborescence de répertoires de WebSphere MQ.

Pourquoi et quand exécuter cette tâche

La tâche utilise un scénario pour illustrer comment résoudre le problème. Dans le scénario, un package appelé `security.jaas` contient une classe d'authentification JAAS appelée `JAASLogin.class`. Elle est stockée dans le chemin `C:\WMQTelemetryApps\security\jaas`. Reportez-vous à la rubrique Configuration JAAS du canal de télémétrie pour en savoir plus sur la configuration de JAAS pour IBM WebSphere MQ Telemetry. L'exemple «Exemple de configuration JAAS», à la page 189 est un modèle de configuration.

Procédure

1. Recherchez dans `mqxr.log` une exception envoyée par `javax.security.auth.login.LoginException`.

   Consultez les «Journaux côté serveur», à la page 164 pour connaître le chemin de `mqxr.log`, et la Figure 54, à la page 190 pour voir un exemple d'exception dans le journal.
2. Corrigez votre configuration JAAS en la comparant avec l'exemple de travail de la section «Exemple de configuration JAAS», à la page 189.
3. Remplacez votre classe de connexion par l'exemple `JAASLoginModule`, après sa restructuration dans le package d'authentification et déployez-le en utilisant le même chemin. Changez la valeur de `loggedIn` en basculant `true` et `false`.

   Si le problème disparaît lorsque `loggedIn` a la valeur `true`, et apparaît lorsque `loggedIn` a la valeur `false`, le problème est donc lié à votre classe de connexion.
4. Vérifiez si le problème se situe au niveau de l'autorisation plutôt que de l'authentification.
   a) Changez la définition de canal de télémétrie pour effectuer un contrôle des autorisations à l'aide d'un ID utilisateur fixé. Sélectionnez un ID qui soit membre d'un groupe `mqm`.
   b) Réexécutez l'application client.

Si le problème disparaît, la solution se trouve au niveau de l'ID transmis pour l'autorisation. Quel nom d'utilisateur a été transmis ? Imprimez-le sur un fichier à partir de votre module de connexion. Vérifiez ses droits d'accès à l'aide d'IBM WebSphere MQ Explorer ou de **dspmqauth**.

Exemple de configuration JAAS

Utilisez l'assistant **Nouveau canal de télémétrie**, dans WebSphere MQ Explorer, pour configurer un canal de télémétrie. Le client se connecte au port 1884 et au canal de télémétrie `JAASMCAUser`. La Figure 48, à la page 189 montre un modèle de fichier de propriétés créé par l'assistant de télémétrie. N'éditez pas ce fichier directement. Le canal est authentifié à l'aide du service JAAS en utilisant une configuration appelée `JAASConfig`. Une fois que le client est authentifié, il utilise l'ID utilisateur `Admin` pour autoriser son accès aux objets 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
```

Figure 48. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties

Le fichier de configuration JAAS comporte une section nommée `JAASConfig` qui nomme la classe Java `security.jaas.JAASLogin`, que JAAS doit utiliser pour authentifier les clients.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Figure 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Lorsque `SYSTEM.MQTT.SERVICE` démarre, il ajoute le chemin de la Figure 50, à la page 189 à son chemin d'accès aux classes.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figure 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

La Figure 51, à la page 189 montre le chemin supplémentaire de la Figure 50, à la page 189 qui a été ajouté au chemin d'accès aux classes du service de télémétrie (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;
```

Figure 51. Sortie Classpath provenant de runMQXRService.bat

La sortie dans la Figure 52, à la page 190 montre que le service de télémétrie (MQXR) a démarré avec la définition de canal de la Figure 48, à la page 189.

```
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
```

Figure 52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log

Lorsque l'application client se connecte au canal JAAS, si `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` ne correspond pas au nom d'une strophe JAAS dans le fichier `jaas.config`, la connexion échoue et le client renvoie une exception avec un code d'anomalie de `0`. Voir Figure 53, à la page 190. La deuxième exception, `Client is not connected (32104)`, a été émise car le client a tenté de se déconnecter alors qu'il n'était pas connecté.

```
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)
```

Figure 53. Exception envoyée lors de la connexion à com.ibm.mq.id.PubAsyncRestartable

`mqxr.log` contient une sortie supplémentaire affichée dans la Figure 53, à la page 190.

L'erreur est détectée par JAAS qui émet l'exception `javax.security.auth.login.LoginException` avec le motif `No LoginModules configured for JAAS`. La raison peut être, comme à la Figure 54, à la page 190, un nom de configuration erroné. Il peut également s'agir d'autres problèmes de service JAAS rencontrés lors du chargement de la configuration JAAS.

Si aucune exception n'est signalée par le service JAAS, cela signifie que JAAS a correctement chargé la classe `security.jaas.JAASLogin` nommée dans la strophe `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
```

Figure 54. mqxr.log - erreur lors du chargement de la configuration JAAS

Résolution des problèmes : Démarrage ou exécution du démon

Consultez le journal de la console du démon pour dispositifsIBM WebSphere MQ Telemetry, activez la trace ou utilisez le tableau des symptômes de la présente rubrique pour traiter les incidents liés au démon.

Procédure

1. Vérifiez le journal de la console.

   Si le démon s'exécute en avant-plan, les messages de le console s'affichent dans la fenêtre du terminal. S'il a été démarré en arrière-plan, la console est l'emplacement cible de `stdout`.
2. Redémarrez le démon.

   Les changements effectués dans le fichier de configuration ne sont pas activés tant que le démon n'est pas redémarré.
3. Consultez le Tableau 7, à la page 191 :

Tableau 7. Tableau des symptômes

Problème	Solution proposée

Le message suivant s'affiche lorsque vous démarrez le démon sous Windows :
`The system cannot execute the specified program`
ou
`The application has failed to start because its side-by-side configuration is incorrect.`
	Installez Microsoft Visual C++ 2008 Redistributable Package.

Plusieurs démons ou serveurs MQTT sont interconnectés par un ou plusieurs ponts, et le processeur est surchargé.	Il existe probablement une boucle de message, avec un ou plusieurs messages transmis de façon répétée d'un serveur à l'autre. Examinez les paramètres de rubrique dans le fichier de configuration. Utilisez des rubriques plus spécifiques quand cela est possible. L'utilisation de caractères génériques étendus est la cause la plus fréquente de ces boucles de connexion.
Le pont ne parvient pas à se connecter à un serveur compatible MQTT auquel d'autres clients MQTT peuvent se connecter.	Le serveur distant n'est peut-être pas compatible avec les tentatives pour déterminer s'il est aussi un démon pour dispositifs WebSphere MQ Telemetry. Paramétrez **`try_private`** sur `off` pour désactiver le traitement spécifique et tenter d'éliminer les boucles de message.
Ce message s'affiche lorsqu'un pont est configuré : `Avertissement : La connexion ne se fait pas sur le premier paquet du connecteur 1888, CONNACK.`	Vous avez probablement configuré un pont pour effectuer une boucle en retour sur le démon local. Le bouclage n'est pas pris en charge.

Résolution des problèmes : Les clients MQTT ne se connectent pas au démon

Les clients ne se connectent pas au démon, ou le démon ne se connecte pas aux autres démons ou à un canal de télémétrie WebSphere MQ.

Pourquoi et quand exécuter cette tâche

Tracez chaque paquet MQTT envoyé et reçu par le démon.

Procédure

Définissez le paramètre **trace_output** sur `protocol` dans le fichier de configuration du démon, ou envoyez à celui-ci une commande à l'aide du fichier `amqtdd.upd`.

Pour obtenir un exemple d'utilisation du fichier `amqtdd.upd` , voir Transférer des messages entre le démon pour dispositifs IBM WebSphere MQ Telemetry et IBM WebSphere MQ .

Avec le paramètre protocol, le démon affiche sur la console un message décrivant chaque paquet envoyé et reçu.

Remarques

:NONE.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service IBM puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM EMEA Director of Licensing
IBM Corporation
Tour Descartes
Armonk, NY 10504-1785
U.S.A.

Pour toute demande d'informations relatives au jeu de caractères codé sur deux octets, contactez le service de propriété intellectuelle IBM ou envoyez vos questions par courrier à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japon

Le paragraphe suivant ne s’applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE "EN L'ETAT" SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Coordinateur d'interopérabilité logicielle, département 49XA
3605 Autoroute 52 N

Rochester, MN 55901
U.S.A.

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans le présent document et tous les éléments sous disponibles s'y rapportant sont fournis par IBM conformément aux dispositions du Contrat sur les produits et services IBM, aux Conditions Internationales d'Utilisation de Logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

Licence sur les droits d'auteur :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Documentation sur l'interface de programmation

Les informations d'interface de programmation, si elles sont fournies, sont destinées à vous aider à créer un logiciel d'application à utiliser avec ce programme.

Ce manuel contient des informations sur les interfaces de programmation prévues qui permettent au client d'écrire des programmes pour obtenir les services de IBM WebSphere MQ.

Toutefois, lesdites informations peuvent également contenir des données de diagnostic, de modification et d'optimisation. Ces données vous permettent de déboguer votre application.

Important : N'utilisez pas ces informations de diagnostic, de modification et d'optimisation en tant qu'interface de programmation car elles sont susceptibles d'être modifiées.

Marques

IBM, le logo IBM , ibm.com, sont des marques d' IBM Corporation dans de nombreux pays. La liste actualisée de toutes les marques d' IBM est disponible sur la page Web "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Les autres noms de produits et de services peuvent être des marques d'IBM ou d’autres sociétés.

Microsoft et Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans d'autres pays.

UNIX est une marque de The Open Group aux Etats-Unis et dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Ce produit inclut des logiciels développés par le projet Eclipse (http://www.eclipse.org/).

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Référence :

(1P) P/N: