7.5

Mobile Messaging i M2M

Uwaga

Przed skorzystaniem z niniejszych informacji oraz produktu, którego one dotyczą, należy zapoznać się z informacjami zamieszczonymi w sekcji “Uwagi” na stronie 199.

Niniejsze wydanie dotyczy wersji 7 wydanie 5 produktu IBM® WebSphere MQ oraz wszystkich kolejnych wydań i modyfikacji, o ile nie zostanie to określone inaczej w nowych wydaniach.

Wysyłając informacje do IBM, użytkownik przyznaje IBM niewyłączne prawo do używania i rozpowszechniania informacji w dowolny sposób, jaki uzna za właściwy, bez żadnych zobowiązań wobec ich autora.

© Copyright International Business Machines Corporation 2007, 2024.

Spis treści

Usługi przesyłania komunikatów dla urządzeń mobilnych i M2M..... 5
Wprowadzenie do produktu MQTT..... 8
Pierwsze kroki z klientami MQTT..... 11
Pierwsze kroki z klientem MQTT dla produktu Java..... 12
Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android..... 19
Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript..... 25
Pierwsze kroki z klientem MQTT dla języka C..... 27
Pierwsze kroki z klientem MQTT dla języka C w systemie iOS..... 49
Programy przykładowe wiersza komend produktu MQTT..... 51
Zabezpieczenia produktu MQTT..... 54
Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java... 58
Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL..... 66
Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS..... 76
Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets..... 81
Budowanie i uruchamianie zabezpieczonego serwera MQTT client sample C app..... 89
Generowanie kluczy i certyfikatów..... 100
Identyfikacja klienta MQTT, autoryzacja i uwierzytelnianie..... 107
Uwierzytelnianie kanału telemetrycznego przy użyciu protokołu SSL..... 112
Prywatność publikacji na kanałach telemetrycznych..... 114
Konfiguracja SSL klientów MQTT i kanałów telemetrycznych..... 115
Konfiguracja kanału telemetrycznego JAAS..... 120
Pojęcia dotyczące programowania..... 122
Klient przesyłania komunikatów MQTT dla produktu JavaScript i aplikacje WWW..... 122
Jak programować aplikacje przesyłania komunikatów w produkcie JavaScript..... 126
Wywołania zwrotne i synchronizacja w aplikacjach klienckich MQTT..... 130
Czyste sesje..... 133
Identyfikator klienta..... 134
Znaczniki dostawy..... 135
Publikacja ostatniego testamentu i testamentu..... 136
Trwałość komunikatu w klientach MQTT..... 136
Publikacje..... 138
Jakość usług świadczonych przez klienta MQTT..... 139
Zachowane publikacje i klienty MQTT..... 140
Subskrypcje..... 141
Łańcuchy tematów i filtry tematów w klientach MQTT..... 142
Skorowidz programistyczny klienta MQTT..... 142
Pierwsze kroki z serwerami MQTT..... 143
IBM WebSphere MQ jako serwer MQTT..... 145
IBM WebSphere MQ Demon telemetryczny dla pojęć związanych z urządzeniami..... 156
Rozwiązywanie problemów z klientami MQTT..... 168
Położenie dzienników telemetrycznych, dzienników błędów i plików konfiguracyjnych..... 169
Kody przyczyn klienta Java MQTT v3..... 172
Śledzenie usługi telemetrycznej (MQXR)..... 173
Śledzenie klienta Java MQTT v3..... 174
Śledzenie klienta MQTT dla języka C..... 176
Śledzenie i debugowanie klienta Java MQTT (Paho)..... 177
Śledzenie klienta MQTT JavaScript..... 180
Wymagania systemowe dotyczące korzystania z zestawów algorytmów szyfrowania SHA-2 z klientami MQTT..... 180

Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL....181
Rozwiązywanie problemu: klient MQTT nie łączy się z serwerem....187
Rozwiązywanie problemu: połączenie klienta MQTT zostało usunięte....189
Rozwiązywanie problemów: utrata komunikatów w aplikacji MQTT....189
Rozwiązywanie problemu: usługa telemetryczna (MQXR) nie jest uruchomiona....191
Rozwiązywanie problemu: moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną....193
Rozwiązywanie problemu: uruchamianie lub uruchamianie demona....196
Rozwiązywanie problemu: klienty MQTT nie nawiązują połączenia z demonem....197

Uwagi....199
Informacje dotyczące interfejsu programistycznego....200
Znaki towarowe....201

Wprowadzenie do produktu MQTT

Więcej informacji na temat wysyłania komunikatów między aplikacjami mobilnymi przy użyciu transportu telemetrycznego produktu MQ (MQTT). Protokół jest przeznaczony do użytku w sieciach bezprzewodowych i sieciach o niskiej przepustowości. Aplikacja dla urządzeń przenośnych, która korzysta z produktu MQTT , wysyła i odbiera komunikaty, wywołując bibliotekę MQTT . Komunikaty są wymieniane za pośrednictwem serwera przesyłania komunikatów MQTT . Klient i serwer MQTT obsługują złożoność dostarczania komunikatów w niezawodny sposób dla aplikacji mobilnej i zapewniają niski koszt zarządzania siecią.

Aplikacje MQTT działają na urządzeniach mobilnych, takich jak smartfony i tablety. Produkt MQTT jest również używany na potrzeby telemetrii do odbierania danych z czujników i do zdalnego sterowania nimi. W przypadku urządzeń mobilnych i czujników MQTT oferuje wysoce skalowalny protokół publikowania/subskrypcji z gwarantowaną dostawą. Aby wysyłać i odbierać komunikaty MQTT , należy dodać do aplikacji bibliotekę klienta MQTT .

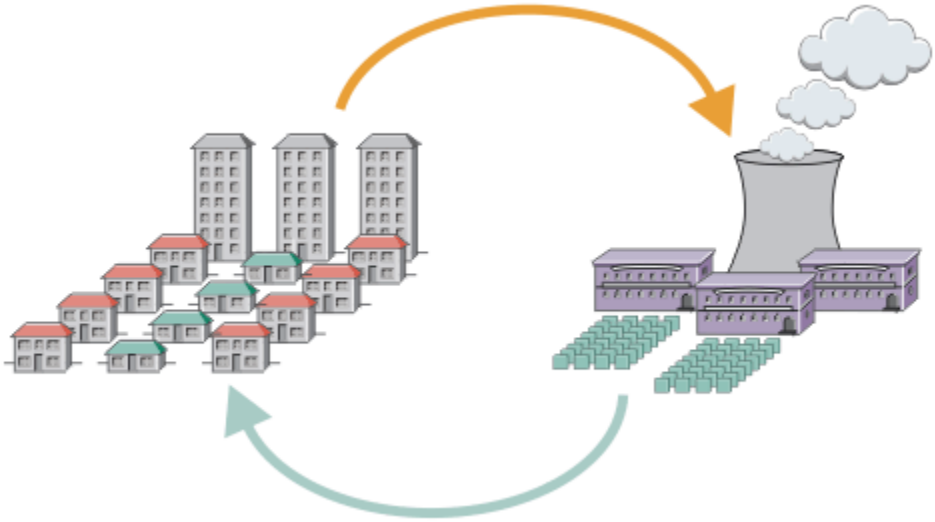
Biblioteka klienta MQTT jest mała. Biblioteka działa jak skrzynka pocztowa, wysyłając i odbierając wiadomości z innymi aplikacjami MQTT połączonymi z serwerem MQTT . Dzięki wysyłaniu komunikatów, a nie pozostawaniu połączonym z serwerem oczekującym na odpowiedź, aplikacje MQTT oszczędzają czas ładowania akumulatora. Biblioteka wysyła komunikaty do innych urządzeń za pośrednictwem serwera MQTT , na którym działa protokół MQTT version 3.1 . Można wysyłać komunikaty do konkretnego klienta lub używać przesyłania komunikatów w trybie publikowania/subskrypcji do łączenia się z wieloma urządzeniami.

Biblioteki klienta MQTT łączą aplikacje dla urządzeń mobilnych i czujników z serwerem MQTT za pomocą protokołu MQTT .

IBM MessageSight i IBM WebSphere MQ to serwery MQTT . Mogą łączyć duże ilości aplikacji klienckich MQTT oraz sieci MQTT i IBM WebSphere MQ . Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143. Zarówno produkt IBM WebSphere MQ , jak i produkt IBM MessageSight mogą tworzyć most między zewnętrznymi aplikacjami WWW działającymi na urządzeniach mobilnych i czujnikach, a także między innymi typami aplikacji publikowania/subskrypcji i przesyłania komunikatów, które działają w przedsiębiorstwie. Most ułatwia tworzenie "inteligentnych rozwiązań" obejmujących urządzenia mobilne i czujniki.

Inteligentne rozwiązania odblokowują bogactwo informacji dostępnych w Internecie dla aplikacji działających na urządzeniach mobilnych i czujnikach. Dwa przykłady inteligentnych aplikacji opartych na telemetrii to inteligentna energia elektryczna i inteligentne usługi zdrowotne.

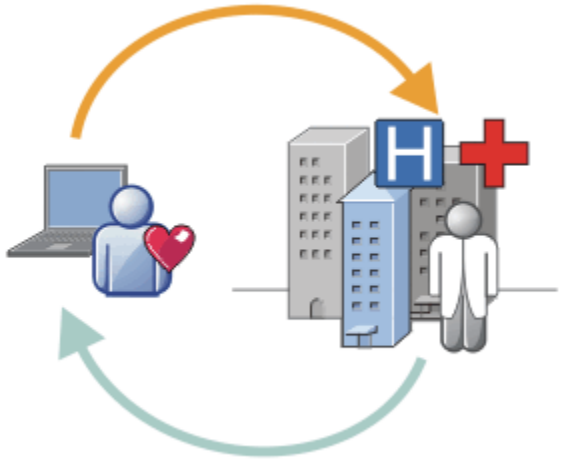
© Copyright IBM Corp. 2007, 2024



- Komunikat MQTT , który zawiera dane dotyczące zużycia energii wysyłane do dostawcy usług.
- Aplikacja telemetryczna wysyła komendy sterujące oparte na analizie danych dotyczących zużycia energii.
- Więcej informacji na ten temat zawiera sekcja Scenariusz telemetryczny: Home energy monitoring and control(Monitorowanie i sterowanie energią w domu).

Rysunek 1. Inteligentne pomiary energii elektrycznej

Rysunek 2. Inteligentne monitorowanie stanu zdrowia



- Aplikacja telemetryczna wysyła dane o zdrowiu do szpitala i lekarza.
- Na podstawie analizy danych dotyczących poprawności można wysyłać alerty lub informacje zwrotne dotyczące komunikatów programu MQTT .
- Więcej informacji na ten temat zawiera sekcja Scenariusz pomiarowy: podstawowe monitorowanie pacjentów.

Program MQTT można wbudować w małe urządzenia, pisząc własną aplikację dla MQTT protocol. W tym celu program IBM udostępnia biblioteki klienckie obsługujące aplikacje, które działają w systemie MQTT. Patrz “Pierwsze kroki z klientami MQTT” na stronie 11. IBM udostępnia biblioteki klienckie dla aplikacji iOS i dla aplikacji Android V 7.5.0.1 oraz JavaScript klienta przeglądarki dla aplikacji WWW bez obsługi platformy. V 7.5.0.1 Strony klienta JavaScript łączą się z systemem IBM MessageSight i IBM WebSphere MQ przy użyciu protokołu MQTT i protokołu WebSockets. IBM udostępnia również przykładowe aplikacje MQTT dla języków C i Java w systemach Linux® i Windows.

Biblioteki C i Java działają na platformach iOS, Android, Windowsi wielu platformach UNIX and Linux . Kod źródłowy w języku C dla biblioteki klienta MQTT można przyportować do innych platform. Biblioteki klienckie MQTT dla języków C i Java są dostępne z licencją Open Source z projektu produktu Eclipse Paho . Patrz Eclipse Paho. Specyfikacja MQTT protocol jest otwarta i dostępna w serwisie MQTT.org.

MQTT protocol

MQTT protocol jest lekki w tym sensie, że klienty są małe i efektywnie wykorzystują przepustowość sieci. Protokół MQTT obsługuje gwarantowane dostarczanie i przesyłanie typu fire-and-forget. W protokole dostarczanie komunikatów jest oddzielone od aplikacji. Zakres oddzielania w aplikacji zależy od sposobu zapisywania klienta MQTT i serwera MQTT . Rozłączone dostarczanie zwalnia aplikację z dowolnego połączenia z serwerem i nie oczekuje na komunikaty. Model interakcji jest podobny do poczty elektronicznej, ale zoptymalizowany pod kątem programowania aplikacji.

Protokół MQTT V3.1 został opublikowany; patrz Specyfikacja protokołu MQTT V3.1. Specyfikacja identyfikuje szereg charakterystycznych cech protokołu:

- Jest to protokół publikowania/subskrypcji.

  Oprócz dystrybucji komunikatów typu jeden do wielu, publikowanie/subskrypcja pozwala na oddzielenie aplikacji. Obie funkcje są przydatne w aplikacjach, które mają wiele klientów.
- Nie jest ona w żaden sposób zależna od treści komunikatu.
- Działa w ramach protokołu TCP/IP, który udostępnia podstawowe połączenia sieciowe.
- Ma on trzy jakości usługi dla dostarczania komunikatów:

  "Co najwyżej jeden raz"
  Komunikaty są dostarczane zgodnie z najlepszymi działaniami bazowej sieci Internet Protocol . Może wystąpić utrata komunikatu.

  Tej jakości usługi należy używać na przykład z danymi z czujników otoczenia. Nie ma znaczenia, czy pojedynczy odczyt zostanie utracony, czy następny zostanie opublikowany wkrótce po tym.

  "Co najmniej raz"
  Komunikaty mogą zostać dostarczone, ale mogą wystąpić duplikaty.

  "Dokładnie jeden raz"
  Komunikaty są dostarczane dokładnie raz.

  Tej jakości usług należy używać na przykład z systemami rozliczeniowymi. Zduplikowane lub utracone komunikaty mogą prowadzić do niedogodności lub nałożenia niepoprawnych opłat.
- Jest to ekonomiczna metoda zarządzania przepływem komunikatów w sieci. Na przykład nagłówek o stałej długości ma tylko 2 bajty, a wymiana protokołów jest minimalizowana w celu zmniejszenia ruchu w sieci.
- Zawiera on funkcję "Last Will and Testament" , która powiadamia subskrybentów o nieprawidłowym odłączeniu klienta od serwera MQTT . Patrz “Publikacja ostatniego testamentu i testamentu” na stronie 136.

Produkt MQTT version 3.1 jest obsługiwany przez produkty IBM IBM WebSphere MQ i IBM MessageSight. Produkt MQTT jest zaimplementowany za pośrednictwem protokołu TCP/IP. Dla sieci innych niż TCP/IP dostępna jest inna wersja protokołu MQTT-S. Patrz Specyfikacja MQTT-S version 1.2.

Społeczności MQTT

Produkt IBM jest uruchamiany dla programistów IBM Developer Przesyłanie komunikatów społeczności for MQTT , którzy tworzą aplikacje dla systemów IBM MessageSight i IBM WebSphere MQ.

MQTT.org to dobre miejsce, aby zapoznać się z implementacjami i rozszerzeniami protokołu MQTT oraz omówić je.

MQTT jest projektem open source Eclipse w katalogu Eclipse Technology Project. Społeczność Paho tworzy klienty i serwery Open Source. Patrz Eclipse Paho.

Wprowadzenie do produktu MQTT

Więcej informacji na temat wysyłania komunikatów między aplikacjami mobilnymi przy użyciu transportu telemetrycznego produktu MQ (MQTT). Protokół jest przeznaczony do użytku w sieciach bezprzewodowych i sieciach o niskiej przepustowości. Aplikacja dla urządzeń przenośnych, która korzysta z produktu MQTT , wysyła i odbiera komunikaty, wywołując bibliotekę MQTT . Komunikaty są wymieniane za pośrednictwem serwera przesyłania komunikatów MQTT . Klient i serwer MQTT obsługują złożoność dostarczania komunikatów w niezawodny sposób dla aplikacji mobilnej i zapewniają niski koszt zarządzania siecią.

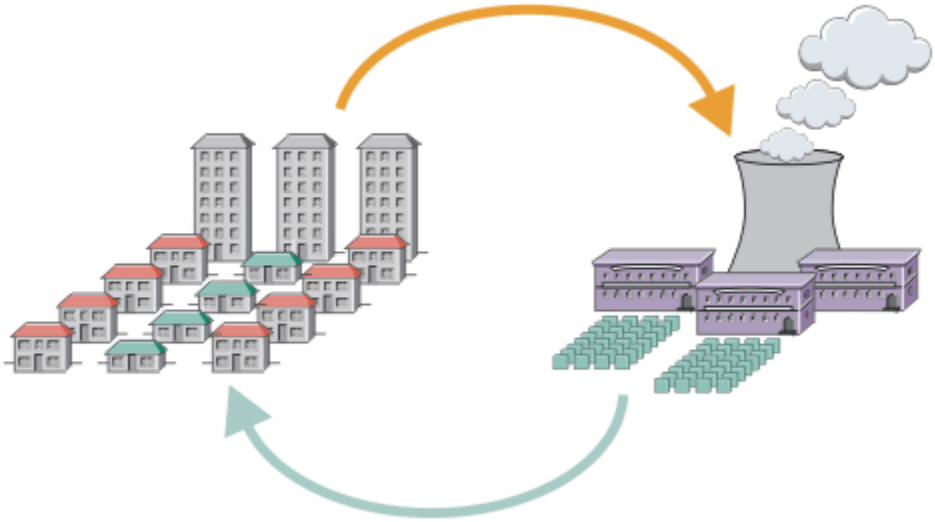
Aplikacje MQTT działają na urządzeniach mobilnych, takich jak smartfony i tablety. Produkt MQTT jest również używany na potrzeby telemetrii do odbierania danych z czujników i do zdalnego sterowania nimi. W przypadku urządzeń mobilnych i czujników MQTT oferuje wysoce skalowalny protokół publikowania/subskrypcji z gwarantowaną dostawą. Aby wysyłać i odbierać komunikaty MQTT , należy dodać do aplikacji bibliotekę klienta MQTT .

Biblioteka klienta MQTT jest mała. Biblioteka działa jak skrzynka pocztowa, wysyłając i odbierając wiadomości z innymi aplikacjami MQTT połączonymi z serwerem MQTT . Dzięki wysyłaniu komunikatów, a nie pozostawaniu połączonym z serwerem oczekującym na odpowiedź, aplikacje MQTT oszczędzają czas ładowania akumulatora. Biblioteka wysyła komunikaty do innych urządzeń za pośrednictwem serwera MQTT , na którym działa protokół MQTT version 3.1 . Można wysyłać komunikaty do konkretnego klienta lub używać przesyłania komunikatów w trybie publikowania/subskrypcji do łączenia się z wieloma urządzeniami.

Biblioteki klienta MQTT łączą aplikacje dla urządzeń mobilnych i czujników z serwerem MQTT za pomocą protokołu MQTT .

IBM MessageSight i IBM WebSphere MQ to serwery MQTT . Mogą łączyć duże ilości aplikacji klienckich MQTT oraz sieci MQTT i IBM WebSphere MQ . Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143. Zarówno produkt IBM WebSphere MQ , jak i produkt IBM MessageSight mogą tworzyć most między zewnętrznymi aplikacjami WWW działającymi na urządzeniach mobilnych i czujnikach, a także między innymi typami aplikacji publikowania/subskrypcji i przesyłania komunikatów, które działają w przedsiębiorstwie. Most ułatwia tworzenie "inteligentnych rozwiązań" obejmujących urządzenia mobilne i czujniki.

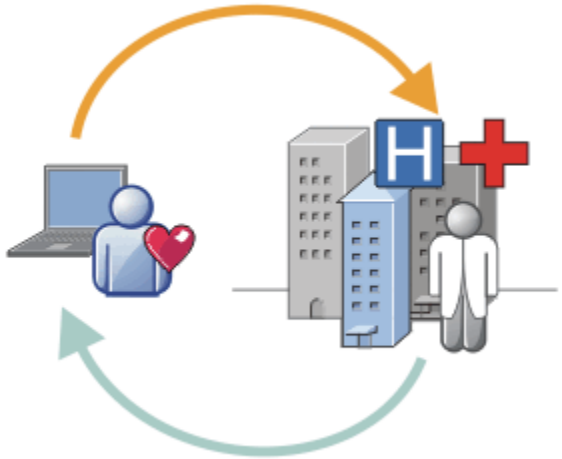
Inteligentne rozwiązania odblokowują bogactwo informacji dostępnych w Internecie dla aplikacji działających na urządzeniach mobilnych i czujnikach. Dwa przykłady inteligentnych aplikacji opartych na telemetrii to inteligentna energia elektryczna i inteligentne usługi zdrowotne.



- Komunikat MQTT , który zawiera dane dotyczące zużycia energii wysyłane do dostawcy usług.
- Aplikacja telemetryczna wysyła komendy sterujące oparte na analizie danych dotyczących zużycia energii.
- Więcej informacji na ten temat zawiera sekcja Scenariusz telemetryczny: Home energy monitoring and control(Monitorowanie i sterowanie energią w domu).

Rysunek 3. Inteligentne pomiary energii elektrycznej

Rysunek 4. Inteligentne monitorowanie stanu zdrowia



- Aplikacja telemetryczna wysyła dane o zdrowiu do szpitala i lekarza.
- Na podstawie analizy danych dotyczących poprawności można wysyłać alerty lub informacje zwrotne dotyczące komunikatów programu MQTT .
- Więcej informacji na ten temat zawiera sekcja Scenariusz pomiarowy: podstawowe monitorowanie pacjentów.

Program MQTT można wbudować w małe urządzenia, pisząc własną aplikację dla MQTT protocol. W tym celu program IBM udostępnia biblioteki klienckie obsługujące aplikacje, które działają w systemie MQTT. Patrz “Pierwsze kroki z klientami MQTT” na stronie 11. IBM udostępnia biblioteki klienckie dla aplikacji iOS i dla aplikacji Android V 7.5.0.1 oraz JavaScript klienta przeglądarki dla aplikacji WWW bez obsługi platformy. V 7.5.0.1 Strony klienta JavaScript łączą się z systemem IBM MessageSight i IBM WebSphere MQ przy użyciu protokołu MQTT i protokołu WebSockets. IBM udostępnia również przykładowe aplikacje MQTT dla języków C i Java w systemach Linux i Windows.

Biblioteki C i Java działają na platformach iOS, Android, Windowsi wielu platformach UNIX and Linux . Kod źródłowy w języku C dla biblioteki klienta MQTT można przyportować do innych platform. Biblioteki klienckie MQTT dla języków C i Java są dostępne z licencją Open Source z projektu produktu Eclipse Paho . Patrz Eclipse Paho. Specyfikacja MQTT protocol jest otwarta i dostępna w serwisie MQTT.org.

MQTT protocol

MQTT protocol jest lekki w tym sensie, że klienty są małe i efektywnie wykorzystują przepustowość sieci. Protokół MQTT obsługuje gwarantowane dostarczanie i przesyłanie typu fire-and-forget. W protokole dostarczanie komunikatów jest oddzielone od aplikacji. Zakres oddzielania w aplikacji zależy od sposobu zapisywania klienta MQTT i serwera MQTT . Rozłączone dostarczanie zwalnia aplikację z dowolnego połączenia z serwerem i nie oczekuje na komunikaty. Model interakcji jest podobny do poczty elektronicznej, ale zoptymalizowany pod kątem programowania aplikacji.

Protokół MQTT V3.1 został opublikowany; patrz Specyfikacja protokołu MQTT V3.1. Specyfikacja identyfikuje szereg charakterystycznych cech protokołu:

- Jest to protokół publikowania/subskrypcji.

  Oprócz dystrybucji komunikatów typu jeden do wielu, publikowanie/subskrypcja pozwala na oddzielenie aplikacji. Obie funkcje są przydatne w aplikacjach, które mają wiele klientów.
- Nie jest ona w żaden sposób zależna od treści komunikatu.
- Działa w ramach protokołu TCP/IP, który udostępnia podstawowe połączenia sieciowe.
- Ma on trzy jakości usługi dla dostarczania komunikatów:

  "Co najwyżej jeden raz"
  Komunikaty są dostarczane zgodnie z najlepszymi działaniami bazowej sieci Internet Protocol . Może wystąpić utrata komunikatu.

  Tej jakości usługi należy używać na przykład z danymi z czujników otoczenia. Nie ma znaczenia, czy pojedynczy odczyt zostanie utracony, czy następny zostanie opublikowany wkrótce po tym.

  "Co najmniej raz"
  Komunikaty mogą zostać dostarczone, ale mogą wystąpić duplikaty.

  "Dokładnie jeden raz"
  Komunikaty są dostarczane dokładnie raz.

  Tej jakości usług należy używać na przykład z systemami rozliczeniowymi. Zduplikowane lub utracone komunikaty mogą prowadzić do niedogodności lub nałożenia niepoprawnych opłat.
- Jest to ekonomiczna metoda zarządzania przepływem komunikatów w sieci. Na przykład nagłówek o stałej długości ma tylko 2 bajty, a wymiana protokołów jest minimalizowana w celu zmniejszenia ruchu w sieci.
- Zawiera on funkcję "Last Will and Testament" , która powiadamia subskrybentów o nieprawidłowym odłączeniu klienta od serwera MQTT . Patrz “Publikacja ostatniego testamentu i testamentu” na stronie 136.

Produkt MQTT version 3.1 jest obsługiwany przez produkty IBM IBM WebSphere MQ i IBM MessageSight. Produkt MQTT jest zaimplementowany za pośrednictwem protokołu TCP/IP. Dla sieci innych niż TCP/IP dostępna jest inna wersja protokołu MQTT-S. Patrz Specyfikacja MQTT-S version 1.2.

Społeczności MQTT

Produkt IBM jest uruchamiany dla programistów IBM Developer Przesyłanie komunikatów społeczności for MQTT , którzy tworzą aplikacje dla systemów IBM MessageSight i IBM WebSphere MQ.

MQTT.org to dobre miejsce, aby zapoznać się z implementacjami i rozszerzeniami protokołu MQTT oraz omówić je.

MQTT jest projektem open source Eclipse w katalogu Eclipse Technology Project. Społeczność Paho tworzy klienty i serwery Open Source. Patrz Eclipse Paho.

Pierwsze kroki z klientami MQTT

Użytkownik może rozpocząć tworzenie aplikacji mobilnej lub aplikacji machine-to-machine (M2M) , budując i uruchamiając przykładową aplikację kliencką MQTT , która korzysta z biblioteki klienta MQTT . Przykładowe aplikacje i powiązane z nimi biblioteki klienckie są dostępne w katalogu Mobile Messaging and M2M Client Pack w serwisie IBM. Istnieją wersje aplikacji i bibliotek klienckich napisane w języku Java, w języku JavaScripti w języku C. Te aplikacje można uruchamiać na większości platform i urządzeń, w tym na urządzeniach z systemem Android i produktach firmy Apple.

Zanim rozpoczniesz

Aby zbudować i uruchomić aplikację, należy mieć pewne doświadczenie w tworzeniu aplikacji dla docelowego urządzenia lub platformy oraz używanego języka programowania. Niewielkie doświadczenie jest zwykle wystarczające, aby przykładowa aplikacja została zainstalowana i uruchomiona na wybranym urządzeniu lub platformie.

Jeśli używany jest korporacyjny serwer MQTT , taki jak IBM WebSphere MQ lub IBM MessageSight, można wymieniać informacje z przykładowej aplikacji z istniejącymi aplikacjami korporacyjnymi.

O tym zadaniu

Cele są następujące:

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.
2. Pobierz Mobile Messaging and M2M Client Pack.
3. Zbuduj, dla docelowego urządzenia lub platformy, przykładowe aplikacje z pakietu klienta.
4. Sprawdź, czy przykłady zachowują się zgodnie z oczekiwaniami, łącząc je z serwerem MQTT .

W wyniku budowania i testowania przykładowych aplikacji dla urządzenia lub platformy tworzone jest działające środowisko programistyczne, którego można następnie użyć do budowania własnych aplikacji klienckich.

Plik Mobile Messaging and M2M Client Pack zawiera pakiet MQTT SDK. Ten pakiet SDK udostępnia następujące zasoby:

- Przykładowe aplikacje klienckie MQTT napisane w języku Java, w języku JavaScripti w języku C.
- Biblioteki klienckie MQTT , które obsługują te aplikacje klienckie, i umożliwiają ich uruchamianie na większości platform i urządzeń.

Pakiet SDK zawiera również kod źródłowy dla MQTT client for C. Ten kod źródłowy można dostosować do budowania bibliotek klienta MQTT dla języka C na innych platformach. Więcej informacji na ten temat zawiera sekcja “Budowanie klienta MQTT dla bibliotek C” na stronie 32. Kod źródłowy MQTT client for C jest również dostępny z licencją Open Source firmy Eclipse Paho.

Procedura

Poniższe artykuły prowadzą użytkownika przez specyficzne dla platformy kroki budowania i uruchamiania przykładowej aplikacji MQTT na komputerze lokalnym lub na urządzeniu mobilnym z systemem Android lub z serwisu Apple:

- “Pierwsze kroki z klientem MQTT dla produktu Java” na stronie 12
- “Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android” na stronie 19
- V 7.5.0.1 “Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript” na stronie 25
- “Pierwsze kroki z klientem MQTT dla języka C” na stronie 27
- “Pierwsze kroki z klientem MQTT dla języka C w systemie iOS” na stronie 49

Co dalej

Aby utworzyć nową aplikację MQTT , należy posiadać lub nabyć następujące umiejętności:

- Programowanie w języku wymaganym dla urządzenia lub platformy.
- Programowanie dla docelowego urządzenia lub platformy.
- Projektowanie aplikacji publikowania/subskrypcji.
- Projektowanie programów dla modelu programistycznego MQTT .
- Projektowanie programów uruchamianych na wybranym urządzeniu mobilnym.
- Używanie protokołu SSL i usługi JAAS do zabezpieczania programów.

Do połączenia klienta MQTT z innym urządzeniem lub aplikacją nie są potrzebne żadne umiejętności programowania sieciowego, ponieważ MQTT jest systemem przesyłania komunikatów i kolejkowania. Biblioteki klienta MQTT zarządzają połączeniami sieciowymi aplikacji.

Aby zintegrować klienta MQTT z istniejącymi aplikacjami korporacyjnymi, można wybrać jedną z dwóch opcji. Tematy związane z publikowaniem i subskrybowaniem produktu MQTT można współużytkować (na przykład) z aplikacją IBM WebSphere MQ lub JMS . Można też napisać własny adapter integracji jako inny klient MQTT .

Źródła informacji, z których należy się dziś konsultować:

- Tworzenie aplikacji dla składnika WebSphere MQ Telemetry
- MQTT.org
- Eclipse Paho

Pojęcia pokrewne

“Pierwsze kroki z serwerami MQTT” na stronie 143

Pierwsze kroki z klientem MQTT dla produktu Java

Pobieranie i uruchamianie z klientem produktu MQTT dla przykładowych aplikacji produktu Java przy użyciu produktu IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Przykładowe aplikacje korzystają z biblioteki klienta z pakietu SDK (Software Development Toolkit) produktu MQTT z serwisu IBM. Przykładowa aplikacja produktu `SampleAsyncCallBack` jest modelem do tworzenia aplikacji MQTT dla systemu Android i innych systemów operacyjnych sterowanych zdarzeniami.

Zanim rozpoczniesz

- Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java". Patrz Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.
- Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .

O tym zadaniu

Celem tego zadania jest sprawdzenie, czy można zbudować i uruchomić przykładową aplikację klienta MQTT dla aplikacji Java , połączyć ją z serwerem IBM WebSphere MQ lub IBM MessageSight jako serwer MQTT version 3 , a następnie wymieniać komunikaty.

Wykonaj to zadanie, aby uruchomić przykładową aplikację z poziomu środowiska roboczego produktu Eclipse lub z wiersza komend. Kroki w tym przykładzie dotyczą produktu Windows. W przypadku niewielkich modyfikacji można uruchomić przykładową aplikację na dowolnej platformie, która obsługuje produkt JSE 1.5 lub wyższy.

Aplikacje można uruchamiać na tym samym serwerze co produkt IBM WebSphere MQ, w którym środowisko do uruchamiania aplikacji, które łączą się z produktem IBM WebSphere MQ , są skonfigurowane dla użytkownika. Wykonaj zadanie “Konfigurowanie usługi MQTT z poziomu wiersza

komend” na stronie 147 , aby zainstalować i skonfigurować produkt IBM WebSphere MQ z opcją IBM WebSphere MQ Telemetry w systemie Windows lub Linux. Po zainstalowaniu i skonfigurowaniu środowiska uruchom przykładową aplikację (MQTTV3Sample), aby zweryfikować instalację.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 . Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Opcjonalne: Skonfiguruj serwer MQTT .
   - W systemie IBM WebSphere MQnależy wykonać jedną lub inne następujące czynności, aby skonfigurować menedżer kolejek i skonfigurować jego usługę telemetryczną (MQXR):
     - “Konfigurowanie usługi MQTT z poziomu wiersza komend” na stronie 147
     - “Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer” na stronie 150
   - Na innych serwerach należy zapoznać się z dokumentacją serwera. Dla partycji Really Small Message Brokernie są wymagane żadne kroki konfiguracyjne. Patrz Really Small Message Broker.
3. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu MQTT. Ścieżka do tego folderu jest określana w tym miejscu jako *sdkroot*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *sdkroot*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *sdkroot*\SDK.
4. Zainstaluj pakiet Java Development Kit (JDK) w wersji 6 lub nowszej.

   Ponieważ tworzy się aplikację Java for Android, produkt JDK musi pochodzić z produktu Oracle. JDK można pobrać z programu Pliki do pobrania Java SE.
5. Skompiluj i uruchom co najmniej jeden klient MQTT dla przykładowych aplikacji produktu Java :
   - “Kompilowanie i uruchamianie przykładowych programów Paho z poziomu wiersza komend” na stronie 14
   - “Compile and run all the MQTT client sample Java apps from Eclipse” na stronie 16
   - “Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android” na stronie 19

   The following MQTT client sample Java apps are included in the SDK:

   MQTTV3Sample
   Przykład ten jest również dołączony do pakietu IBM WebSphere MQ i odsyłaczy do pakietu `com.ibm.micro.client.mqttv3.jar` .

   Sample
   `Sample` znajduje się w pakiecie Paho i odsyłacze do pakietu `org.eclipse.paho.client.mqttv3` . Jest on podobny do `MQTTV3Sample`; oczekuje do czasu zakończenia każdej czynności MQTT .

   SampleAsyncWait
   Klasa `SampleAsyncWait` znajduje się w pakiecie `org.eclipse.paho.client.mqttv3`. Korzysta z asynchronicznego interfejsu API produktu MQTT – czeka w innym wątku na zakończenie działania. Wątek główny może wykonywać inne zadania do czasu zsynchronizowania z wątkiem czekającym na zakończenie działania produktu MQTT.

   SampleAsyncCallBack
   Klasa `SampleAsyncCallBack` znajduje się w pakiecie `org.eclipse.paho.client.mqttv3`. Wywołuje ona asynchroniczny interfejs API produktu MQTT. Asynchroniczny interfejs API nie

czeka na zakończenie przetwarzania wywołania przez produkt MQTT. Powoduje on powrót do aplikacji. Aplikacja wykonuje inne zadania, a następnie czeka na nadejście następnego zdarzenia do przetworzenia. Produkt MQTT wysyła powiadomienie o zdarzeniu z powrotem do aplikacji, gdy zakończy przetwarzanie. Interfejs produktu MQTT sterowany zdarzeniami jest odpowiedni dla modelu programistycznego usług i działań systemu operacyjnego Android oraz innych systemów operacyjnych sterowanych zdarzeniami.

Aplikacja `mqttExerciser` stanowi przykład integracji produktu MQTT z systemem Android przy użyciu modelu programistycznego usług i działań.

`mqttExerciser`
Program `mqttExerciser` jest przykładowym programem dla produktu Android. Ponieważ jest ona zbudowana i uruchamiana inaczej, jest opisywany osobno. Patrz sekcja "Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android" na stronie 19.

Wyniki

Skompilowano i uruchomiono przykładowe aplikacje programu MQTT Java połączone z produktem IBM WebSphere MQ lub IBM MessageSight jako serwer MQTT .

Co dalej

Zapoznaj się z informacjami referencyjnymi Javadoc , patrz krok "3" na stronie 17 produktu "Compile and run all the MQTT client sample Java apps from Eclipse" na stronie 16. Alternatywnie można otworzyć pliki HTML Javadoc znajdujące się w katalogu SDK`\clients\java\doc\javadoc` w Mobile Messaging and M2M Client Pack.

Kompilowanie i uruchamianie przykładowych programów Paho z poziomu wiersza komend

Skompiluj i uruchom przykładową aplikację Paho `Sample.java` z poziomu wiersza komend. Przykład znajduje się w pakiecie SDK produktu MQTT . Przykład został zbudowany przy użyciu bibliotek klienta MQTT Paho w pakiecie `org.eclipse.paho.client.mqttv3` . Demonstruje on publikator i subskrybent produktu MQTT . Dwie inne przykłady Paho w tym samym katalogu mogą być budowane i uruchamiane w ten sam sposób. Różnią się one przez wywołanie asynchronicznie biblioteki produktu MQTT .

Zanim rozpoczniesz

Wykonaj kroki od "1" na stronie 13 do "4" na stronie 13 w głównym zadaniu, aby skonfigurować serwer MQTT i pobrać Mobile Messaging and M2M Client Pack.

O tym zadaniu

Compile and run `Sample.java` from the SDK client samples subdirectory SDK`\clients\java\samples`. Kod Java znajduje się w katalogu SDK`\clients\java\samples\org\eclipse\paho\sample\mqttv3app`.

Procedura

1. Utwórz skrypt w katalogu przykładów klienta, aby skompilować i uruchomić produkt `Sample` na wybranej platformie.

   Poniższy skrypt kompiluje i uruchamia przykład w produkcie 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
```

Rysunek 5. Skompiluj i uruchom `Sample.java`

2. Uruchom skrypt.

   Wyniki:

```
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
```

Rysunek 6. `MQTTV3Sample` Subskrybent

```
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
```

Rysunek 7. `MQTTV3Sample` Publikator

   Jeśli aplikacja subskrybenta nie zostanie uruchomiona, nie będzie można uruchomić tego samego subskrybenta ponownie. Identyfikator klienta nowego subskrybenta jest taki sam, jak stary subskrybent. W tym samym czasie nie można uruchamiać dwóch klientów MQTT z tym samym identyfikatorem klienta. Można ustawić opcję `-i` , aby ustawić identyfikator klienta, tak aby możliwe było uruchamianie różnych subskrybentów w tym samym czasie.

   Jeśli ten sam klient zostanie uruchomiony ponownie, można skorzystać z opcji `-c` , aby uruchomić i uruchomić klienta za pomocą opcji `oczyszczanie` ustawione na wartość false. Za pomocą tej opcji można eksplorować zachowanie przerwanych sesji klientów.

3. Zakończ subskrybent, naciskając klawisz Enter lub zamykając okno.

Co dalej

Utwórz skrypty, aby skompilować i uruchomić inne przykłady w podkatalogu `SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` . Copy the script in Rysunek 5 na stronie 15 and replace `Sample` with `SampleAsyncWait` or `SampleAsyncCallBack`. Pozostałe przykłady to asynchroniczne wersje programu `Sample` synchronicznego.

`SampleAsyncWait`
Klasa `SampleAsyncWait` znajduje się w pakiecie `org.eclipse.paho.client.mqttv3`. Korzysta z asynchronicznego interfejsu API produktu MQTT – czeka w innym wątku na zakończenie działania. Wątek główny może wykonywać inne zadania do czasu zsynchronizowania z wątkiem czekającym na zakończenie działania produktu MQTT.

`SampleAsyncCallBack`
Klasa `SampleAsyncCallBack` znajduje się w pakiecie `org.eclipse.paho.client.mqttv3`. Wywołuje ona asynchroniczny interfejs API produktu MQTT. Asynchroniczny interfejs API nie czeka na zakończenie przetwarzania wywołania przez produkt MQTT. Powoduje on powrót do aplikacji. Aplikacja wykonuje inne zadania, a następnie czeka na nadejście następnego zdarzenia do przetworzenia. Produkt MQTT wysyła powiadomienie o zdarzeniu z powrotem do aplikacji, gdy

zakończy przetwarzanie. Interfejs produktu MQTT sterowany zdarzeniami jest odpowiedni dla modelu programistycznego usług i działań systemu operacyjnego Android oraz innych systemów operacyjnych sterowanych zdarzeniami.

Aplikacja `mqttExerciser` stanowi przykład integracji produktu MQTT z systemem Android przy użyciu modelu programistycznego usług i działań.

Przykłady asynchroniczne demonstrują sposób skrócenie czasu oczekiwania przez aplikację MQTT podczas oczekiwania na klienta MQTT . Ważne jest, aby wyeliminować połączenia blokujące w głównym wątku, aby zwiększyć odpowiedzialność i żywotność baterii w środowisku mobilnym.

W przykładach przedstawiono dwa wzorce wywoływania interfejsów asynchronicznych w kliencie MQTT .

1. Program `SampleAsyncWait` nie blokuje, gdy produkt MQTT oczekuje na interakcje sieciowe.
2. Program `SampleAsyncCallBack` nie blokuje czekania na zakończenie działań przez klienta MQTT . Ta ostatnia jest niezbędna, gdy strona JavaScript wywoła klienta MQTT z przeglądarki. Strony JavaScript nie mogą blokować stron. Odpowiedzi na działania muszą zostać wysłane z powrotem do głównego wątku przeglądarki, który wywołuje procedurę obsługi zdarzeń produktu MQTT zapisanej w celu przetworzenia powiadomienia.

Compile and run all the MQTT client sample Java apps from Eclipse

Compile and run the MQTT client sample Java apps that are in the Mobile Messaging and M2M Client Pack. Demonstrują one publikator i subskrybent produktu MQTT .

O tym zadaniu

Skompiluj i uruchom przykłady MQTT Java , `Sample`i `MQTTV3Sample` w Eclipse. `Sample` znajduje się w podkatalogu klientów pakietu SDK produktu *sdkroot*`\SDK\clients\java\samples\org\eclipse\paho\sample\mqttv3app` , a produkt `MQTTV3Sample.java` znajduje się w katalogu *sdkroot*`\SDK\clients\java\samples`.

Procedura

1. Pobierz produkt Eclipse IDE for Java Developers.
2. Utwórz projekt produktu Java o nazwie `MQTT Samples` w produkcie Eclipse.
   a) **Plik** > **Nowy** > **Projekt Java** i wpisz `MQTT Samples`. Kliknij przycisk **Dalej**.

      Sprawdź, czy środowisko JRE jest w poprawnej lub nowszej wersji.JSE musi być w wersji 1.5 lub nowszej.

   b) W oknie **Ustawienia Java** kliknij opcję **Dowiąż dodatkowe foldery źródłowe**.

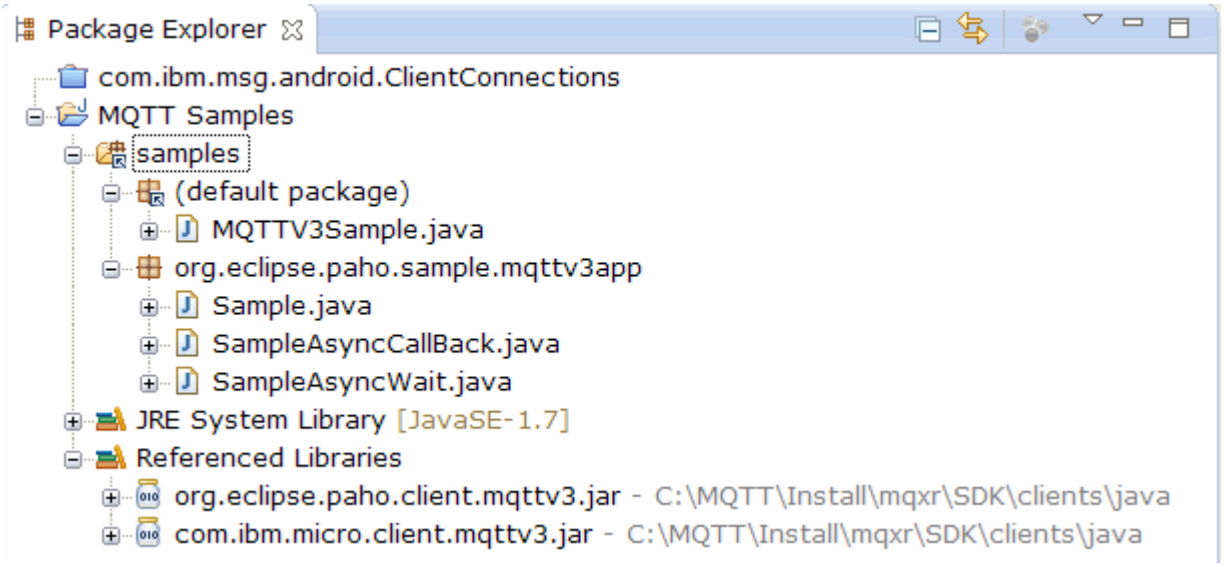
   c) Przejdź do katalogu, w którym został zainstalowany folder pakietu SDK produktu MQTT Java . Wybierz folder *sdkroot*`\SDK\clients\java\samples` i kliknij przycisk **OK** > **Dalej** > **Zakończ**.

   d) W oknie **Java Settings** (Ustawienia Java) kliknij opcję **Libraries** > **Add External Jars** (Biblioteki > Dodaj pliki zewnętrzne)

   e) Przejdź do katalogu, w którym został zainstalowany folder pakietu SDK produktu MQTT Java . Odszukaj folder *sdkroot*`\SDK\clients\java` i wybierz pliki `org.eclipse.paho.client.mqttv3.jar` i `com.ibm.micro.client.mqttv3.jar` , a następnie kliknij opcję **Otwórz** > **Zakończ**.

      Przykładowe odsyłacze produktu `MQTTV3Sample.java` do produktu `com.ibm.micro.client.mqttv3.jar`oraz przykłady w drzewie katalogów produktu paho są odsyłaczami do produktu `org.eclipse.paho.client.mqttv3.jar`. `com.ibm.micro.client.mqttv3.jar` jest zachowywany w taki sposób, że istniejące aplikacje produktu MQTT będą nadal budować i uruchamiać bez zmian.

   Projekt `MQTT Samples` jest kompilowany z ostrzeżeniami, ale nie ma błędów.



Rysunek 8. Klient MQTT dla projektu Java

3. Opcjonalne: Zainstaluj klienta MQTT Javadoc.

   Po zainstalowaniu klienta MQTT w wersji Dokumentacja Javadoc edytor Java opisuje klasy produktu MQTT w pomocy w dymku.

   a) Otwórz w projekcie Java opcję **Eksplorator pakietów** > **Przywoływane biblioteki** . Kliknij prawym przyciskiem myszy opcję `org.eclipse.paho.client.mqttv3.jar` **> Właściwości**.

   b) W nawigatorze właściwości kliknij opcję **Położenie dokumentacji Javadoc**.

   c) Kliknij opcję **Adres URL dokumentacji Javadoc** > **Przeglądaj** na stronie **Położenie dokumentacji Javadoc** , a następnie znajdź folder SDK`\clients\java\doc\javadoc` **> OK**.

   d) Kliknij opcję **Sprawdź poprawność** > **OK** .

   Zostanie wyświetlona prośba o otwarcie przeglądarki w celu wyświetlenia dokumentacji.

   e) Powtórz tę procedurę dla pliku `com.ibm.micro.client.mqttv3.jar` .

4. Utwórz publikator i konfigurację środowiska wykonawczego subskrybenta, aby uruchomić aplikację `mqttv3app.Sample` .

   a) Kliknij prawym przyciskiem myszy klasę **Przykład** , a następnie kliknij opcję **Uruchom jako** > **Uruchom konfiguracje**.

   b) Kliknij prawym przyciskiem myszy opcję **Aplikacja Java** > **Nowy** i wpisz nazwę `SampleSubscriber`.

   c) Kliknij kartę argumentów, a następnie wpisz argumenty programu, a następnie **Zastosuj**.

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

   d) Powtórz ostatni krok, aby utworzyć konfigurację `SamplePublisher` , pomijając parametr `-a subscribe` .

5. Uruchom subskrybent `mqttv3app.Sample` , a następnie publikator.

   a) Kliknij opcję **Uruchom** > **Uruchom konfiguracje** .

   b) Kliknij przycisk **SampleSubscriber** > **Uruchom**.

   Otwórz widok **Konsola** . Subskrybent oczekuje na publikację.

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

   c) Kliknij przycisk **SamplePublisher** > **Uruchom**.

Otwórz widok **Konsola** . Zostanie wyświetlona publikacja, która została utworzona przez publikatora:

```
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) Przełącz widoki konsoli na konsolę subskrybenta.

Ikona konsoli przełącznika to .

Subskrybent otrzymał publikację:

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

6. Opcjonalne: Utwórz publikator i konfigurację środowiska wykonawczego subskrybenta dla produktu `MQTTV3Sample`.
   a) Kliknij prawym przyciskiem myszy klasę **MQTTV3Sample** , a następnie kliknij opcję **Uruchom jako** > **Uruchom konfiguracje**.
   b) Kliknij prawym przyciskiem myszy opcję **Aplikacja Java** > **Nowy** i wpisz nazwę `MQTTV3SampleSubscriber`.
   c) Kliknij kartę argumentów, a następnie wpisz argumenty programu, a następnie **Zastosuj**.

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

   d) Powtórz ostatni krok, aby utworzyć konfigurację serwera `MQTTV3SamplePublisher` , pomijając parametr `-a subscribe` .
7. Opcjonalne: Uruchom subskrybent `MQTTV3Sample` , a następnie publikator.
   a) Kliknij opcję **Uruchom** > **Uruchom konfiguracje** .
   b) Kliknij przycisk **MQTTV3SampleSubscriber** > **Uruchom**.

Otwórz widok **Konsola** . Subskrybent oczekuje na publikację.

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

   c) Kliknij przycisk **MQTTV3SamplePublisher** > **Uruchom**.

Otwórz widok **Konsola** . Istnieje możliwość wyświetlenia publikacji, która jest tworzona przez publikator.

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

   d) Przełącz widoki konsoli na konsolę subskrybenta.

Ikona konsoli przełącznika to .

Subskrybent otrzymał publikację:

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

Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android

Istnieje możliwość zainstalowania serwera Przykładowa aplikacja klienta MQTT Java dla produktu Android , który wymienia komunikaty z serwerem MQTT . Aplikacja używa biblioteki klienta z pakietu MQTT SDK firmy IBM. Można samodzielnie zbudować aplikację lub pobrać wstępnie zbudowaną aplikację przykładową.

Zanim rozpoczniesz

- Informacje na temat obsługiwanych platform klienckich MQTT zawiera sekcja Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.
- Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
- Przykładowa aplikacja kliencka MQTT działa w systemie Ice Cream Sandwich (Android 4.0) i w wersji nowszej. Ta wersja produktu Android zapewnia również bardziej kryjącą rozdzielczość na tabletach.

O tym zadaniu

Plik Przykładowa aplikacja klienta MQTT Java dla produktu Android nosi nazwę "`mqttExerciser`". Ta aplikacja używa biblioteki klienta z pakietu MQTT SDK i wymienia komunikaty z serwerem MQTT .

Można samodzielnie zbudować przykładową aplikację, a następnie wyeksportować ją z katalogu Eclipse jako plik `mqttExerciser.apk`lub użyć wstępnie zbudowanej przykładowej aplikacji, która jest dostępna jako plik `mqttExerciser.apk` w folderze *`sdkroot`*`\SDK\clients\android\samples\apks` w katalogu Mobile Messaging and M2M Client Pack. Jeśli zdecydujesz się na samodzielne budowanie aplikacji, środowisko programistyczne, które budujesz, zostanie dostosowane tak, aby uwzględnić przesyłanie wiadomości w aplikacjach mobilnych for Android. Powinno to być pomocne, gdy użytkownik zaczyna dołączać wiadomości mobilne do własnych aplikacji.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 . Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Zdobądź odpowiednie narzędzia.

   Zainstaluj pakiet Java Development Kit (JDK) w wersji 6 lub nowszej. Ponieważ tworzy się aplikację Java for Android, produkt JDK musi pochodzić z produktu Oracle. JDK można pobrać z programu Pliki do pobrania Java SE.

   Wymagane jest także środowisko programistyczne Eclipse . Musi to być wersja Eclipse 3.6.2 (Helios) lub nowsza. Eclipse musi mieć kompilator Java w wersji co najmniej 6, aby był zgodny z pakietem JDK. Wszystko to można uzyskać z serwisu Eclipse Foundation.

   Na koniec potrzebny jest pakiet Android SDK. Można to uzyskać z serwisu Get the Android SDK.
3. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu `MQTT`. Ścieżka do tego folderu jest określana w tym miejscu jako *`sdkroot`*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *`sdkroot`*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *`sdkroot`*`\SDK`.
4. Opcjonalne: Zbuduj przykładową aplikację mqttExerciser for Android.

Skonfiguruj narzędzia Eclipse i Android oraz zaimportuj i zbuduj projekt `mqttExerciser` z pakietu SDK MQTT .

Uwaga: Jeśli nie chcesz tego robić teraz, możesz użyć wstępnie zbudowanej aplikacji przykładowej, która jest dostępna jako plik `mqttExerciser.apk` w folderze *sdkroot*`\SDK\clients\android\samples\apks` pakietu SDK MQTT .

a) Uruchom środowisko programistyczne Eclipse ze środowiskiem JRE z pakietu JDK.

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

b) Wybierz i zainstaluj zestaw pakietów i platform z pakietu Android SDK.

Listę platform i pakietów rekomendowanych przez program Google zawiera sekcja Dodawanie platform i pakietów .

Uwaga: Platforma SDK musi być na poziomie interfejsu API Android 16 lub nowszym. W przypadku wcześniejszych poziomów interfejsu API nie można pomyślnie skompilować projektu.

c) Dodaj wtyczkę Android Development Tools (ADT) do pliku Eclipse.

d) Zaimportuj przykładowy projekt aplikacji `mqttExerciser` do produktu Eclipsei napraw błędy.

i) Zaimportuj przykładowy projekt aplikacji z pakietu MQTT SDK w ścieżce *sdkroot*`\SDK\clients\android\samples\mqttExerciser`.

Widok **Problemy** zawiera wiele błędów budowania. Błędy budowania zostały rozwiązane w kilku następnych krokach.

ii) Skopiuj bibliotekę `org.eclipse.paho.client.mqttv3.jar` do folderu **libs** w projekcie Android . Windows Na przykład w systemie Windowsznajduje się on w folderze *sdkroot*`\SDK\clients\java` . Zostanie wyświetlone okno **Operacja pliku** . Zaakceptuj wybór opcji **Kopiuj pliki** i kliknij przycisk **OK**.

iii) Kliknij prawym przyciskiem myszy folder projektu, `com.ibm.msg.android`, a następnie kliknij **Narzędzia systemu Android ...** > **Dodaj bibliotekę obsługi ...**. Przeczytaj i zaakceptuj warunki licencji, a następnie kliknij przycisk **Instaluj**.

iv) Kliknij prawym przyciskiem myszy folder projektu, `com.ibm.msg.android`, a następnie kliknij **Narzędzia systemu Android ...** > **Właściwości projektu poprawki**.

v) Jeśli w obszarze roboczym nadal występuje około 84 błędów, odwołując się do przesłaniania metody nadklasy, poziom zgodności kompilatora jest prawdopodobnie ustawiony na wartość 1.5 lub niższą. Pakiet Android SDK w wersji 16 oczekuje, że poziom zgodności kompilatora nie będzie wyższy niż 1.5. Aby naprawić pozostałe błędy, wykonaj następujące kroki:

a) Sprawdź i (jeśli to konieczne) zaktualizuj pakiet Android SDK i odpowiednie wtyczki Eclipse do pakietu Android SDK w wersji 17.

b) Kliknij prawym przyciskiem myszy folder projektu **com.ibm.msg.android** , a następnie wybierz opcję **Właściwości** > **Kompilator Java**. Sprawdź poziom zgodności kompilatora, ustaw go na co najmniej 1.6, a następnie odbuduj obszar roboczy.

Projekt jest kompilowany z ostrzeżeniami i bez błędów.

5. Zainstaluj i uruchom Przykładowa aplikacja klienta MQTT Java na urządzeniu z systemem Android .

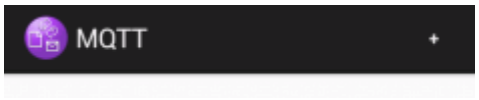
Patrz `developer.android.com` strona Uruchamianie aplikacji.

Jeśli aplikacja została zbudowana samodzielnie jako projekt Eclipse , można ją uruchomić z katalogu Eclipse.

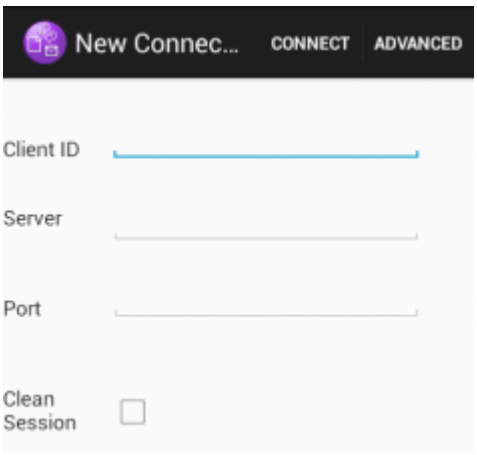
Jeśli dostępny jest plik pakietu aplikacji (APK) `mqttExerciser.apk`, można go zainstalować poza środowiskiem Eclipse za pomocą komendy instalacji Android Debug Bridge (ADB). Ta komenda przyjmuje położenie pliku APK jako argument. Jeśli używana jest wstępnie zbudowana aplikacja przykładowa, położenie jest następujące: *sdkroot*`\SDK\clients\android\samples\apks\mqttExerciser.apk`.

6. Przykładowa aplikacja mqttExerciser for Android służy do nawiązywania połączenia, subskrybowania i publikowania w temacie.
   a) Otwórz Przykładowa aplikacja klienta MQTT Java dla produktu Android.

      To okno jest otwarte w urządzeniu Android :

   b) Nawiąj połączenie z serwerem MQTT .
      i) Kliknij znak **+** , aby otworzyć nowe połączenie MQTT .

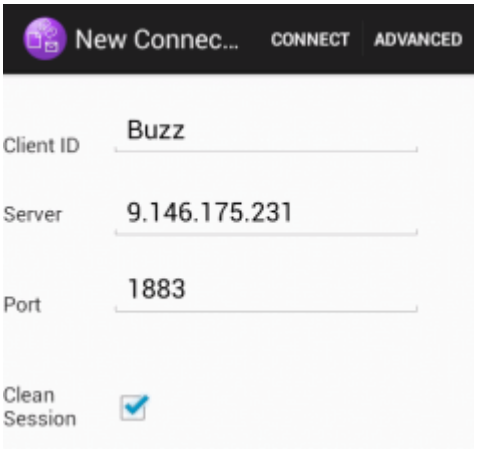
      ii) W polu **ID klienta** wprowadź dowolny unikalny identyfikator. Bądź cierpliwy, naciśnięcia klawiszy mogą być powolne.

      iii) Wpisz w polu **Serwer** adres IP serwera MQTT .

      Jest to serwer, który został wybrany w pierwszym kroku głównym. Adres IP nie może być adresem `127.0.0.1`

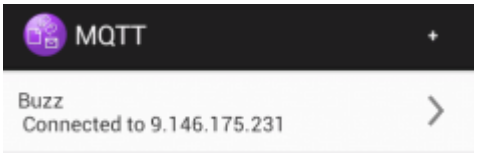
      iv) Wprowadź numer portu połączenia MQTT .

      Domyślny numer portu dla normalnego połączenia MQTT to 1883.

      v) Kliknij przycisk **Połącz**.

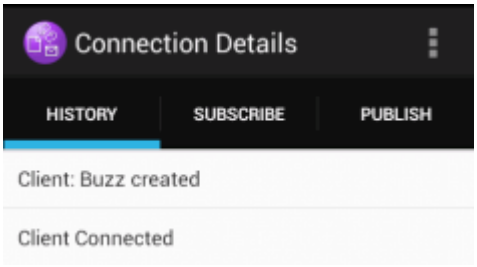
      Jeśli nawiązanie połączenia powiedzie się, zostanie wyświetlony komunikat "Łączenie" , po którym następuje okno:



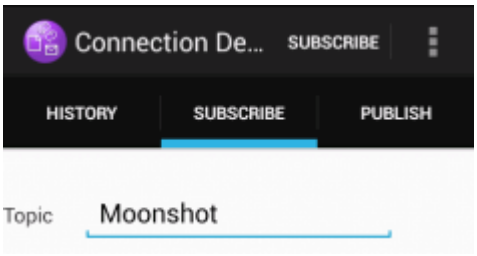
c) Subskrybuj temat.

i) Kliknij komunikat **Połączono** .

Zostanie otwarte okno **Szczegóły połączenia** z wyświetloną historią:



ii) Kliknij kartę **Subskrybuj** i wprowadź łańcuch tematu.

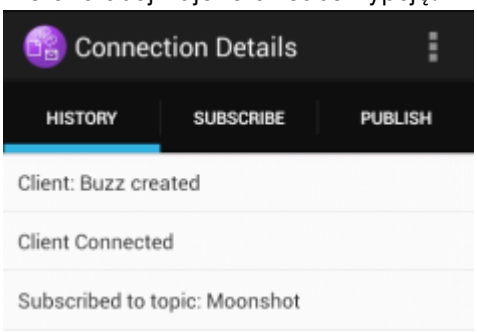


iii) Kliknij działanie **Subskrybuj** .

Komunikat "Zasubskrybowany" jest wyświetlany przez krótki czas.

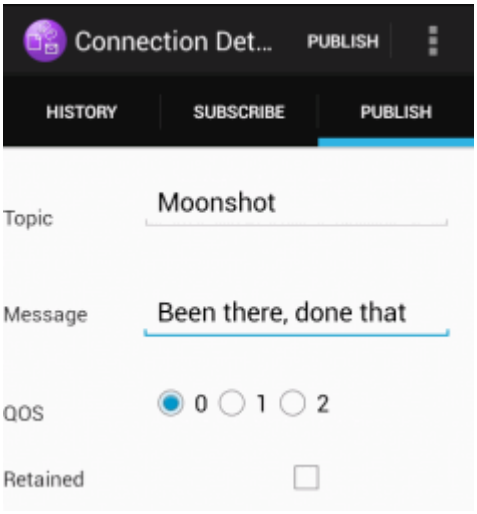
iv) Kliknij kartę **Historia**.

Historia obejmuje teraz subskrypcję:



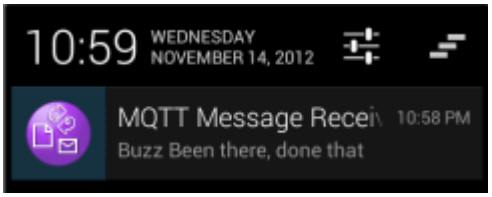
d) Teraz publikuj w tym samym temacie.

i) Kliknij kartę **Publikuj** i wprowadź ten sam łańcuch tematu, co w przypadku subskrybowania. Wprowadź komunikat.

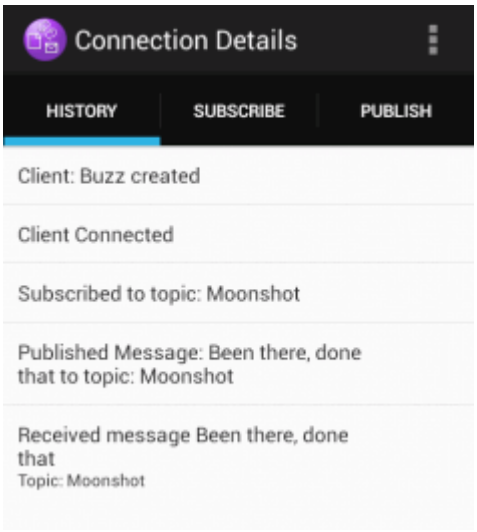


ii) Kliknij działanie **Publikuj** .

Przez krótki czas wyświetlane są dwa komunikaty: "Opublikowany" , a następnie "Zasubskrybowany". Publikacja zostanie wyświetlona w obszarze statusu (w celu otwarcia okna statusu należy przeciągnąć pasek separatora).



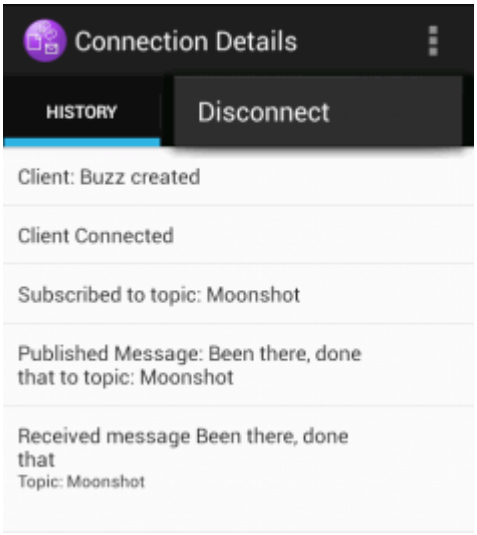
iii) Kliknij kartę **Historia** , aby wyświetlić pełną historię.



e) Odłącz instancję klienta.

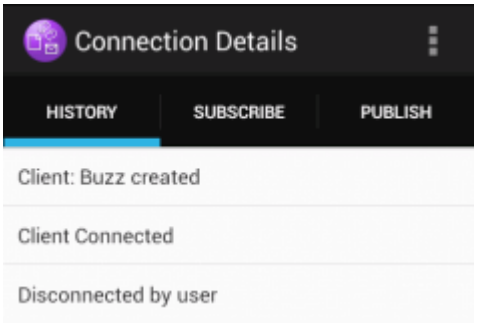
i) Kliknij ikonę menu na pasku działań.

Program Przykładowa aplikacja klienta MQTT Java dla produktu Android dodaje przycisk **Rozłącz** do okna MQTT **Szczegóły połączenia** .

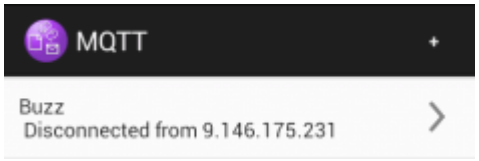


ii) Kliknij przycisk **Rozłącz**.

Status połączenia zmienia się na rozłączony:



f) Kliknij przycisk **Wstecz** , aby powrócić do listy sesji Przykładowa aplikacja klienta MQTT Java .



- Kliknij znak plus, aby rozpocząć nową sesję Przykładowa aplikacja klienta MQTT Java .
- Kliknij odłączonego klienta, aby go ponownie połączyć.
- Kliknij przycisk **Wstecz** , aby wrócić do startera.

g) Kliknij przycisk zadania, aby wyświetlić uruchomione aplikacje. Znajdź Przykładowa aplikacja klienta MQTT Java. Przeciągnij ikonę poza ekran, aby ją zamknąć.

Co dalej

Jeśli przykładową aplikację utworzono samodzielnie, można rozpocząć tworzenie własnych aplikacji Android , które wywołują biblioteki MQTT w celu wymiany komunikatów. Aplikacje Android można modelować w klasach w programie `mqttExerciser`. Aby przestudiować przykład, należy wygenerować dokumentację Javadoc dla klas w systemach `com.ibm.msg.android` i `com.ibm.msg.android.service` w projekcie `mqttExerciser` .

Informacje pokrewne

Zarządzanie projektami w środowisku Eclipse za pomocą narzędzia ADT

Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript

Można rozpocząć od Klient przesyłania komunikatów MQTT dla produktu JavaScript , wyświetlając przykładową stronę główną klienta przesyłania komunikatów i przeglądając zasoby, do których odsyłacze te są dostępne. Aby wyświetlić tę stronę główną, należy skonfigurować serwer MQTT w taki sposób, aby akceptować połączenia z serwera Przykładowe strony JavaScript klienta przesyłania komunikatów MQTT, a następnie należy wpisać adres URL skonfigurowany na serwerze w przeglądarce WWW. Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zostanie uruchomiony automatycznie na urządzeniu, a zostanie wyświetlona przykładowa strona główna klienta przesyłania komunikatów. Ta strona zawiera odsyłacze do programów narzędziowych, dokumentacji interfejsu programistycznego, kursu i innych przydatnych informacji.

Zanim rozpoczniesz

W przypadku zaawansowanego użycia lub użycia w środowisku produkcyjnym użytkownik chce zmienić kształt lub usunąć przykładową stronę główną klienta przesyłania komunikatów. Należy pamiętać, że interfejsy użytkownika wynikające z przykładowego kodu nie są uzasadnione, aby były zgodne z wszelkimi normami dostępności lub wymaganiami dotyczącymi ułatwień dostępu.

Do obsługi serwera Klient przesyłania komunikatów MQTT dla produktu JavaScriptpotrzebny jest serwer MQTT . Ten serwer musi obsługiwać protokół MQTT V3.1 w systemie WebSockets. IBM MessageSight, and IBM WebSphere MQ Version 7.5.0, Fix Pack 1 and later versions, support the MQTT protocol over WebSockets. Patrz sekcja “Pierwsze kroki z serwerami MQTT” na stronie 143. Aby zainstalować produkt IBM WebSphere MQ w celu uzyskania bezpłatnej 90-dniowej oceny, należy zapoznać się z “instalowanieIBM WebSphere MQ” na stronie 145.

WebSocket protocol jest w ostatnim czasie ustanawiany. Jeśli między klientem a serwerem jest firewall, należy sprawdzić, czy nie blokuje on ruchu produktu WebSockets. Podobnie, jeśli przeglądarka nie obsługuje jeszcze produktu WebSocket protocol[1]Nie będzie można korzystać z programu narzędziowego klienta ani kursów dostępnych na przykładowej stronie głównej klienta przesyłania komunikatów. Tabela Tabela 1 na stronie 25 zawiera listę przeglądarek, których najnowsza wersja została przetestowana i wyświetlona w celu współpracy z klientem przesyłania komunikatów.

Tabela 1. Obsługiwane przeglądarki do użycia z produktem Klient przesyłania komunikatów MQTT dla produktu JavaScript

Android	**iOS**	**Linux**	**Windows**
Firefox for Android 19.0 i nowsze Chrome for Android 25.0 i nowsze	Safari 6.0 i nowsze Chrome 14.0 i nowsze	Firefox 6.0 i nowsze Chrome 14.0 i nowsze	Firefox 6.0 i nowsze Chrome 14.0 i nowsze

O tym zadaniu

Większość kroków w tym zadaniu ma na celu skonfigurowanie serwera MQTT . Aby uzyskać dostęp do klienta przesyłania komunikatów w systemie JavaScript , należy uruchomić przeglądarkę, która obsługuje WebSocket protocol.

W systemie IBM WebSphere MQwykonaj kroki, aby włączyć program IBM WebSphere MQ Telemetry , tworząc przykładowe kanały. Połącz się z przykładowym domyślnym kanałem MQTT WebSockets na porcie 1883. Przykładowy adres URL strony głównej klienta przesyłania komunikatów to `http://`*`hostname`*`:1883` w systemie IBM WebSphere MQ.

W systemie IBM MessageSightzainstaluj i skonfiguruj urządzenie, skonfiguruj koncentrator przesyłania komunikatów, aby akceptować połączenia, a następnie utwórz punkt końcowy MQTT WebSockets .

[1] W szczególności, jeśli nie obsługuje standardu RFC 6455 (WebSocket).

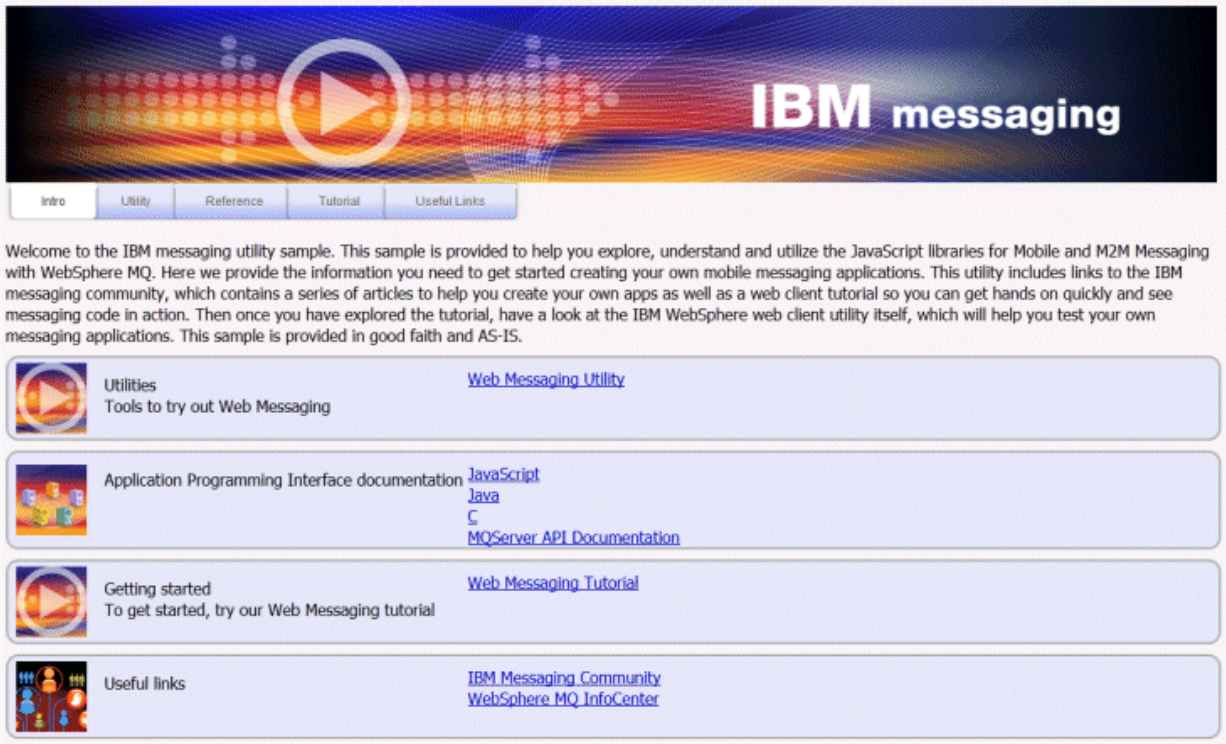
Procedura

1. Pobierz Mobile Messaging and M2M Client Pack, a następnie wybierz serwer MQTT , z którym można połączyć aplikację kliencką.

   Patrz sekcja “Pierwsze kroki z klientami MQTT” na stronie 11.
2. Skonfiguruj serwer MQTT w taki sposób, aby akceptować połączenia z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript .
   - W systemie IBM WebSphere MQ:
     - Jeśli menedżer kolejek produktu IBM WebSphere MQ jest już skonfigurowany dla produktu MQTT, należy zmodyfikować protokół w definicji kanału, aby obsługiwać zarówno protokół MQTT, jak i protokół HTTP. Patrz **ALTER CHANNEL**.
     - Aby utworzyć menedżer kolejek produktu IBM WebSphere MQ i skonfigurować przykładowy punkt końcowy MQTT WebSockets , wykonaj jedną z następujących czynności:
       - “Konfigurowanie usługi MQTT z poziomu wiersza komend” na stronie 147
       - “Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer” na stronie 150
3. Otwórz przeglądarkę WWW na urządzeniu.
4. Wpisz adres URL przykładowej strony głównej klienta przesyłania komunikatów.
   - W systemie IBM WebSphere MQjest to `http://`*`hostname`*`:1883`
   - W systemie IBM MessageSightjest to `http://`*`hostname`*`:`*`port`*

     gdzie *`hostname`* to nazwa DNS lub adres IP gniazda Ethernet skonfigurowanego na urządzeniu IBM MessageSight jako punkt końcowy, do którego klient ma nawiązać połączenie, a *`port`* to numer portu TCP/IP przypisany do punktu końcowego dla klienta.

   Zostanie wyświetlona przykładowa strona główna klienta przesyłania komunikatów.



Rysunek 9. Przykładowa strona główna produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript

Wyniki

Skonfigurowano kanał MQTT dla produktu WebSockets.

Na przykładowej stronie głównej produktu Klient przesyłania komunikatów MQTT dla produktu JavaScriptkliknij opcję **Program narzędziowy przesyłania komunikatów WWW** , aby wypróbować różne funkcje w interfejsie API klienta przesyłania komunikatów. Na przykład można połączyć się z menedżerem kolejek, zasubskrybować komunikaty, a następnie opublikować niektóre komunikaty. Można również kliknąć opcję **Kurs przesyłania komunikatów WWW** , aby dowiedzieć się, w jaki sposób utworzyć stronę WWW, która wywołuje klient przesyłania komunikatów produktu MQTT dla interfejsu API produktu JavaScript .

Pojęcia pokrewne

“Klient przesyłania komunikatów MQTT dla produktu JavaScript i aplikacje WWW” na stronie 122
“Jak programować aplikacje przesyłania komunikatów w produkcie JavaScript” na stronie 126

Zadania pokrewne

“Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets” na stronie 81

Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

Pierwsze kroki z klientem MQTT dla języka C

Na dowolnej platformie, na której można skompilować źródło C, należy pobrać i uruchomić przykładowy klient MQTT dla języka C. Sprawdź, czy można uruchomić przykładowy klient MQTT dla języka C z produktem IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

Zanim rozpoczniesz

- Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
- W przypadku obsługiwanych i referencyjnych klientów MQTT dla platform C. Więcej informacji na ten temat zawiera sekcja Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.

O tym zadaniu

Wykonaj to zadanie, aby skompilować i uruchomić przykładowy klient MQTT dla języka C w systemie Windows z poziomu wiersza komend lub produktu Microsoft Visual Studio 2010. Produkt Microsoft Visual Studio 2010 jest również używany do kompilowania klienta w przykładzie wiersza komend. Zmodyfikuj skrypty wiersza komend, aby skompilować i uruchomić przykład na innych platformach.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 . Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Należy zainstalować środowisko programistyczne języka C na platformie, na której przeprowadzane są operacje budowania.

   Pliki makefile w przykładach w tym temacie mają następujące narzędzia:

   - iOSW przypadku produktu iOSw systemie Apple Mac z systemem operacyjnym OS X 10.8.2 z narzędziami programistycznym iOS z poziomu Xcode.
   - LinuxW przypadku produktu Linuxw wersji gcc 4.4.6 z produktu Red Hat® Enterprise Linux w wersji 6.2.

     Minimalny obsługiwany poziom biblioteki `glibc` C to 2.12, a jądro Linux to 2.6.32.
   - WindowsW przypadku produktu Microsoft Windowsw wersji Visual Studio 10.0.
3. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu `MQTT`. Ścieżka do tego folderu jest określana w tym miejscu jako *`sdkroot`*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *`sdkroot`*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *`sdkroot`*\SDK.
4. Opcjonalne: Wykonaj kroki opisane w sekcji “Budowanie klienta MQTT dla bibliotek C” na stronie 32.

   Wykonaj ten krok tylko wtedy, gdy Mobile Messaging and M2M Client Pack nie zawiera biblioteki klienta C dla platformy docelowej.
5. Skompiluj i uruchom przykładową aplikację C klienta MQTT , `MQTTV3Sample.c`.

   - Z poziomu wiersza komend wykonaj kroki opisane w sekcji “Skompiluj i uruchom przykładową aplikację C klienta MQTT z poziomu wiersza komend” na stronie 29.
   - Z poziomu środowiska IDE należy wykonać kroki opisane w sekcji “Skompiluj i uruchom przykładową aplikację C klienta MQTT z produktu Microsoft Visual Studio.” na stronie 30.

Skompiluj i uruchom przykładową aplikację C klienta MQTT z poziomu wiersza komend

Skompiluj i uruchom przykładową aplikację C klienta MQTT z poziomu wiersza komend. Przykład znajduje się w pakiecie SDK produktu MQTT . Demonstruje on publikator i subskrybent produktu MQTT .

Zanim rozpoczniesz

Zainstaluj środowisko programistyczne języka C, na przykład produkt Microsoft Visual Studio 2010, który jest używany w tym przykładzie.

O tym zadaniu

Skompiluj i uruchom przykład C, `MQTTV3Sample`, w podkatalogu klientów SDK, *sdkroot*`\SDK\clients\c\samples`.

Procedura

Utwórz skrypt w katalogu przykładów klienta, aby skompilować i uruchomić produkt `Sample` na wybranej platformie.

Poniższy skrypt kompiluje i uruchamia przykład na 32-bitowej platformie produktu Windows , zbudowanej przy użyciu produktu Microsoft Visual Studio 2010. Uruchom skrypt z podkatalogu *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
```

Wyniki

Publikator i subskrybent zapisuje dane wyjściowe do swoich okien komend:

```
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 . . .
```

Rysunek 10. Dane wyjściowe publikatora

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

Rysunek 11. Dane wyjściowe subskrybenta

Skompiluj i uruchom przykładową aplikację C klienta MQTT z produktu Microsoft Visual Studio.

Skompiluj i uruchom przykładową aplikację C klienta MQTT z produktu Microsoft Visual Studio. Przykład znajduje się w Mobile Messaging and M2M Client Pack. Demonstruje on publikator i subskrybent produktu MQTT .

Zanim rozpoczniesz

W tym przykładzie użyto programu Microsoft Visual Studio 2010. Innych środowisk programistycznych języka C można używać na serwerze Windows i na innych platformach, na przykład Eclipse IDE for C/C++ Developers.

O tym zadaniu

Skompiluj i uruchom przykład C, `MQTTV3Sample` z programem Microsoft Visual Studio 2010. `MQTTV3Sample.c` znajduje się w podkatalogu klientów SDK, *`sdkroot`*`\SDK\clients\c\samples`.

Procedura

1. Uruchom program Microsoft Visual Studio.
2. Utwórz nowy projekt na podstawie istniejącego kodu.
   a) Kliknij opcję **Plik** > **Nowy** > **Projekt z istniejącego kodu**.
   b) Wybierz opcję **Visual C++** (Visual C++) jako typ projektu do utworzenia.
   c) Kliknij przycisk **Dalej**.
3. Określ parametry w oknie **Położenie projektu i pliki źródłowe** .
   a) Kliknij przycisk **Przeglądaj** i znajdź katalog *`sdkroot`*`\SDK\clients\c\samples` .
   b) Nadaj nazwę projektowi `MQTTV3Sample`.
   c) Kliknij przycisk **Dalej**.
4. Wybierz opcję **Projekt aplikacji konsoli** na liście **Typ projektu** . Kliknij przycisk **Zakończ**.
5. Skonfiguruj tylko konfigurację debugowania.

   Domyślnie program Microsoft Visual Studio tworzy zarówno konfigurację wydania, jak i debugowania. W tym kursie zostanie skonfigurowana konfiguracja debugowania. Aby wyłączyć błędy budowania, usuń zaznaczenie opcji **Buduj** dla konfiguracji wersji.

   a) Kliknij opcję **Project** > **MQTTV3Sample Properties** > **Configuration Manager**(Projekt > Właściwości MQTTV3Sample). Wybierz opcję **Zwolnij** jako **Konfiguracja rozwiązania aktywnego** i usuń zaznaczenie opcji **Buduj**.
   b) Wybierz opcję **Debuguj** jako opcję **Konfiguracja aktywnego rozwiązania** > **Zamknij**.

      Należy sprawdzić, czy konfiguracja debugowania jest modyfikowana we wszystkich poniższych krokach.
6. Zmodyfikuj ustawienia **C/C++** na stronie właściwości **MQTTV3Sample**.
   a) W oknie **MQTTV3Sample Property Pages** (Strony właściwości MQTTV3Sample) otwórz opcję **Configuration properties** > **C/C++** > **General**(Właściwości konfiguracyjne-C/C++-Ogólne)
   b) Na liście właściwości ogólnych kliknij opcję **Additional Include Directories**(Dodatkowe katalogi włączeń), a następnie dodaj ścieżkę do katalogu *`sdkroot`*`\SDK\clients\c\include` i kliknij przycisk **Apply**(Zastosuj).
7. Zmodyfikuj ustawienia **Linker**
   a) Otwórz kolejno opcje **Właściwości konfiguracyjne** > **Linker** > **Ogólne**.
   b) Na liście właściwości ogólnych kliknij opcję **Dodatkowe katalogi biblioteki**, a następnie dodaj ścieżkę do katalogu *`sdkroot`*`\SDK\clients\c\windows_ia32` .

c) Na liście właściwości konsolidatora kliknij opcję **Wiersz komend**. Wpisz `mqttv3c.lib` w obszarze wprowadzania danych **Opcje dodatkowe** i kliknij przycisk **Zastosuj**.

8. Usuń plik źródłowy `MQTTV3SSample.c` z projektu.
   a) Otwórz folder **MQTTV3sample** > **Pliki źródłowe** w oknie **Eksplorator rozwiązań** .
   b) Kliknij **MQTTV3SSample.c** > **Wyklucz z projektu** przyciskiem
9. Zbuduj projekt `MQTTV3Sample` .
   a) Kliknij prawym przyciskiem myszy projekt **MQTTV3sample** w eksploratorze rozwiązań, a następnie kliknij opcję **Build**(Buduj).

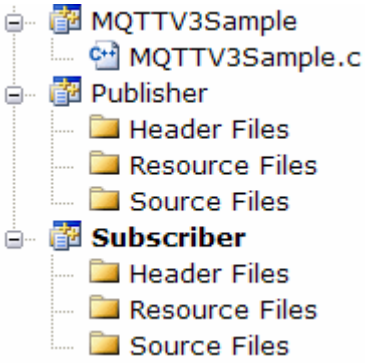
      Budowanie kończy się bez żadnych błędów.
10. Dodaj dwa nowe projekty, aby uruchomić produkt **`MQTTV3Sample`** jako instancję środowiska wykonawczego Subscriber i Publisher.

    Projekty publikatora i subskrybenta zawierają komendy służące do uruchamiania produktu **`MQTTV3Sample`**. Nie są one zbudowane i nie zawierają kodu.

    a) W **Eksploratorze rozwiązań**kliknij prawym przyciskiem myszy opcję **Rozwiązanie ` MQTTV3Sample `** > **Dodaj** > **Nowy projekt**.
    b) Wpisz `Subscriber` w polu **Nazwa** . Pozostaw zaznaczone pole **Aplikacja konsoli produktu Win32** . Kliknij przycisk **OK**.

       Zostanie uruchomiony **Kreator aplikacjiWin32** .
    c) W kreatorze aplikacji **Win32 Application Wizard**kliknij przycisk **Dalej**. Zaznacz pole wyboru **Pusty projekt** > **Zakończ** .
    d) Powtórz te kroki, aby dodać projekt publikatora.
    e) Kliknij prawym przyciskiem myszy projekt Subscriber i kliknij opcję **Ustaw jako projekt StartUp**

    Okno **Eksplorator rozwiązań** jest wyświetlane w programie Rysunek 12 na stronie 31.



Rysunek 12. Rozwiązanie MQTTV3Sample

11. Skonfiguruj strony właściwości Subscriber.
    a) Kliknij prawym przyciskiem myszy opcję **Subskrybent** w eksploratorze rozwiązań **Właściwości** > **Właściwości konfiguracji** > **Właściwości** > **Debugowanie** .

       Sprawdź, czy tytuł okna to **Strony właściwości subskrybenta**.
    b) Kliknij opcję **Środowisko**. Wpisz `path=%path%;`*`sdkroot`*`\SDK\clients\c\windows_ia32` i kliknij przycisk **Zastosuj**.

       Zmień *`sdkroot`* , aby dopasować je do używanego środowiska.
    c) Kliknij opcję **Command**(Komenda), a następnie zastąp obiekt `$(TargetPath)` ścieżką do modułu `MQTTV3Sample` .

       Na przykład *`sdkroot`*`\SDK\clients\c\samples\debug\MQTTV3Sample`

       Zmień *`sdkroot`* , aby dopasować je do używanego środowiska.

d) Kliknij opcję **Argumenty komendy** i wpisz `-a subscribe -b localhost -p 1883`, a następnie kliknij przycisk **Zastosuj**.

12. Skonfiguruj strony właściwości publikatora.

    Wskazówka: Projekt stron właściwości można przełączać, klikając projekty w oknie **Eksplorator rozwiązań** .

    a) Powtórz kroki subskrybenta dla publikatora.

    Argument komendy to `-b localhost -p 1883`

13. Zatrzymaj proces budowania, który buduje projekty produktu Publisher and Subscriber.

    a) Na stronach właściwości dowolnego projektu kliknij opcję **Configuration Manager** (Menedżer konfiguracji) i usuń zaznaczenie opcji **Build** (Buduj) dla publikatora i subskrybenta w konfiguracjach Release i Debug (Wersja i debugowanie). Naciśnij przycisk **Zamknij**.

14. Uruchom przykład.

    a) Kliknij przycisk **F5** , aby uruchomić subskrybent.

    b) Kliknij prawym przyciskiem myszy pozycję **Publisher** w eksploratorze rozwiązań, **Debuguj** > **Uruchom nową instancję** .

Wyniki

Dane wyjściowe publikatora i subskrybenta w oknach komend. Program Visual Studio zamknie okno publikatora. Sprawdź okno subskrybenta, które jest wyświetlane na poniższym rysunku, a następnie zamknij okno subskrybenta.

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

Rysunek 13. Dane wyjściowe subskrybenta

Co dalej

Zbuduj i uruchom asynchroniczny publikator i subskrybent. Przykłady to: `MQTTV3ASample.c` i `MQTTV3ASSample.c` w podręczniku *sdkroot*`\SDK\clients\c\samples`.

Budowanie klienta MQTT dla bibliotek C

Aby zbudować klienta MQTT dla bibliotek C, należy wykonać następujące kroki. Temat zawiera przełączniki kompilacji i łącza dla wielu platform, a także przykłady budowania bibliotek w systemach iOS i Windows.

Zanim rozpoczniesz

1. Należy zbudować bibliotekę klienta C tylko wtedy, gdy jest to konieczne. Utwórz dowiązanie do wstępnie zbudowanych bibliotek klienta w pakiecie SDK (Software Development Kit) w podkatalogu `SDK\clients\c` , jeśli jeden z nich jest zgodny z platformą docelową.
2. Skonfiguruj serwer MQTT w celu przetestowania biblioteki, która została zbudowana za pomocą MQTT client sample C app. Więcej informacji zawiera sekcja “Pierwsze kroki z serwerami MQTT” na stronie 143. Sprawdź konfigurację serwera, uruchamiając jedną z przykładowych aplikacji klienta MQTT .
3. Jeśli budujesz bezpieczną wersję biblioteki C, która obsługuje (Secure Sockets Layer) SSL, należy również zbudować bibliotekę OpenSSL . Patrz “Budowanie pakietu OpenSSL” na stronie 47.

   Ważne: Pobieranie i redystrybucja pakietu OpenSSL podlega rygorystycznym regulom importu i eksportu oraz warunkom licencji Open Source. Przed podjęciem decyzji o tym, czy pobrać pakiet, należy uważnie zapoznać się z ograniczeniami i ostrzeżeniami.

O tym zadaniu

Postępuj zgodnie z instrukcjami w sekcji “Budowanie pakietu OpenSSL” na stronie 47 , aby pobrać i zbudować bibliotekę produktu OpenSSL . Aby zbudować bezpieczną wersję klienta MQTT dla biblioteki C, należy zbudować OpenSSL . Aby zbudować niezabezpieczoną wersję biblioteki MQTT , nie jest wymagane OpenSSL . Kroki te obejmują przykłady budowania biblioteki dla produktów iOS i Windows.

Zbuduj klienta MQTT dla biblioteki C, pobierając narzędzia biblioteki programowania C, a pakiet narzędzi programistycznych produktu MQTT (SDK) na platformie budowania. Napisz plik makefile, aby zbudować bibliotekę dla platformy docelowej, zawierającą opcje opisane w sekcji “Opcje budowania produktuMQTT dla różnych platform” na stronie 34. Kroki specyficzne dla platformy służące do budowania i uruchamiania pliku makefile są podane tutaj:

- iOS “Budowanie bibliotek klienta MQTT na potrzeby języka C na serwerze Apple Mac w celu użycia z urządzeniami iOS” na stronie 35
- Windows “Budowanie bibliotek produktu MQTT w systemie Windows” na stronie 41

Procedura

1. Należy zainstalować środowisko programistyczne języka C na platformie, na której przeprowadzane są operacje budowania.

   Pliki makefile w przykładach w tym temacie mają następujące narzędzia:

   - iOS W przypadku produktu iOSw systemie Apple Mac z systemem operacyjnym OS X 10.8.2 z narzędziami programistycznym iOS z poziomu Xcode.
   - Linux W przypadku produktu Linuxw wersji gcc 4.4.6 z produktu Red Hat Enterprise Linux w wersji 6.2.

     Minimalny obsługiwany poziom biblioteki `glibc` C to 2.12, a jądro Linux to 2.6.32.
   - Windows W przypadku produktu Microsoft Windowsw wersji Visual Studio 10.0.
2. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.
   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu MQTT. Ścieżka do tego folderu jest określana w tym miejscu jako *`sdkroot`*.
   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *`sdkroot`*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *`sdkroot`*\SDK.
3. Rozwiń kod źródłowy dla klienta MQTT dla bibliotek C.

   Skompresowany plik kodu źródłowego to *`sdkroot`*`\SDK\clients\c\source.zip`.
4. Opcjonalne: Zbuduj OpenSSL.

   Patrz “Budowanie pakietu OpenSSL” na stronie 47.
5. Zbuduj klienta MQTT dla bibliotek C.

   Komendy i opcje służące do budowania bibliotek są wymienione w sekcji “Opcje budowania produktuMQTT dla różnych platform” na stronie 34.

   Wykonaj kroki opisane w poniższych przykładach, aby napisać plik makefile w celu zbudowania klienta MQTT dla bibliotek C dla platformy docelowej.

   - “Budowanie bibliotek klienta MQTT na potrzeby języka C na serwerze Apple Mac w celu użycia z urządzeniami iOS” na stronie 35
   - “Budowanie bibliotek produktu MQTT w systemie Windows” na stronie 41

Opcje budowania produktuMQTT dla różnych platform

Poniższa tabela zawiera listę opcji kompilatora i kompilacji w celu zbudowania klienta MQTT dla bibliotek języka C na różnych platformach.

Tabela 2. Opcje budowania produktuMQTT dla różnych platform

Platforma	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	-m64
Linux x86-64		-fPIC -Os -Wall -I *MQTTCLIENT_DIR*		
Linux x86-32				-m32
Linux ARM (glibc)	arm-linux-gcc			
Linux ARM (uclibc)	arm-unknown-linux -uclibcgnueabi-gcc			
Windows (wersja 32-bitowa)	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)	

Tabela 2. Opcje budowania produktuMQTT dla różnych platform (kontynuacja)

Platforma	Compiler	Compiler Options	Linker Options	Extra Options
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`			
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 *Budowanie bibliotek klienta MQTT na potrzeby języka C na serwerze Apple Mac w celu użycia z urządzeniami iOS*

Wykonaj poniższe kroki, aby napisać plik makefile w celu zbudowania bibliotek klienta MQTT dla języka C na serwerze Apple Mac, aby można było go później używać z urządzeniami iOS .

Zanim rozpoczniesz

1. Zainstaluj narzędzia budowania, opracuj i uruchom plik makefile w systemie Apple Mac z systemem OS X 10.8.2lub nowszym.
2. Zainstaluj narzędzia wiersza komend dla programu Xcode, które zawierają program **make** . Pobierz narzędzia wiersza komend z Xcode.

O tym zadaniu

Utwórz plik makefile, który buduje biblioteki klienta MQTT dla języka C dla systemów iPhone lub iPad z procesorem ARMv7 lub ARMv7s , a symulator iPhone działający na procesorze bitowym i386-64 . Patrz Lista urządzeń z systemem iOS.

Wskazówka: “Lista plików makefile produktu MQTTios.mak” na stronie 40 zawiera listę kompletnych plików makefile.

1. Skopiuj i wklej listę do pliku.
2. Przekształć wiodący znak w każdym wierszu, który jest zgodny z celem, na kartę; patrz krok “8” na stronie 38.
3. Uruchom go za pomocą komendy wymienionej w kroku “9” na stronie 39 procedury.

Procedura

1. Pobierz i zainstaluj narzędzia programistyczne produktu iOS .
   a. Zaloguj się przy użyciu identyfikatora użytkownika, który ma uprawnienia administracyjne.
   b. Sprawdź, czy Apple Mac jest w wersji 10.8.2 lub nowszej.
   c. Przejdź do serwisu WWW Xcode , aby pobrać produkt Xcode ze składnicy aplikacji Mac .
   d. Zainstaluj produkt Xcode, środowisko wiersza komend i symulator.

   Jeśli składnica aplikacji Mac oferuje wiele wersji symulatora, wybierz wersję kompatybilną z poziomem produktu iOS , który ma być używany dla danej aplikacji.
2. Utwórz plik makefile `MQTTios.mak`

   Dodaj prologu:

   ```
   # Dane wyjściowe budowania są tworzone w bieżącym katalogu.
   # Parametr MQTTCLIENT_DIR musi wskazywać na katalog podstawowy zawierający kod źródłowy
   klienta MQTT.
   # Domyślnym katalogiem MQTTCLIENT_DIR jest bieżący katalog
   # domyślnym plikiem TOOL_DIR jest /Applications/Xcode.app/Contents/Developer/Platforms
   # Domyślny parametr OPENSSL_DIR to sdkroot/openssl, względny w stosunku do sdkroot/sdk/
   clients/c/mqttv3c/src
   # OPENSSL_DIR musi wskazywać na katalog podstawowy zawierający kompilację OpenSSL .
   # Przykład: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all
   ```
3. Ustaw położenie kodu źródłowego MQTT .

   Uruchom plik makefile w tym samym katalogu, w którym znajdują się pliki źródłowe produktu MQTT , lub ustaw parametr wiersza komend `MQTTCLIENT_DIR` :

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

   Dodaj następujące wiersze do pliku makefile:

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

   Przykład ustawia wartość VPATH w katalogu, w którym program **make** wyszukuje pliki źródłowe, które nie zostały jawnie zidentyfikowane; na przykład wszystkie pliki nagłówkowe, które są wymagane w kompilacji.
4. Opcjonalne: Ustaw położenie bibliotek produktu OpenSSL .

   Ten krok jest wymagany do zbudowania wersji SSL klienta MQTT dla bibliotek C.

Ustaw domyślną ścieżkę do bibliotek produktu OpenSSL w tym samym katalogu, w którym został rozwinięty pakiet SDK produktu MQTT . W przeciwnym razie ustaw parametr OPENSSL_DIR jako parametr wiersza komend.

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

Wskazówka: *OpenSSL* to katalog OpenSSL , który zawiera wszystkie podkatalogi OpenSSL . Może być konieczne przeniesienie drzewa katalogów z miejsca, w którym zostało ono rozpakowane, w celu wyeliminowania istniejących niepotrzebnych pustych katalogów nadrzędnych.

5. Ustaw katalogi narzędzi programistycznych.

   Jeśli produkt Xcode został zainstalowany w innym miejscu, należy w wierszu komend ustawić parametr TOOL_DIR .

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

6. Wybierz wszystkie pliki źródłowe, które są wymagane do zbudowania każdej biblioteki MQTT . Ponadto ustaw nazwę i położenie biblioteki MQTT do zbudowania.

   Dodaj następujący wiersz do pliku makefile, aby wyświetlić listę wszystkich plików źródłowych produktu MQTT :

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

   Wymagane są różne pliki źródłowe w zależności od tego, czy budujesz bibliotekę synchroniczną, czy też asynchroniczną, oraz od tego, czy biblioteka uwzględnia protokół SSL, czy też nie.

   Dodaj co najmniej jedną z tych linii, które zależą od celów budowania. Biblioteki współużytkowane są tworzone w katalogu darwin_x86_64 .

   - Synchroniczna, niezabezpieczona:

```
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
```

   - Synchroniczny, zabezpieczony:

```
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
```

   - Asynchroniczne, niezabezpieczone:

```
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
```

   - Zabezpieczone asynchroniczne:

```
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. Zdefiniuj kompilator i opcje kompilatora.

   Zapoznaj się z opcjami dla różnych platform, które są wyświetlane w sekcji Opcje budowania MQTT dla różnych platform.

   a) Jako kompilator należy ustawić projekt Gnu w języku C i C++ (**gcc**).

Wybierz trzy współkompilatory, aby zbudować bibliotekę dla różnych urządzeń i symulatora produktu 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) Dodaj opcje kompilatora.

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

c) Dodaj ścieżki włączania.

```
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. Zdefiniuj cele budowania.

Wskazówka: Każdy kolejny wiersz definiujący implementację elementu docelowego musi zaczynać się od znaku tabulacji.

a) Zdefiniuj cel **all** .

Element docelowy "all" buduje wszystkie biblioteki.

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

Po pierwsze, jest to domyślny cel.

a) Zbuduj synchroniczną, niezabezpieczoną bibliotekę `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 $@
```

Instrukcja `rm *.o` usuwa wszystkie pliki obiektów, które są tworzone dla każdej biblioteki. `lipo` konkatenuje wszystkie trzy biblioteki w jeden plik.

b) Zbuduj asynchroniczną, niezabezpieczoną bibliotekę `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 $@
```

Instrukcja `rm *.o` usuwa wszystkie pliki obiektów, które są tworzone dla każdej biblioteki. `lipo` konkatenuje wszystkie trzy biblioteki w jeden plik.

c) Zbuduj synchroniczną, zabezpieczoną bibliotekę `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 $@
```

Instrukcja `rm *.o` usuwa wszystkie pliki obiektów, które są tworzone dla każdej biblioteki. `lipo` konkatenuje wszystkie trzy biblioteki w jeden plik.

d) Zbuduj asynchroniczną, zabezpieczoną bibliotekę `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 $@
```

Instrukcja `rm *.o` usuwa wszystkie pliki obiektów, które są tworzone dla każdej biblioteki. `lipo` konkatenuje wszystkie trzy biblioteki w jeden plik.

e) Zdefiniuj cel **`clean`** .

Element docelowy "clean" usuwa wszystkie pliki i katalogi wygenerowane przez plik makefile.

```
.PHONY: czyste
czyste:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

9. Uruchom plik makefile.

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

Wyniki

Następujące pliki są tworzone w katalogu *sdkroot/sdk/clients/c/mqttv3c/src/* `darwin_x86_64` .

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

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

Lista plików makefile produktu `MQTTios.mak`

```
# Dane wyjściowe budowania są tworzone w bieżącym katalogu.
# Parametr MQTTCLIENT_DIR musi wskazywać na katalog podstawowy zawierający kod źródłowy klienta
MQTT.
# Domyślnym katalogiem MQTTCLIENT_DIR jest bieżący katalog
# domyślnym plikiem TOOL_DIR jest /Applications/Xcode.app/Contents/Developer/Platforms
# Domyślny parametr OPENSSL_DIR to sdkroot/openssl, względny w stosunku do sdkroot/sdk/
clients/c/mqttv3c/src
# OPENSSL_DIR musi wskazywać na katalog podstawowy zawierający kompilację OpenSSL .
# Przykład: make -f MQTTios.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src all

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

MQTTLIB = mqttv3c
SOURCE_FILES = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN = darwin_x86_64/lib${MQTTLIB} .a
MQTTLIB_S = mqttv3cs
SOURCE_FILES_S = ${filter-out ${MQTTCLIENT_DIR}/MQTTAsync.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_S = darwin_x86_64/lib${MQTTLIB_S} .a
MQTTLIB_A = mqttv3a
SOURCE_FILES_A = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c ${MQTTCLIENT_DIR}/SSLSocket.c, $
{ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_A = darwin_x86_64/lib${MQTTLIB_A} .a
MQTTLIB_AS = mqttv3as
SOURCE_FILES_AS = ${filter-out ${MQTTCLIENT_DIR}/MQTTClient.c, ${ALL_SOURCE_FILES}}
MQTTLIB_DARWIN_AS = darwin_x86_64/lib${MQTTLIB_AS} .a

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

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

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

${MQTTLIB_DARWIN_A}: ${SOURCE_FILES_A}
    -mkdir darwin_x86_64
    ${CC_armv7} ${CCFLAGS_SO_ARM} -c ${SOURCE_FILES_A} -DMQTT_ASYNC -DUSE_NAMED_SEMAPHORES
```

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

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

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

.PHONY: czyste
czyste:
    -rm -f *.obj
    -rm -f -r darwin_x86_64
```

Windows *Budowanie bibliotek produktu MQTT w systemie Windows*

Wykonaj poniższe kroki, aby napisać plik makefile w celu zbudowania bibliotek klienta MQTT dla języka C w systemie Windows.

Zanim rozpoczniesz

1. W razie potrzeby zainstaluj na swojej stacji roboczej wersję produktu **Make** , która jest kompatybilna z plikami makefile napisanych dla programu Gnu . W przeciwnym razie pobierz program Gnu i zbuduj go. Patrz Gnu Make. Serwis WWW, Make for Windows, zawiera instalowalną wersję produktu **Make** dla produktu Windows.
2. Do korzystania z czystego celu w przykładzie pliku makefile wymagane są również komendy produktu Linux dla programu Windows . Komendy Linux dla produktu Windows można uzyskać z serwisów WWW, takich jak Cygwin.

O tym zadaniu

Utwórz plik makefile, który buduje biblioteki klienta MQTT dla języka C dla 32-bitowego systemu Windows .

Wskazówka: “Lista plików makefile produktu MQTTwin.mak” na stronie 46 zawiera listę kompletnych plików makefile.

1. Skopiuj i wklej listę do pliku.
2. Przekształć wiodący znak w każdym wierszu, który jest zgodny z celem, na kartę; patrz krok “7” na stronie 44.
3. Uruchom go za pomocą komendy wymienionej w kroku “9” na stronie 45 procedury.

Procedura

1. Utwórz plik makefile `MQTTwin.mak`

   Dodaj prologu:

   ```
   # Dane wyjściowe budowania są tworzone w bieżącym katalogu.
   # Parametr MQTTCLIENT_DIR musi wskazywać na katalog podstawowy zawierający kod źródłowy
   klienta MQTT.
   # Domyślnym katalogiem MQTTCLIENT_DIR jest bieżący katalog
   # Domyślny parametr OPENSSL_DIR to sdkroot\openSSL, względny w stosunku do
   sdkroot\sdk\clients\c\mqttv3c\src
   # OPENSSL_DIR musi wskazywać na katalog podstawowy zawierający kompilację OpenSSL .
   # Przykład: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
   # Ustaw środowisko budowania, na przykład:
   # %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. Ustaw położenie kodu źródłowego MQTT .

   Uruchom plik makefile w tym samym katalogu, w którym znajdują się pliki źródłowe produktu MQTT , lub ustaw parametr wiersza komend `MQTTCLIENT_DIR` :

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

   Dodaj następujące wiersze do pliku makefile:

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

   Przykład ustawia wartość VPATH w katalogu, w którym program **make** wyszukuje pliki źródłowe, które nie zostały jawnie zidentyfikowane; na przykład wszystkie pliki nagłówkowe, które są wymagane w kompilacji.

3. Opcjonalne: Ustaw położenie bibliotek produktu OpenSSL .

   Ten krok jest wymagany do zbudowania wersji SSL klienta MQTT dla bibliotek C.

   Ustaw domyślną ścieżkę do bibliotek produktu OpenSSL w tym samym katalogu, w którym został rozwinięty pakiet SDK produktu MQTT . W przeciwnym razie ustaw parametr `OPENSSL_DIR` jako parametr wiersza komend.

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

   Wskazówka: *OpenSSL* to katalog OpenSSL , który zawiera wszystkie podkatalogi OpenSSL . Może być konieczne przeniesienie drzewa katalogów z miejsca, w którym zostało ono rozpakowane, w celu wyeliminowania istniejących niepotrzebnych pustych katalogów nadrzędnych.

4. Wybierz wszystkie pliki źródłowe, które są wymagane do zbudowania każdej biblioteki MQTT . Ponadto ustaw nazwę i położenie biblioteki MQTT do zbudowania.

   Dodaj następujący wiersz do pliku makefile, aby wyświetlić listę wszystkich plików źródłowych produktu MQTT :

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

Wymagane są różne pliki źródłowe w zależności od tego, czy budujesz bibliotekę synchroniczną, czy też asynchroniczną, oraz od tego, czy biblioteka uwzględnia protokół SSL, czy też nie.

Dodaj co najmniej jedną z tych linii, które zależą od celów budowania. Biblioteki współużytkowane i manifesty są tworzone w katalogu `windows_ia32` .

- Synchroniczna, niezabezpieczona:

```
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
```

- Synchroniczny, zabezpieczony:

```
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
```

- Asynchroniczne, niezabezpieczone:

```
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
```

- Zabezpieczone asynchroniczne:

```
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. Zdefiniuj kompilator i opcje kompilatora.

   Zapoznaj się z opcjami dla różnych platform, które są wyświetlane w sekcji Opcje budowania MQTT dla różnych platform.

   a) Ustaw program Microsoft Visual C++ jako kompilator.

```
CC = cl
```

   b) Dodaj opcje preprocesora.

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

   c) Dodaj opcje kompilatora.

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

   d) Dodaj ścieżki włączania.

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

   e) Opcjonalne: Jeśli budujesz bezpieczną bibliotekę, dodaj opcję preprocesora.

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

   f) Opcjonalne: Jeśli budujesz bezpieczną bibliotekę, dodaj pliki nagłówkowe OpenSSL .

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

   Wskazówka: Pliki nagłówkowe znajdują się w katalogu `${OPENSSL_DIR}/inc32/openssl`, ale plik `ssl.h` jest dołączony do katalogu "`openssl/ssl.h`".

6. Ustaw opcje konsolidatora i konsolidatora.

   a) Ustaw program Microsoft Visual C++ jako konsolidator.

```
LD = link
```

b) Dodaj opcje konsolidatora.

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

c) Dodaj ścieżki do bibliotek.

```
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) Dodaj pośrednie pliki wyjściowe.

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

e) Opcjonalne: Jeśli budujesz bezpieczną bibliotekę, dodaj biblioteki OpenSSL .

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

f) Opcjonalne: Jeśli budowana jest bezpieczna biblioteka, dodaj ścieżkę do biblioteki produktu OpenSSL .

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

7. Zdefiniuj cztery cele budowania.

a) Zdefiniuj cel **all** .

Wskazówka: Każdy kolejny wiersz definiujący implementację elementu docelowego musi zaczynać się od znaku tabulacji.

Element docelowy "all" buduje wszystkie biblioteki.

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

b) Zbuduj synchroniczną, niezabezpieczoną bibliotekę `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}
```

Instrukcja `-rm ${CURDIR}/MQTTAsync.obj` usuwa wszelkie `MQTTAsync.obj` utworzone dla wcześniejszego elementu docelowego. `MQTTAsync.obj` i `MQTTClient.obj` wzajemnie się wykluczają.

c) Zbuduj asynchroniczną, niezabezpieczoną bibliotekę `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}
```

Instrukcja `-rm ${CURDIR}/MQTTClient.obj` usuwa wszelkie `MQTTClient.obj` utworzone dla wcześniejszego elementu docelowego. `MQTTAsync.obj` i `MQTTClient.obj` wzajemnie się wykluczają.

d) Zbuduj synchroniczną, zabezpieczoną bibliotekę `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}
```

Instrukcja `-rm ${CURDIR}/MQTTAsync.obj` usuwa wszelkie `MQTTAsync.obj` utworzone dla wcześniejszego elementu docelowego. `MQTTAsync.obj` i `MQTTClient.obj` wzajemnie się wykluczają.

e) Zbuduj asynchroniczną, zabezpieczoną bibliotekę `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 }
```

Instrukcja `-rm $(CURDIR}/MQTTClient.obj` usuwa wszelkie `MQTTClient.obj` utworzone dla wcześniejszego elementu docelowego. `MQTTAsync.obj` i `MQTTClient.obj` wzajemnie się wykluczają.

f) Zdefiniuj cel **clean** .

Element docelowy "clean" usuwa wszystkie pliki i katalogi wygenerowane przez plik makefile.

```
.PHONY: czyste
czyste:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

8. Ustaw ścieżkę Windows , aby uruchomić plik makefile.

Ustaw części kursywy, aby dopasować ją do danej instalacji.

a) Ustaw środowisko produktu Microsoft Visual Studio.

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

b) Ustaw zmienną `Path` tak, aby zawierała program make i środowisko komend Linux .

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

9. Uruchom plik makefile.

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

Wskazówka: Znak separatora plików musi być ukośnikiem, a nie ukośnikiem odwrotnym.

Wyniki

Następujące pliki są tworzone w katalogu *sdkroot**SDK**clients**c**mqttv3c**src*\windows_ia32 .

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

Lista plików makefile produktu `MQTTwin.mak`

```
# Dane wyjściowe budowania są tworzone w bieżącym katalogu.
# Parametr MQTTCLIENT_DIR musi wskazywać na katalog podstawowy zawierający kod źródłowy klienta
MQTT.
# Domyślnym katalogiem MQTTCLIENT_DIR jest bieżący katalog
# Domyślny parametr OPENSSL_DIR to sdkroot\openSSL, względny w stosunku do
sdkroot\sdk\clients\c\mqttv3c\src
# OPENSSL_DIR musi wskazywać na katalog podstawowy zawierający kompilację OpenSSL .
# Przykład: make -f MQTTwin.mak MQTTCLIENT_DIR=sdkroot/sdk/clients/c/mqttv3c/src
# Ustaw środowisko budowania, na przykład:
# %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 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: czyste
czyste:
    -rm -f *.obj
    -rm -f -r windows_ia32
```

Budowanie pakietu OpenSSL

Przed zbudowaniem bezpiecznych bibliotek klienta MQTT dla języka C, `mqttv3cs` i `mqttv3as`należy zbudować pakiet OpenSSL . W ramach budowania tworzone są biblioteki, które są wymagane do zbudowania bezpiecznej wersji klienta MQTT dla biblioteki C oraz narzędzia do zarządzania certyfikatami OpenSSL .

Zanim rozpoczniesz

1. iOS Dostosowywanie pliku makefile produktu iOS jest przeznaczone dla urządzeń docelowych, na których działa produkt iOS6. Dostosowanie może być inne dla wcześniejszych lub późniejszych wersji produktu iOS.
2. Windows Dostosowywanie pliku makefile produktu Windows jest przeznaczone dla okien 32-bitowych.

O tym zadaniu

Pobierz i zainstaluj pakiet OpenSSL oraz wszystkie wstępnie wymagane oprogramowanie. Dostosuj pliki makefile produktu OpenSSL i zbuduj biblioteki produktu OpenSSL dla platformy docelowej. W systemach Windows i Linuxtworzy również narzędzie do tworzenia i zarządzania kluczami OpenSSL .

Procedura

1. Zainstaluj pakiet OpenSSL .
   a) Pobierz pakiet OpenSSL z programu OpenSSL

      Ważne: Pobieranie i redystrybucja pakietu OpenSSL podlega rygorystycznym regulom importu i eksportu oraz warunkom licencji Open Source. Przed podjęciem decyzji o tym, czy pobrać pakiet, należy uważnie zapoznać się z ograniczeniami i ostrzeżeniami.

   b) Rozwiń skompresowaną zawartość pliku do programu *sdkroot*.

      Na karcie **News** (Wiadomości) w serwisie OpenSSL znajduje się położenie pobierania najnowszego pakietu. Pakiet jest skompresowany jako plik tar z rozszerzeniem `tar.gz`. Po rozwinięciu pakiet tworzy folder najwyższego poziomu `openssl`*`version`*, na przykład *`openssl-1.0.1c`*. The examples refer to the path to the folder as %`openssl`% on Windows and $`openssl` on iOS; for example, on Windows, %`openssl`% is *`sdkroot`*`\openssl-1.0.1c`.

      Wskazówka: Sprawdź ścieżkę do katalogu, który został utworzony przez wyodrębnienie pakietu OpenSSL . Niektóre pakiety mają zduplikowane poziomy folderu `openssl`*`version`* .

2. Windows
   Opcjonalne: W systemie Windowspobierz i zainstaluj produkt Perl. Patrz perl.org.

   Na przykład plik perl został pobrany z programu ActivePerl – zasoby do pobrania.

3. iOS

Opcjonalne: W systemie iOSutwórz trzy katalogi.

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

W przypadku produktu iOS należy zbudować pakiet OpenSSL dla trzech różnych platform sprzętowych.

4. Wygeneruj plik makefile programu OpenSSL w celu zbudowania pakietu OpenSSL dla używanego sprzętu i systemu operacyjnego.
   a) Otwórz okno komend w katalogu %openssl% lub $openssl .
   b) Uruchom komendę **Configure** perl z odpowiednimi parametrami.
   - 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

   W systemie iOSdostosuj wygenerowany plik makefile OpenSSL dla różnych urządzeń Apple .
   a) Utwórz trzy kopie wygenerowanego pliku makefile, $openssl/Makefile

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

   b) Zmień instrukcję "CC=gcc" w każdym pliku makefile.

   Instrukcja CC=gcc znajduje się w wierszu 62, lub w przypadku ich przerwania. Zmień je na następujące komendy:

   $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) Zmień instrukcję "CFLAG=..." w każdym pliku makefile.

   Instrukcja znajduje się na linii 63 lub na nich (w podziale na trzy linie do czytelności):

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

   Położenie pokazanych pakietów SDK jest zależne od opcji instalacji produktu Xcode . Wersja SDKs jest zależna od poziomu systemu operacyjnego, dla którego jest on przeznaczony do tworzenia pliku makefile.

Symulator iPhone
Plik makefile programu iPhone simulator to $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
Pliki makefile produktu iOS to $openssl/Makefile_arm7 i $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. Uruchom wygenerowany plik makefile.

- Windows

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

- iOS W systemie 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
```

Wyniki

Kompilacja generuje biblioteki współużytkowane, biblioteki lib i pliki nagłówkowe, które są wymagane do budowania bezpiecznych wersji biblioteki klienta produktu MQTT dla systemu C.

Pierwsze kroki z klientem MQTT dla języka C w systemie iOS

Dowiedz się, jak uzyskać aplikacje iOS w celu wymiany komunikatów z serwerem MQTT . W przypadku urządzeń iOS (tj. iPhone i iPad) należy zbudować bibliotekę klienta MQTT dla języka C z kodu źródłowego, który jest udostępniany jako część pakietu narzędzi programistycznych produktu MQTT .

Zanim rozpoczniesz

1. Odsyłacz do centrum iOS Dev Center i informacje na temat tworzenia aplikacji dla produktu iOS.
2. Uzyskaj Apple Mac z systemem OS X 10.8.2lub nowszym, aby uruchomić zintegrowane środowisko programistyczne produktu Xcode (IDE).
3. (Opcjonalnie) Skonfiguruj środowisko programistyczne języka C w systemie Windows lub Linux. Przed opracowaniem aplikacji MQTT iOS warto zbudować i uruchomić przykładowe aplikacje C klienta produktu MQTT na serwerze Windows lub Linux . Można również zbadać przykładowy kod źródłowy bez tworzenia próbek.
4. Informacje na temat obsługiwanych i referencyjnych platform klienckich produktu MQTT dla języka C można znaleźć w sekcji Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.

5. Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .

O tym zadaniu

Procedura prowadzi użytkownika przez następujące kroki:

1. Informacje na temat programowania dla produktu MQTT przez badanie, budowanie i uruchamianie przykładowych aplikacji klienta MQTT oraz bibliotek klienta produktu MQTT w systemie C.
2. Install the Xcode development environment for iOS on Apple Mac.
3. Wykonaj zadanie “Budowanie klienta MQTT dla bibliotek C” na stronie 32 , aby zbudować klienta MQTT dla bibliotek C dla urządzeń iOS .

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 . Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu MQTT. Ścieżka do tego folderu jest określana w tym miejscu jako *sdkroot*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *sdkroot*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *sdkroot*\SDK.
3. Opcjonalne: Zapoznanie się z interfejsem API produktu MQTT przez badanie MQTT client sample C app.

   a) Build the synchronous MQTT client sample C app `MQTTV3sample.c` for Windows or Linux. Więcej informacji zawiera sekcja “Pierwsze kroki z klientem MQTT dla języka C” na stronie 27.

   b) Nawiąże połączenie z serwerem MQTT version 3 , a następnie publikuj i zasubskrybuj tematy na serwerze.

   c) Zapoznaj się z kodem źródłowym i dokumentacją interfejsu API produktu MQTT . W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT.

      Sekcja zawiera informacje na temat tworzenia i wznawiania klientów produktu MQTT oraz publikowania i subskrybowania tematów produktu MQTT poprzez zapoznanie się z przykładem synchronicznym. Próbka synchroniczna jest prostsza niż próbka asynchroniczna. Jeśli produkt MQTT nie został wcześniej zaprogramowany, należy napisać synchroniczny program MQTT w celu zapoznania się z modelem programistycznym MQTT i interfejsem API.

   d) Zbuduj asynchroniczną bibliotekę klienta MQTT dla języka C w systemie Windows lub Linux. Więcej informacji zawiera sekcja “Budowanie klienta MQTT dla bibliotek C” na stronie 32.

   e) Zbuduj i uruchom asynchroniczną aplikację klienta publikowania i subskrybowania klienta MQTT .

   f) Zapoznaj się z przykładowym kodem źródłowym aplikacji asynchronicznej C aplikacji klienta MQTT i dokumentacją odwołania do produktu MQTT .

      Aby napisać aplikację MQTT dla urządzenia przenośnego, należy użyć interfejsu asynchronicznego. Dobrze napisane aplikacje, które wywołują interfejs asynchroniczny, są bardziej reaktywne i rozciągają żywotność baterii dalej niż aplikacje zapisane w interfejsie synchronicznym.

      Interfejs asynchroniczny ma dwa stopnie asynchroniczności:

i) Pierwszym stopniem jest odblokowanie aplikacji, gdy biblioteka klienta MQTT oczekuje na publikacje z serwera.

ii) Drugi stopień polega na odblokowaniu aplikacji, gdy biblioteka klienta łączy się z serwerem, tworzy subskrypcje i publikuje publikacje.

4. Pobierz i zainstaluj narzędzia programistyczne produktu iOS .
   a. Zaloguj się przy użyciu identyfikatora użytkownika, który ma uprawnienia administracyjne.
   b. Sprawdź, czy Apple Mac jest w wersji 10.8.2 lub nowszej.
   c. Przejdź do serwisu WWW Xcode , aby pobrać produkt Xcode ze składnicy aplikacji Mac .
   d. Zainstaluj produkt Xcode, środowisko wiersza komend i symulator.

   Jeśli składnica aplikacji Mac oferuje wiele wersji symulatora, wybierz wersję kompatybilną z poziomem produktu iOS , który ma być używany dla danej aplikacji.
5. Zbuduj biblioteki klienta MQTT dla języka C w systemie iOS. Patrz sekcja “Budowanie klienta MQTT dla bibliotek C” na stronie 32.

Co dalej

1. Sprawdź, czy biblioteki klienta MQTT dla języka C zostały zbudowane:
   a. Użyj środowiska programistycznego produktu Xcode , aby skompilować asynchroniczną bibliotekę MQTT client sample C appi utworzyć połączenie z niezabezpieczoną biblioteką asynchroniczną klienta MQTT dla systemu C.
   b. Użyj środowiska programistycznego Xcode , aby uruchomić asynchroniczną przykładową aplikację C klienta MQTT na urządzeniu iOS . Połącz próbkę z skonfigurowanym serwerem MQTT version 3 . Patrz “Konfigurowanie usługi MQTT z poziomu wiersza komend” na stronie 147.
2. Utwórz aplikację C klienta MQTT dla produktu iOS. Przykłady kodowania dla asynchronicznego aplikacji C mogą okazać się pomocne. Przykłady to: `MQTTV3ASample.c` i `MQTTV3ASSample.c` w podręczniku *`sdkroot`*`\SDK\clients\c\samples`. Jako ćwiczenie należy rozpocząć od zaimplementowania przykładu publikowania/subskrypcji produktu MQTT .

   Wskazówka: Aby uzyskać informacje na temat wyglądu i czynności aplikacji, należy przejrzeć zrzuty ekranu produktu MQTT client sample C app. Patrz sekcja “Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android” na stronie 19.

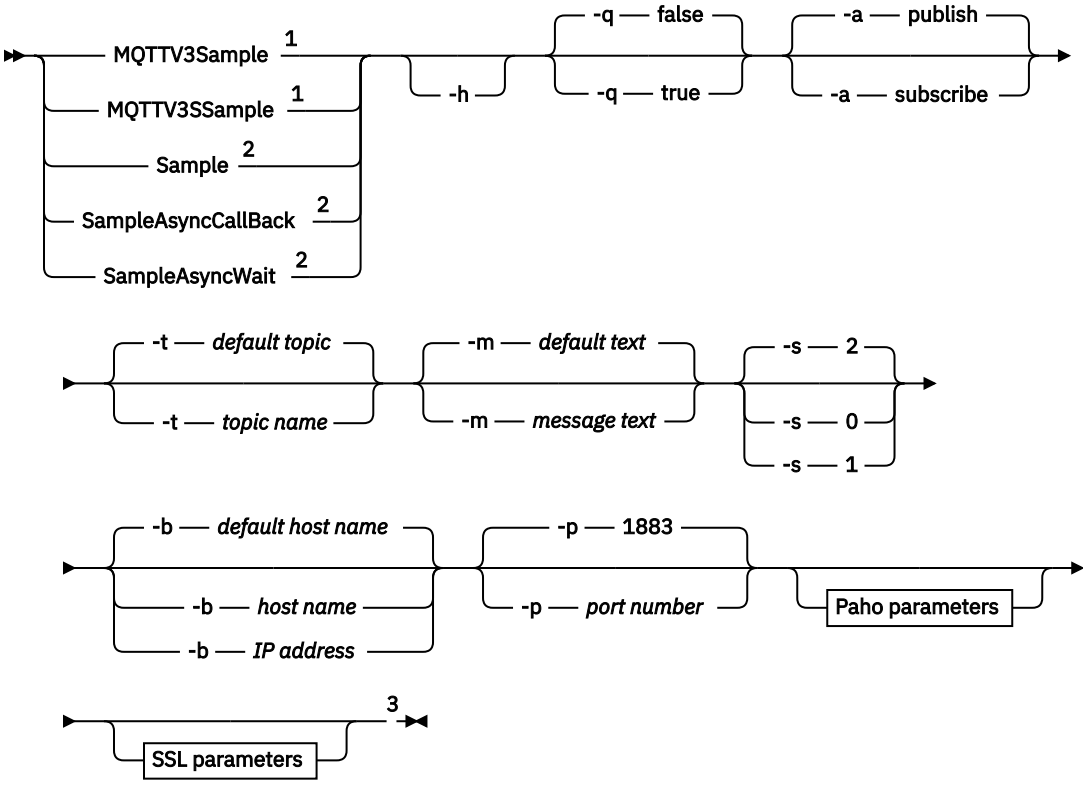
Programy przykładowe wiersza komend produktu MQTT

Składnia i parametry przykładowych programów wiersza komend produktu MQTT .

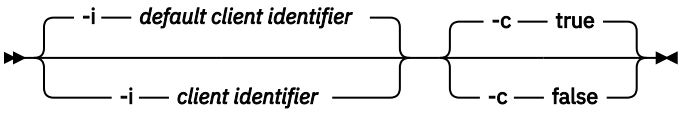
Przeznaczenie

Publikowanie i subskrybowanie tematu.

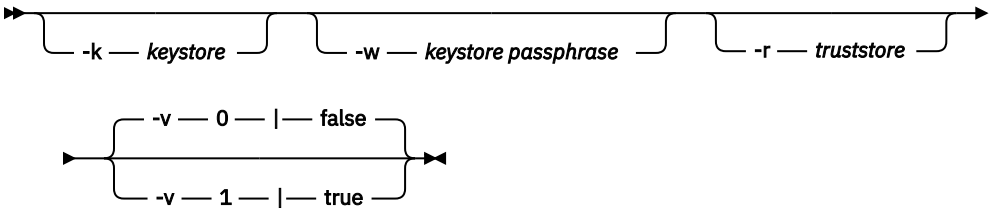
Syntax



Paho parameters



SSL parameters



Uwagi:

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

Parametry

`-h`
Wydrukuj ten tekst pomocy i zakończ

`-q`
Ustaw tryb cichy, zamiast używać domyślnego trybu false.

`-a publish|subscribe`
Ustaw działanie na wartość `publish` lub `subscribe`, a nie zakładając domyślne działanie publikowania.

-t ***nazwa tematu***
Opublikuj lub zasubskrybuj produkt *topic name*zamiast publikowania lub subskrybowania tematu domyślnego. Tematy domyślne są następujące:

Paho – przykłady

Publikowanie
`Sample/Java/v3`

Subskrybowanie
`Sample/#`

IBM WebSphere MQ – przykłady

Publikowanie
`MQTTV3Sample/Java/v3` lub `MQTTV3Sample/C/v3`

Subskrybowanie
`MQTTV3Sample/#`

-m ***tekst komunikatu***
Opublikuj *message text* zamiast wysyłania domyślnego tekstu. Domyślny tekst to "`Message from MQTTv3 C client`"lub "`Message from MQTTv3 Java client`"

-s 0|1|2
Ustaw jakość usługi (QoS) zamiast domyślnej wartości QoS 2.

-b ***nazwa hosta***
Nawiąże połączenie z adresem *host name* lub adresem IP, a nie nawiąże połączenia z domyślną nazwą hosta. Domyślną nazwą hosta dla przykładów Paho jest `m2m.eclipse.org`. Dla przykładów IBM WebSphere MQ jest to `localhost`.

-p ***numer portu***
Użyj portu *port number* , a nie portu domyślnego 1883.

Parametry produktu Paho

-i ***identyfikator klienta***
Set the client identifier to *client identifier.* The default client identifier is `SampleJavaV3_"+action`, where `action` is `publish` or `subscribe`.

-c true|false
Ustaw flagę czyszczenia sesji. Wartością domyślną jest `true`: subskrypcje nie są trwałe.

Parametry SSL

-k ***magazyn kluczy***
Ustaw ścieżkę do pliku kluczy zawierającego klucz prywatny, który identyfikuje klienta w programie *keystore*. W przypadku przykładów w języku C składnicą jest plik poczty elektronicznej o zwiększonej prywatności (Privacy-Enhanced Mail – PEM). W przypadku przykładów w języku Java jest nią magazyn kluczy Java (Java keystore – JKS).

-w ***hasło magazynu kluczy***
Ustaw hasło, aby autoryzować klienta w celu uzyskania dostępu do magazynu kluczy na serwerze *keystore passphrase*.

-r ***magazyn zaufanych certyfikatów***
Ustaw ścieżkę do magazynu kluczy zawierającego klucze publiczne serwerów produktu MQTT , które są zaufane przez klienta do produktu *truststore*. Magazyn kluczy jest plikiem PEM (Privacy-Enhanced Mail). W przypadku przykładów w języku C składnicą jest plik poczty elektronicznej o zwiększonej prywatności (Privacy-Enhanced Mail – PEM). W przypadku przykładów w języku Java jest nią magazyn kluczy Java (Java keystore – JKS).

-v 0|false|1|true
Ustaw opcję sprawdzania na wartość `1` | `true` , aby wymagała certyfikatu serwera. Wartością domyślną jest `0` | `false`: certyfikat serwera nie jest sprawdzany. Kanał SSL jest zawsze szyfrowany.

Ustaw opcję `0 | 1` dla programów w języku C oraz `true | false` dla programów Java .

Zadania pokrewne

"Pierwsze kroki z klientem MQTT dla produktu Java" na stronie 12
Pobieranie i uruchamianie z klientem produktu MQTT dla przykładowych aplikacji produktu Java przy użyciu produktu IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Przykładowe aplikacje korzystają z biblioteki klienta z pakietu SDK (Software Development Toolkit) produktu MQTT z serwisu IBM. Przykładowa aplikacja produktu `SampleAsyncCallBack` jest modelem do tworzenia aplikacji MQTT dla systemu Android i innych systemów operacyjnych sterowanych zdarzeniami.

"Pierwsze kroki z klientem MQTT dla języka C" na stronie 27
Na dowolnej platformie, na której można skompilować źródło C, należy pobrać i uruchomić przykładowy klient MQTT dla języka C. Sprawdź, czy można uruchomić przykładowy klient MQTT dla języka C z produktem IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

"Budowanie klienta MQTT dla bibliotek C" na stronie 32
Aby zbudować klienta MQTT dla bibliotek C, należy wykonać następujące kroki. Temat zawiera przełączniki kompilacji i łącza dla wielu platform, a także przykłady budowania bibliotek w systemach iOS i Windows.

Zabezpieczenia produktu MQTT

Trzy pojęcia są fundamentalne dla bezpieczeństwa produktu MQTT : tożsamość, uwierzytelnianie i autoryzacja. Tożsamość określa nazwę klienta, który jest autoryzowany i nadawany. Uwierzytelnianie polega na potwierdzeniu tożsamości klienta, a autoryzacja polega na zarządzaniu prawami, które nadawane są klientowi.

Wypróbuj przykłady zabezpieczeń

- "Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java" na stronie 58
- "Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL" na stronie 66
- "Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS" na stronie 76
- V 7.5.0.1 "Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets" na stronie 81
- "Budowanie i uruchamianie zabezpieczonego serwera MQTT client sample C app" na stronie 89

Tożsamość

Zidentyfikuj klienta MQTT według jego identyfikatora klienta, identyfikatora użytkownika lub publicznego certyfikatu cyfrowego. Jeden lub drugi z tych atrybutów definiuje tożsamość klienta. Serwer MQTT uwierzytelnia certyfikat wysłany przez klienta za pomocą protokołu SSL lub tożsamość klienta z hasłem ustawionym przez klienta. Serwer określa zasoby, do których klient może uzyskać dostęp w oparciu o tożsamość klienta.

Serwer MQTT identyfikuje się do klienta z jego adresem IP i certyfikatem cyfrowym. Klient MQTT korzysta z protokołu SSL do uwierzytelniania certyfikatu wysłanego przez serwer. W niektórych przypadkach używa nazwy DNS serwera w celu sprawdzenia, czy serwer, który go wysłał, jest zarejestrowany jako posiadacz certyfikatu.

Ustaw tożsamość klienta w jeden z następujących sposobów:

Identyfikator klienta
Klasa `MqttClient` (`MQTTClient_create` lub `MQTTAsync_create` w języku C) ustawia identyfikator klienta. Wywołaj konstruktor klasy, aby ustawić identyfikator klienta jako parametr, lub zwróć losowo wygenerowany identyfikator klienta. Identyfikator klienta musi być unikalny w obrębie wszystkich klientów nawiązujących połączenie z serwerem. Ponadto nie może on być taki sam jak nazwa menedżera kolejek na serwerze. Wszystkie klienty muszą mieć identyfikator klienta, nawet

jeśli nie jest on używany do sprawdzania tożsamości. Więcej informacji zawiera sekcja “Identyfikator klienta” na stronie 134.

ID użytkownika
Klasa `MqttClient` (`MQTTClient_create` lub `MQTTAsync_create` w C) ustawia ID użytkownika klienta jako atrybut `MqttConnectOptions` (`MqttClient_ConnectOptions` w C). Identyfikator użytkownika nie musi być unikalny dla klienta.

Certyfikat cyfrowy klienta
Certyfikat cyfrowy klienta jest przechowywany w magazynie kluczy klienta. Położenie magazynu kluczy zależy od klienta:

- **Java**

  Ustaw położenie i właściwości magazynu kluczy klienta, wywołując metodę `setSSLProperties` produktu `MqttConnectOptions` i przekazując właściwości magazynu kluczy. Więcej informacji na ten temat zawiera sekcja Modyfikacje SSL w pliku Example.java. Narzędzie **keytool** służy do zarządzania kluczami Java i magazynami kluczy.

- **C**

  Produkt `MQTTClient_create` lub `MQTTAsync_create` ustawia właściwości magazynu kluczy jako atrybuty produktu `MQTTClient_SSLOptions ssl_opts`. Narzędzie **openSSL** tworzy i zarządza kluczami i magazynami kluczy, do których dostęp jest uzyskiwany przez klienta MQTT w systemie C.

- **Android**

  Zarządzanie plikiem kluczy urządzenia Android z menu **Ustawienia** > **Zabezpieczenia** . Załaduj nowe certyfikaty z karty SD.

Ustaw tożsamość serwera, zapisując jego klucz prywatny w magazynie kluczy serwera:

IBM WebSphere MQ
Magazyn kluczy serwera MQTT jest atrybutem kanału telemetrycznego, z którym jest połączony klient.

Ustaw położenie i atrybuty magazynu kluczy za pomocą komendy IBM WebSphere MQ Explorerlub za pomocą komendy **DEFINE CHANNEL** . Patrz sekcja ZDEFINIUJ KANAŁ (MQTT). Wiele kanałów może współużytkować magazyn kluczy.

Uwierzytelnianie

Klient MQTT może uwierzytelnić serwer MQTT , z którym łączy się połączenie, a serwer może uwierzytelnić klienta, który się z nim łączy.

Klient uwierzytelnia serwer za pomocą protokołu SSL. Serwer MQTT uwierzytelnia klienta za pomocą protokołu SSL, hasła lub obu tych elementów.

Jeśli klient uwierzytelnia serwer, ale serwer nie uwierzytelnia klienta, klient jest często znany jako anonimowy klient. Powszechne jest ustanowienie anonimowego połączenia klienckiego przez SSL, a następnie uwierzytelnienie klienta przy użyciu hasła zaszyfrowanego przez sesję SSL. Bardziej powszechne jest uwierzytelnienie klienta z hasłem niż z certyfikatem klienta, ze względu na problem z dystrybucją i zarządzaniem certyfikatami. Prawdopodobnie znajdziesz certyfikaty klientów używane w urządzeniach o wysokiej wartości, takich jak bankomaty i automaty chipowe, a także w urządzeniach niestandardowych, takich jak inteligentne liczniki energii elektrycznej.

Uwierzytelnianie serwera przez klienta
Klient MQTT sprawdza, czy jest on połączony z poprawnym serwerem przez uwierzytelnienie certyfikatu serwera za pomocą protokołu SSL. Ta forma weryfikacji jest dla Ciebie zaznajomiona z przeglądaniem serwisu internetowego za pośrednictwem protokołu HTTPS.

Serwer wysyła do klienta swój certyfikat publiczny, podpisany przez ośrodek certyfikacji. Klient korzysta z klucza publicznego ośrodka certyfikacji w celu sprawdzenia podpisu ośrodka certyfikacji na certyfikacie serwera. Sprawdza również, czy certyfikat jest aktualny. Kontrole te stwierdzają, że świadectwo jest ważne.

Certyfikaty ośrodka certyfikacji, często określone jako certyfikaty główne, są przechowywane w magazynie zaufanych certyfikatów klienta:

- **Java**

  Wywołaj metodę `setSSLProperties` produktu `MqttConnectOptions` i przekaz właściwości magazynu zaufanych certyfikatów, aby ustawić położenie i właściwości magazynu zaufanych certyfikatów klienta. Więcej informacji na ten temat zawiera sekcja Modyfikacje SSL w pliku Example.java. Zarządzanie certyfikatami i magazynami zaufanych certyfikatów za pomocą narzędzia **keytool** .

- **C**

  `MQTTClient_create` lub `MQTTAsync_create` ustaw właściwości magazynu zaufanych certyfikatów jako atrybuty `MQTTClient_SSLOptions ssl_opts`. Zarządzanie certyfikatami i magazynami zaufanych certyfikatów za pomocą narzędzia **openSSL** .

- **Android**

  Zarządzaj magazyną zaufanych certyfikatów urządzenia Android z menu **Ustawienia** > **Zabezpieczenia** . Załaduj nowe certyfikaty główne z karty SD.

Uwierzytelnianie klienta przez serwer

Serwer MQTT sprawdza, czy jest on połączony z poprawnym klientem, uwierzytelnia certyfikat klienta za pomocą protokołu SSL lub uwierzytelnia tożsamość klienta za pomocą hasła.

Uwierzytelnia on klienta za pomocą hasła, które jest wysyłane przez klienta do serwera w nagłówku MQTT protocol . Serwer może uwierzytelnić identyfikator klienta, identyfikator użytkownika lub certyfikat z hasłem. Jest to zależne od serwera. Zwykle serwer uwierzytelnia identyfikator użytkownika. Sprawdź hasła za pomocą połączenia SSL, które zostało zabezpieczone przez sprawdzenie serwera, aby uniknąć wysyłania haseł w sposób jawny.

- **IBM WebSphere MQ**

  Produkt IBM WebSphere MQ uwierzytelnia certyfikat klienta za pomocą protokołu SSL. Zapisz certyfikaty główne w magazynie kluczy IBM WebSphere MQ Telemetry . Certyfikat klienta można uwierzytelnić tylko jako część wzajemnego uwierzytelniania SSL. Oznacza to, że należy udostępnić klientowi certyfikat publiczny serwera, a także udostępnić serwer w certyfikacie publicznym klienta.

  Produkt IBM WebSphere MQ Telemetry korzysta z tego samego sklepu zarówno dla własnego certyfikatu prywatnego, jak i publicznego, a także innych certyfikatów publicznych, takich jak certyfikaty główne udostępniane przez ośrodek certyfikacji.

  Ustaw położenie i atrybuty magazynu kluczy za pomocą komendy IBM WebSphere MQ Explorerlub za pomocą komendy **DEFINE CHANNEL** . Patrz sekcja ZDEFINIUJ KANAŁ (MQTT). Wiele kanałów może współużytkować magazyn kluczy.

  Produkt IBM WebSphere MQ uwierzytelnia identyfikator użytkownika klienta lub identyfikator klienta, wywołując usługę uwierzytelniania i autoryzacji produktu Java (JAAS).

  Skonfiguruj usługę JAAS w sekcji konfiguracji produktu `MQXRConfig` , która jest przechowywana w pliku `jaas.config` . Plik ten jest zapisany w katalogu `qmgrs\`*`QmgrName`*`\mqxr` w ścieżce danych IBM WebSphere MQ .

  Sprawdź autentyczność klienta, pisząc metodę `login` dla `JAASLoginModule`. Patrz sekcja “Konfiguracja kanału telemetrycznego JAAS” na stronie 120.

  Program IBM WebSphere MQ Telemetry przekazuje metodę `JAASLoginModule.login` do następujących parametrów:

  – ID użytkownika
  – Hasło
  – Identyfikator klienta
  – identyfikator sieciowy
  – Nazwa kanału

– ValidPrompts

Autoryzacja

Autoryzacja nie jest częścią MQTT protocol. Jest ona udostępniana przez serwery MQTT . To, co jest autoryzowane, zależy od tego, co robi serwer. Serwery MQTT są brokerami publikowania/subskrypcji, a przydatne reguły autoryzacji produktu MQTT kontrolują, które klienty mogą łączyć się z serwerem, a także które tematy mogą być publikowane lub subskrybowane przez klienta. Jeśli klient MQTT może administrować serwerem, więcej reguł autoryzacji sterują tym, którzy klienci mogą administrować różnymi aspektami serwera.

Liczba możliwych klientów jest ogromna, więc nie jest możliwe autoryzowanie każdego klienta osobno. Serwer MQTT będzie miał środki do grupowania klientów według profili lub grup.

Tożsamość klienta, z punktu widzenia dostępu i autoryzacji, nie jest czymś wyjątkowym dla klienta MQTT . Nie należy zrównać tożsamości klienta z identyfikatorem klienta. Mogą być takie same, ale są powszechnie różne. Na przykład prawdopodobnie istnieje nazwa użytkownika, która jest często spotykana w wielu usługach, a niektóre z tych usług współdziałają w "pojedynczym logowaniu". Serwer MQTT w skali przedsiębiorstwa może wywołać usługę autoryzacji, która oferuje wspólne tożsamości i uprawnienia dla różnych aplikacji.

IBM WebSphere MQ

Produkt IBM WebSphere MQ zawiera podłączalną usługę autoryzacji. Domyślną usługą autoryzacji udostępnianej w systemach Windows i Linux jest menedżer uprawnień do obiektów (Object Authority Manager-OAM). Więcej informacji na ten temat zawiera sekcja Kontrolowanie dostępu do obiektów za pomocą OAM w systemach UNIX, Linux i Windows. Przypisuje on identyfikatory i grupy użytkowników systemu operacyjnego z operacjami na obiektach IBM WebSphere MQ , takimi jak tematy i kolejki.

Kanał pomiarowy można skonfigurować w taki sposób, aby miał dostęp do produktu IBM WebSphere MQ przy użyciu stałego identyfikatora użytkownika. W ten sposób jest skonfigurowany przykładowy kanał. Można również uzyskać dostęp do produktu IBM WebSphere MQ z identyfikatorem użytkownika ustawionym przez klienta MQTT . Autoryzowanie klientów MQTT w celu uzyskania dostępu do obiektów produktu WebSphere MQ opisuje sposoby konfigurowania produktu IBM WebSphere MQ Telemetry w celu uzyskania grubej, średniej i ukaranej grzywną kontroli dostępu klienta.

Zadania pokrewne

“Budowanie i uruchamianie zabezpieczonego serwera MQTT client sample C app” na stronie 89
Na przykład w oparciu o przykład Windows można pobrać i uruchomić aplikację Secure sample C w dowolnym systemie operacyjnym, dla którego można skompilować źródło C. Sprawdź, czy można uruchomić przykładową aplikację C w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

“Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java” na stronie 58
W oparciu o przykład Windows można uruchomić i uruchomić aplikację Java z przykładową bezpieczną przykładową aplikacją w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java"

“Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets” na stronie 81
Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

“Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL” na stronie 66
Pobieranie i uruchamianie przykładowego klienta Android MQTT połączonego z programem IBM WebSphere MQ za pośrednictwem protokołu SSL.

“Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS” na stronie 76

Sekcja zawiera informacje na temat uwierzytelniania klienta w produkcie JAAS. Wykonaj kroki opisane w tym zadaniu, aby zmodyfikować przykładowy program `JAASLoginModule.java` i skonfigurować produkt IBM WebSphere MQ w celu uwierzytelniania aplikacji Java klienta MQTT za pomocą programu JAAS.

Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java

W oparciu o przykład Windows można uruchomić i uruchomić aplikację Java z przykładową bezpieczną przykładową aplikacją w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java"

Zanim rozpoczniesz

1. Użytkownik musi mieć dostęp do serwera MQTT version 3.1 , który obsługuje protokół MQTT protocol przez SSL.
2. Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
3. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java". Patrz Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.
4. Kanały SSL muszą być uruchomione.

O tym zadaniu

W tym artykule przedstawiono sposób kompilowania i uruchamiania zabezpieczonego produktu Przykładowa aplikacja klienta MQTT Java w systemie Windows z poziomu wiersza komend.

Należy zabezpieczyć kanał SSL za pomocą kluczy podpisanych ośrodka certyfikacji lub kluczy samopodpisanych.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 przy użyciu protokołu SSL. Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Opcjonalne: Zainstaluj pakiet Java Development Kit (JDK) w wersji 7 lub nowszej.

   7 jest wymagany do uruchomienia komendy **keytool** w celu certyfikacji certyfikatów. Jeśli nie będziesz certyfikować certyfikatów, wersja 7 pakietu JDK nie jest wymagana.
3. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu `MQTT`. Ścieżka do tego folderu jest określana w tym miejscu jako *sdkroot*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *sdkroot*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *sdkroot*\SDK.
4. Utwórz i uruchom skrypty w celu wygenerowania par kluczy i certyfikatów, a następnie skonfiguruj produkt IBM WebSphere MQ jako serwer MQTT .

Wykonaj kroki opisane w sekcji “Generowanie kluczy i certyfikatów” na stronie 100 , aby utworzyć i uruchomić skrypty. Skrypty są również wymienione w sekcji “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 60.

5. Należy sprawdzić, czy kanały SSL zostały uruchomione i czy zostały skonfigurowane zgodnie z oczekiwaniami.

   W produkcie IBM WebSphere MQ należy wpisać następującą komendę w oknie komend:

   - 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. Utwórz skrypty w celu zbudowania i uruchomienia zabezpieczonego produktu Przykładowa aplikacja klienta MQTT Java.

   a) Utwórz i uruchom program `ssjavaclient.bat` , aby przetestować kanał SSL, który jest zabezpieczony samopodpisanymi certyfikatami.

   b) Utwórz i uruchom program `cajavaclient.bat` , aby przetestować kanał SSL, który jest zabezpieczony podpisanymi certyfikatami ośrodka certyfikacji.

Skrypty służące do uruchamiania bezpiecznego klienta Java produktu MQTT

Uruchom skrypty w produkcie “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 60 przed uruchomieniem tych skryptów.

MQTT bezpieczny klient Java z samopodpisanymi certyfikatami.

Uruchom ten skrypt z samopodpisanymi certyfikatami, które zostały utworzone przez uruchomienie skryptu `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
```

Rysunek 14. `ssjavaclient.bat`

Uruchom bezpieczny klient Java MQTT z certyfikatami podpisanymi przez ośrodek certyfikacji.
Uruchom ten skrypt przy użyciu certyfikatów podpisanych przez ośrodek certyfikacji, które zostały utworzone przez uruchomienie skryptu `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
```

Rysunek 15. `cajavaclient.bat`

Pojęcia pokrewne

“Zabezpieczenia produktu MQTT” na stronie 54
Trzy pojęcia są fundamentalne dla bezpieczeństwa produktu MQTT : tożsamość, uwierzytelnianie i autoryzacja. Tożsamość określa nazwę klienta, który jest autoryzowany i nadawany. Uwierzytelnianie polega na potwierdzeniu tożsamości klienta, a autoryzacja polega na zarządzaniu prawami, które nadawane są klientowi.

Zadania pokrewne

“Generowanie kluczy i certyfikatów” na stronie 100
Wykonaj tę procedurę, aby wygenerować klucze i certyfikaty dla klientów Java i C, w tym aplikacji Android i iOS oraz serwerów IBM WebSphere MQ i IBM MessageSight .

“Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL” na stronie 66
Pobieranie i uruchamianie przykładowego klienta Android MQTT połączonego z programem IBM WebSphere MQ za pośrednictwem protokołu SSL.

“Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS” na stronie 76
Sekcja zawiera informacje na temat uwierzytelniania klienta w produkcie JAAS. Wykonaj kroki opisane w tym zadaniu, aby zmodyfikować przykładowy program `JAASLoginModule.java` i skonfigurować produkt IBM WebSphere MQ w celu uwierzytelniania aplikacji Java klienta MQTT za pomocą programu JAAS.

Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows

** **

Przykładowe pliki komend tworzą certyfikaty i bazy certyfikatów tak, jak to zostało opisane w krokach w zadaniu. Ponadto konfigurują menedżer kolejek klienta MQTT pod kątem korzystania z bazy

certyfikatów serwera. Usuwają i ponownie tworzą menedżer kolejek, wywołując skrypt `SampleMQM.bat` dostarczony z produktem IBM WebSphere MQ.

initcert.bat

Skrypt `initcert.bat` ustawia nazwy certyfikatów i innych parametrów oraz ścieżki do nich wymagane przez komendy **keytool** i **openSSL**. Ustawienia zostały opisane w komentarzach w skrypcie.

```
@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

Komendy w skrypcie `cleancert.bat` usuwają menedżer kolejek klienta MQTT, aby mieć pewność, że baza certyfikatów serwera nie będzie zablokowana, a następnie usuwają wszystkie magazyny kluczy i certyfikaty utworzone przez przykładowe skrypty zabezpieczeń.

```
@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

Komendy w skrypcie `genkeys.bat` tworzą pary kluczy dla prywatnych ośrodków certyfikacji, serwera i klienta.

```
@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

Komendy w skrypcie `sscerts.bat` eksportują samopodpisane certyfikaty klienta i serwera z ich magazynów kluczy. Następnie importują certyfikat serwera do magazynu zaufanych certyfikatów klienta, a certyfikat klienta do magazynu kluczy serwera. Serwer nie ma magazynu zaufanych certyfikatów. Komendy tworzą magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt importuje certyfikat główny ośrodka certyfikacji do prywatnych magazynów kluczy. Certyfikat główny ośrodka CA jest potrzebny do utworzenia łańcucha kluczy między certyfikatem głównym a podpisanym certyfikatem. Skrypt `cacerts.bat` eksportuje żądania certyfikatów klienta i serwera z własnych magazynów kluczy. Skrypt podpisuje żądania certyfikatów przy użyciu klucza prywatnego ośrodka certyfikacji znajdującego się w magazynie kluczy `cajkskeystore.jks`, a następnie importuje podpisane certyfikaty z powrotem do tych magazynów kluczy, z których zostały wysłane żądania. Podczas importu tworzony jest łańcuch łączący certyfikat z certyfikatem głównym ośrodka CA. Skrypt tworzy magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt generuje listę magazynów kluczy i certyfikatów w katalogu certyfikatów. Następnie tworzy przykładowy menedżer kolejek produktu MQTT i konfiguruje zabezpieczone kanały pomiarowe.

```
@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
```

Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL

Pobieranie i uruchamianie przykładowego klienta Android MQTT połączonego z programem IBM WebSphere MQ za pośrednictwem protokołu SSL.

Zanim rozpoczniesz

W tym artykule założono, że użytkownik jest uruchomiony co najmniej na poziomie interfejsu API systemu Android w wersji 14 (ICS 4.0). Wcześniejszy poziom miał magazyn kluczy, ale tylko aplikacje systemowe mogły uzyskać do niego dostęp.

1. Użytkownik musi mieć dostęp do serwera MQTT version 3.1 , który obsługuje protokół MQTT protocol przez SSL.
2. Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
3. Jeśli połączenie jest testowane na wczesnym urządzeniu Android , może być potrzebna karta SD do przesłania certyfikatu do urządzenia.
4. Jeśli testowane jest połączenie na wirtualnym urządzeniu Android , należy skonfigurować kartę SD dla urządzenia wirtualnego.
5. Kanały SSL muszą być uruchomione.

O tym zadaniu

Wykonaj to zadanie, aby uruchomić produkt Przykładowa aplikacja klienta MQTT Java dla produktu Android za pośrednictwem protokołu SSL. Pomyślne połączenie SSL ustanawia bezpieczny, szyfrowany kanał między urządzeniem Android a serwerem MQTT . Tożsamość serwera jest uwierzytelniona.

Za pomocą programu Androidmożna uwierzytelnić serwer za pomocą protokołu SSL. Można również uwierzytelnić urządzenie, mimo że aplikacja przykładowa nie obsługuje tego urządzenia. Aby uwierzytelnić urządzenie, należy użyć funkcji API KeyChainlub użyć programu JAAS do uwierzytelniania

identyfikatora klienta, adresu IP klienta lub nazwy użytkownika i hasła udostępnionego przez aplikację MQTT Android .

Wszystkie certyfikaty X.509 instalowane w magazynie zaufanych certyfikatów produktu Android muszą być podpisane przez ośrodek certyfikacji. W tym przykładzie użytkownik tworzy ośrodek certyfikacji, który podpisuje certyfikat, który jest instalowany w urządzeniu Android . Wiele certyfikatów głównych jest wstępnie zainstalowanych na urządzeniach Android .

Przed zainstalowaniem zaufanego certyfikatu należy utworzyć blokadę na urządzeniu Android . Blokada uniemożliwia osobie instalujce certyfikaty na urządzeniu bez Twojej wiedzy.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 przy użyciu protokołu SSL. Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Uruchom przykładową aplikację klienta MQTT for Android "`MQTTExerciser`" na niezabezpieczonym kanale MQTT . Patrz sekcja “Pierwsze kroki z produktem Klient MQTT dla produktu Java w systemie Android” na stronie 19.

   Aby przetestować bezpieczny kanał, należy ponownie użyć aplikacji.

   Jeśli zostało uruchomione urządzenie wirtualne Android , pozostaw je uruchomione.
3. Opcjonalne: Zainstaluj pakiet Java Development Kit (JDK) w wersji 7 lub nowszej.

   7 jest wymagany do uruchomienia komendy **`keytool`** w celu certyfikacji certyfikatów. Jeśli nie będziesz certyfikować certyfikatów, wersja 7 pakietu JDK nie jest wymagana.
4. Utwórz i uruchom skrypty w celu wygenerowania par kluczy i certyfikatów, a następnie skonfiguruj produkt IBM WebSphere MQ jako serwer MQTT .

   Wykonaj kroki opisane w sekcji “Generowanie kluczy i certyfikatów” na stronie 100 , aby utworzyć i uruchomić skrypty. Skrypty są również wymienione w sekcji “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 71.

   Potrzebny jest certyfikat publiczny ośrodka certyfikacji i magazyn kluczy serwera. Nie są wymagane certyfikaty klienta ani żadne certyfikaty w formacie `.pem` lub `.p12` .
5. Należy sprawdzić, czy kanały SSL zostały uruchomione i czy zostały skonfigurowane zgodnie z oczekiwaniami.

   W produkcie IBM WebSphere MQ należy wpisać następującą komendę w oknie komend:

   - 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. Zainstaluj certyfikat ośrodka certyfikacji w magazynie zaufanych certyfikatów Android .

   Plik ośrodka certyfikacji w tym przykładzie to `cacert.cer`.

   a) Zmień nazwę certyfikatu na `cacert.crt`

   b) Skopiuj certyfikat do głównej wewnętrznej pamięci masowej lub na kartę SD.

      W przypadku działającego urządzenia wirtualnego należy otworzyć produkt Eclipse lub uruchomić program Android Debug Bridge (ADB), aby skopiować certyfikat do urządzenia wirtualnego:

Eclipse

i) Uruchom program Eclipsei otwórz perspektywę DDMS.

ii) W widoku głównym otwórz okno **Eksplorator plików** .

iii) Otwórz katalog `mnt/sdcard` .

iv) Przeciągnij plik `cacert.crt` do katalogu `mnt/sdcard` .

ADB

i) Otwórz okno komend i ustaw jego bieżący katalog na `android-sdk\platform-tools` w katalogu instalacyjnym android, na przykład `C:\Program Files\Android\android-sdk\platform-tools`.

ii) Skopiuj certyfikat do katalogu `mnt/sdcard` :

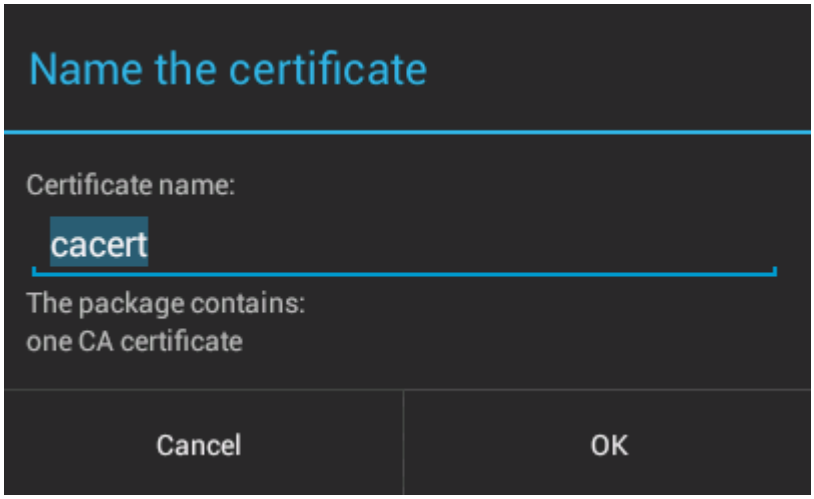
```
adb push %cacert% /mnt/sdcard/cacert.crt
```

7. Zainstaluj certyfikat w magazynie zaufanych certyfikatów na urządzeniuAndroid .

   Certyfikat musi mieć klauzulę Basic Constraints z wartością `Subject Type=CA`.

   a) Odblokuj urządzenie, a następnie kliknij przycisk **widgets** .

   b) Kliknij opcję **Ustawienia** > **Zabezpieczenia** > **Pamięć masowa informacji autoryzacyjnych** > **Zainstaluj z karty SD**.

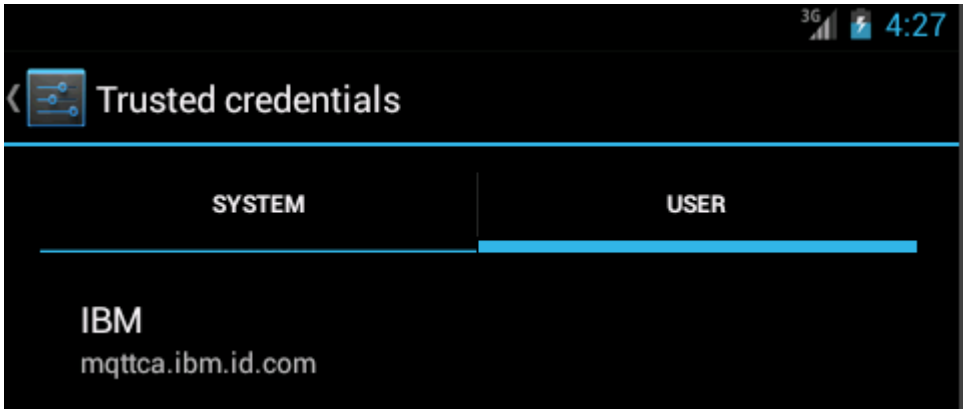


   c) Upewnij się, że nazwa pliku certyfikatu jest poprawna, a następnie kliknij przycisk **OK**.

   Uwaga: Jeśli blokada urządzenia nie została zdefiniowana, program Android zostanie poproszony o ustawienie blokady.

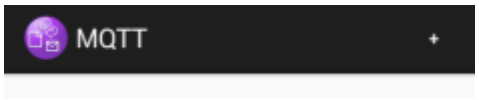
8. Potwierdź, że certyfikat jest zainstalowany na urządzeniu.

   a) Kliknij opcję **Zaufane referencje** > **Użytkownik** i poczekaj kilka minut, aby certyfikat został wyświetlony na liście certyfikatów użytkownika.

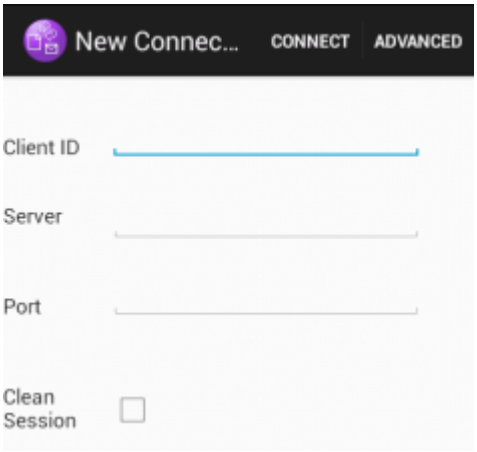


9. Ponownie uruchom aplikację `MQTTExerciser` i nawiążesz połączenie z kanałem MQTT skonfigurowanym dla anonimowych klientów SSL.
   a) Otwórz Przykładowa aplikacja klienta MQTT Java dla produktu Android.

      To okno jest otwarte w urządzeniu Android :



   b) Nawiąj połączenie z serwerem MQTT .
      i) Kliknij znak **+** , aby otworzyć nowe połączenie MQTT .



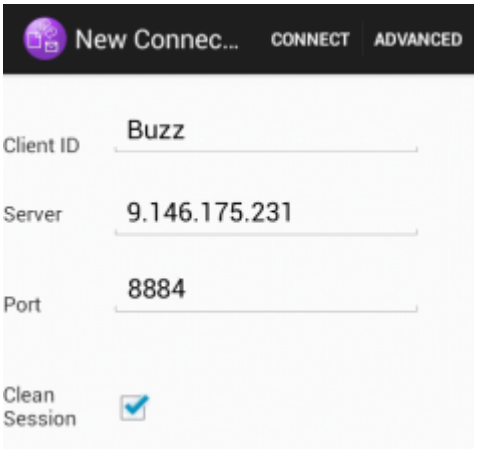
      ii) W polu **ID klienta** wprowadź dowolny unikalny identyfikator. Bądź cierpliwy, naciśnięcia klawiszy mogą być powolne.

      iii) Wpisz w polu **Serwer** adres IP serwera MQTT .

      Jest to serwer, który został wybrany w pierwszym kroku głównym. Adres IP nie może być adresem `127.0.0.1`

      iv) Wprowadź numer portu dla połączenia MQTT .

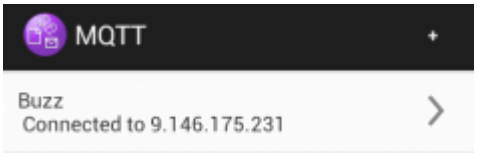
      Ustaw numer portu na 8884, który jest ustawiany przez zmienną `%sslportopt%` w przykładowych skryptach. Ten numer portu to numer portu kanału MQTT , który został skonfigurowany dla anonimowych klientów SSL, uruchamiając przykładowe skrypty w kroku “4” na stronie 67.



v) Kliknij zakładkę **Advanced** (Zaawansowane) i wybierz opcję **SSL** (SSL). Kliknij przycisk **Save** (Zapisz).

vi) Kliknij przycisk **Połącz**.

Jeśli nawiązanie połączenia powiedzie się, zostanie wyświetlony komunikat "Łączenie" , po którym następuje okno:



Wyniki

Aplikacja `MQTTExerciser` zajmuje trochę więcej czasu, aby nawiązać połączenia i wymieniać komunikaty, ale w przeciwnym razie zachowuje się inaczej, jeśli nie ma połączenia w przypadku niezabezpieczonego połączenia.

Pojęcia pokrewne

“Zabezpieczenia produktu MQTT” na stronie 54
Trzy pojęcia są fundamentalne dla bezpieczeństwa produktu MQTT : tożsamość, uwierzytelnianie i autoryzacja. Tożsamość określa nazwę klienta, który jest autoryzowany i nadawany. Uwierzytelnianie polega na potwierdzeniu tożsamości klienta, a autoryzacja polega na zarządzaniu prawami, które nadawane są klientowi.

Zadania pokrewne

“Generowanie kluczy i certyfikatów” na stronie 100
Wykonaj tę procedurę, aby wygenerować klucze i certyfikaty dla klientów Java i C, w tym aplikacji Android i iOS oraz serwerów IBM WebSphere MQ i IBM MessageSight .

“Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java” na stronie 58
W oparciu o przykład Windows można uruchomić i uruchomić aplikację Java z przykładową bezpieczną przykładową aplikacją w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java"

“Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS” na stronie 76
Sekcja zawiera informacje na temat uwierzytelniania klienta w produkcie JAAS. Wykonaj kroki opisane w tym zadaniu, aby zmodyfikować przykładowy program `JAASLoginModule.java` i skonfigurować produkt IBM WebSphere MQ w celu uwierzytelniania aplikacji Java klienta MQTT za pomocą programu JAAS.

Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows

** **

Przykładowe pliki komend tworzą certyfikaty i bazy certyfikatów tak, jak to zostało opisane w krokach w zadaniu. Ponadto konfigurują menedżer kolejek klienta MQTT pod kątem korzystania z bazy certyfikatów serwera. Usuwają i ponownie tworzą menedżer kolejek, wywołując skrypt `SampleMQM.bat` dostarczony z produktem IBM WebSphere MQ.

`initcert.bat`

Skrypt `initcert.bat` ustawia nazwy certyfikatów i innych parametrów oraz ścieżki do nich wymagane przez komendy **`keytool`** i **openSSL**. Ustawienia zostały opisane w komentarzach w skrypcie.

```
@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

Komendy w skrypcie `cleancert.bat` usuwają menedżer kolejek klienta MQTT, aby mieć pewność, że baza certyfikatów serwera nie będzie zablokowana, a następnie usuwają wszystkie magazyny kluczy i certyfikaty utworzone przez przykładowe skrypty zabezpieczeń.

```
@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

Komendy w skrypcie `genkeys.bat` tworzą pary kluczy dla prywatnych ośrodków certyfikacji, serwera i klienta.

```
@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

Komendy w skrypcie `sscerts.bat` eksportują samopodpisane certyfikaty klienta i serwera z ich magazynów kluczy. Następnie importują certyfikat serwera do magazynu zaufanych certyfikatów klienta, a certyfikat klienta do magazynu kluczy serwera. Serwer nie ma magazynu zaufanych

certyfikatów. Komendy tworzą magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt importuje certyfikat główny ośrodka certyfikacji do prywatnych magazynów kluczy. Certyfikat główny ośrodka CA jest potrzebny do utworzenia łańcucha kluczy między certyfikatem głównym a podpisanym certyfikatem. Skrypt `cacerts.bat` eksportuje żądania certyfikatów klienta i serwera z własnych magazynów kluczy. Skrypt podpisuje żądania certyfikatów przy użyciu klucza prywatnego ośrodka certyfikacji znajdującego się w magazynie kluczy `cajkskeystore.jks`, a następnie importuje podpisane certyfikaty z powrotem do tych magazynów kluczy, z których zostały wysłane żądania. Podczas importu tworzony jest łańcuch łączący certyfikat z certyfikatem głównym ośrodka CA. Skrypt tworzy magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt generuje listę magazynów kluczy i certyfikatów w katalogu certyfikatów. Następnie tworzy przykładowy menedżer kolejek produktu MQTT i konfiguruje zabezpieczone kanały pomiarowe.

```
@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
```

Uwierzytelnianie aplikacji Java klienta MQTT za pomocą programu JAAS

Sekcja zawiera informacje na temat uwierzytelniania klienta w produkcie JAAS. Wykonaj kroki opisane w tym zadaniu, aby zmodyfikować przykładowy program `JAASLoginModule.java` i skonfigurować produkt IBM WebSphere MQ w celu uwierzytelniania aplikacji Java klienta MQTT za pomocą programu JAAS.

Zanim rozpoczniesz

1. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java". Patrz Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack.
2. Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
3. You must have access to the `MQXR JAASLoginModule` and `JAASPrincipal` Java samples in an IBM WebSphere MQ installation. Przykłady znajdują się w ścieżce `%MQ_FILE_PATH%\mqxr\samples.`
4. Wykonaj kroki opisane w sekcji Windows lub Linux, a przykłady są pobierane z programu Windows.
5. Aby wykonać krok “1” na stronie 77, użytkownik musi mieć uprawnienia do tworzenia menedżera kolejek produktu `MQXR_SAMPLE_QM` w systemie IBM WebSphere MQ.

O tym zadaniu

W zadaniu wyprowadzane są parametry identyfikacji klienta MQTT `Sample` z wersji produktu `JAASLoginModule`. Zapisanie parametrów klienta pociąga za sobą modyfikację przykładowego programu `JAASLoginModule` i skonfigurowanie produktu IBM WebSphere MQ w celu załadowania używanej wersji produktu `JAASLoginModule`.

Procedura

1. Complete the steps in “Compile and run all the MQTT client sample Java apps from Eclipse” na stronie 16 to run the Paho MQTT `Sample` client.

   Celem jest przygotowanie środowiska programistycznego w celu opracowania i przetestowania uwierzytelniania produktu JAAS . Aby dostosować moduł uwierzytelniania produktu JAAS , wymagane jest środowisko programistyczne produktu Java . W tym przykładzie uruchamiany jest przykładowy klient Paho dla produktu Java w celu przetestowania konfiguracji produktu JAAS . W celu uproszczenia należy użyć tego samego środowiska programistycznego w celu zmodyfikowania zarówno przykładowego klienta, jak i przykładowego modułu logowania do produktu JAAS . Alternatywnie można przetestować moduł logowania produktu JAAS przy użyciu klienta MQTT dla języka C lub dowolnego innego klienta MQTT .

2. Opcjonalne: Dodaj nazwę użytkownika i parametr hasła do przykładu MQTT Paho .

   Uwaga: Jeśli klient Paho dla produktu Java zawiera parametry nazwy użytkownika i hasła, ten krok jest niepotrzebny. Zapoznaj się z serwisem pobierania, aby uzyskać aktualizację. Więcej informacji na ten temat zawiera sekcja Pobieranie społeczności przesyłania komunikatówIBM. W przeciwnym razie należy zmienić kopię produktu `Sample.java`.

   a) Otwórz eksplorator pakietów w pakiecie `org.eclipse.paho.sample.mqttv3app` w projekcie przykładów produktu Paho .

   b) Kliknij prawym przyciskiem myszy opcję `Sample.java` **Kopiuj** > **Wklej**. W oknie **Konflikt nazw** wpisz nazwę `SampleForJAAS`.

   c) Dodaj następujące wiersze kodu do metody `main` .

      i) Po wierszu "`boolean ssl = false;`"zadeklaruj zmienne `userName` i `password` :

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

      W celu zapewnienia zgodności ze starszymi serwerami MQTT domyślnie nie są ustawiane parametry hasła i nazwy użytkownika.

      ii) Po wierszu "`case 'v': ssl = Boolean.valueOf(args[++i]).booleanValue(); break;`"przeanalizuj dwa nowe parametry wejściowe:

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

      iii) Przed wierszem, "`if (action.equals("publish")) {`", dodaj `userName` i `password` do argumentów konstruktora `Sample` :

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

   d) Dodaj `userName` i `password` do konstruktora `Sample`.

      Zmień element:

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

      to:

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

e) Dodaj następujące wiersze kodu do konstruktora `Sample` .

Po wierszu "`conOpt.setCleanSession(clean);`"ustaw zmienne `userName` i `password` w obiekcie `conOpt` w metodzie `Sample` :

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

f) W metodach `publish` i `subscribe` zmodyfikuj następujące wiersze kodu:

Zmień wiersz "`client.connect();`" na

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

3. Utwórz projekt Java (`JAASSample`) na przykład dla produktu JAAS .
   a) W obszarze roboczym produktu Eclipse otwórz kreator **Nowy projekt Java** : kliknij opcję **Plik** > **Nowy** > **Projekt Java**.
   b) W polu **Nazwa projektu** wpisz `JAASSample`.
   c) W opcjach środowiska JRE wybierz opcję `J2SE-1.5` jako środowisko wykonawcze JRE, a następnie kliknij przycisk **Dalej**.

   Środowisko JRE musi być zgodne ze środowiskiem JRE, na którym działa serwer IBM WebSphere MQ . Program IBM WebSphere MQ Version 7.5 uruchamia produkt `J2SE-1.5`.
   d) W oknie **Java Settings** (Ustawienia Java) kliknij kartę **Libraries** (Biblioteki) i kliknij opcję **Add External JARS ...**(Dodaj zewnętrzne pliki JARS). Przejdź do`%MQ_FILE_PATH%\mqxr\lib` i wybierz plik **MQXR.jar**, a następnie kliknij przycisk **Zakończ**.
4. Import the JAAS samples `JAASLoginModule` and `JAASPrincipal` classes.
   a) Kliknij prawym przyciskiem myszy projekt `JAASSample` w eksploratorze pakietów **Importuj ...** > **Ogólne** > **System plików** i kliknij przycisk **Dalej**.
   b) Przejdź do opcji `%MQ_FILE_PATH%\mqxr\samples` i zaznacz opcje `JAASLoginModule.java` i `JAASPrincipal.java`, a następnie kliknij przycisk **Zakończ**.
   c) Zaznacz i kliknij prawym przyciskiem myszy oba pliki Java w eksploratorze pakietów, **Refaktoryzuj ...** > **Przenieś**.
   d) W oknie **Przenieś** sprawdź, czy produkt `JAASSample` jest wybrany jako miejsce docelowe dla obu elementów, a następnie kliknij opcję **Utwórz pakiet ...** .
   e) Wpisz `samples` w polu **Nazwa** w kreatorze **Nowy pakiet Java** , a następnie kliknij przycisk **Zakończ** > **OK** .

   Środowisko Eclipse buduje zaimportowane klasy produktu Java z liczbą ostrzeżeń dotyczących nieużywanych wartości.
5. Zmień nazwę klasy `JAASLoginModule`

   Zmień nazwę klasy, tak aby łatwiej było ją odróżnić od przykładowej klasy `JAASLoginModule` dostarczanej razem z produktem IBM WebSphere MQ.
   a) Kliknij prawym przyciskiem myszy `JAASloginModule.java` w eksploratorze pakietów **Refaktoryzuj ...** > **Zmień nazwę**.
   b) W oknie **Rename Compilation Unit** (Zmiana nazwy jednostki kompilacji) zmień wartość w polu **New name** (Nowa nazwa) z `JAASloginModule` na `MyJAASloginModule`; kliknij przycisk **Finish**(Zakończ).
6. Zmodyfikuj klasę `MyJAASloginModule` w taki sposób, aby wyprowadzała treść pól wywołania zwrotnego.

a) Dodaj następujący wiersz kodu do produktu `MyJAASloginModule.java`.

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

Umieść linie tuż przed instrukcją:`"if (true) loggedIn = true;"`

b) Naciśnij klawisz `CTRL+Shift+O` , aby zreorganizować importy, a następnie zapisz plik.

7. Zmień nazwę `JAASPrincipal` na `MyJAASPrincipal`.

Zmień nazwę klasy, aby uniknąć nieporozumień z przykładową klasą `JAASPrincipal` . W tym przykładzie pozostaw zawartość klasy `MyJAASPrincipal` w niezmienionym stanie.

8. Podaj ID użytkownika, który uruchomił procesy menedżera kolejek `read` i `execute` , a także uprawnienia do klas JAAS .

a) W programie Windows Explorer otwórz katalog obszaru roboczego produktu Eclipse . W tym przykładzie położenie obszaru roboczego Eclipse jest reprezentowane przez zmienną Eclipse , *`workspace_loc`*.

b) Przejdź do katalogu, w którym znajdują się klasy `MyJAASLoginModule` i `MyJAASPrincipal` .

Ścieżka do katalogu to *`workspace_loc`*`\JAASSample\bin\samples`

c) Wybierz, a następnie kliknij prawym przyciskiem myszy oba te klasy, a następnie kliknij opcję **Właściwości**. Kliknij kartę **Zabezpieczenia** w oknie **Właściwości** .

d) Kliknij przycisk **Dodaj ...**, wpisz nazwę obiektu `mqm`, a następnie kliknij opcję **Sprawdź nazwy** , aby go zweryfikować; kliknij przycisk **OK**.

e) Wybierz pozycję **mqm** z listy **Nazwy grup lub użytkowników** , a następnie zaznacz pole **read and execute** (odczyt i wykonanie) oraz **read** (odczyt) na liście uprawnień dla produktu `mqm`. Kliknij przycisk OK.

9. Skonfiguruj produkt IBM WebSphere MQ do uruchamiania klasy `MyJAASLoginModule` .

a) Dodaj plik `service.env` do konfiguracji IBM WebSphere MQ , aby zdefiniować ścieżki klas w celu załadowania klasy `MyJAASLoginModule` .

Utwórz plik `service.env` z następującą instrukcją ścieżki klasy w katalogu *`WMQ_DATA_PATH`* :

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

Gdzie *`user.dir`* to katalog główny katalogu dla plików klas, które są kompilowane w obszarze roboczym produktuEclipse . Katalog *`WMQ_DATA_PATH`* zawiera katalog `qmgrs` . Patrz Dodatkowe zmienne środowiskowe.

Wskazówka: `CLASSPATH=`*`user.dir`*`\JAASSample\bin` może być jedyną instrukcją w pliku `service.env` .

b) Dodaj sekcję `MyJAASStanza`do pliku `jaas.config` , aby zidentyfikować klasę `MyJAASLoginModule` w odniesieniu do ścieżek klas w pliku `service.env` .

`jaas.config` znajduje się w katalogu `mqxr` menedżera kolejek; *`WMQ_DATA_PATH`*`\Qmgrs\MQXR_SAMPLE_QM\mqxr`.

Sekcja jest następująca:

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

c) Skonfiguruj kanał produktu IBM WebSphere MQ Telemetry przy użyciu nazwy sekcji konfiguracji produktu JAAS .

Uruchom następującą komendę w oknie komend:

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

Komenda powiąże kanał MyJAAS z MyJAASStanza w pliku jaas.config . Przez niepodanie opcji MCAUSER lub podanie opcji USECLTID w definicji kanału, kanał autoryzuje dostęp do zasobów menedżera kolejek z nazwą użytkownika dostarczonym przez program kliencki MQTT . W tym przykładzie nazwa użytkownika, która jest dostarczana przez klienta, jest ustawiona na wartość "Guest". W tym przykładzie używane są istniejące autoryzacje dla Guest ustawione przez plik komend SampleMQM .

10. Zrestartuj usługę IBM WebSphere MQ Telemetry , aby zapoznać się z nowymi danymi konfiguracyjnymi.
    Aby zrestartować usługę IBM WebSphere MQ Telemetry , uruchom menedżer kolejek lub usługę z produktu IBM WebSphere MQ Explorerlub uruchom następujące komendy dla przykładowej konfiguracji:

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

11. Uruchom program Sample .

    Aby skonfigurować uruchamianie konfiguracji dla produktu SampleForJAAS , należy wykonać tę samą procedurę, co w kroku “1” na stronie 77, z następującymi modyfikacjami:

    a) Ustaw numer portu na 1890, aby był zgodny z konfiguracją kanału MQTT .

    b) Dodaj parametry -u Guest -z password do haseł na karcie **(x) = Argumenty** dla konfiguracji Subskrybenta i publikatora utworzonego dla programu Sample .

    Programy przykładowe są uruchamiane bez żadnych zmian w danych wyjściowych, z wyjątkiem numeru portu, który jest teraz 1890 , a nie w wersji 1883.

    W katalogu *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM otwórz plik mqxr.stdout . Dane wyjściowe produktu MyJAASLoginModule są zapisywane w produkcie mqxr.stdout:

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

Co dalej

Jeśli przykład nie działa, należy zapoznać się z tematem rozwiązywania problemów dotyczących JAAS; “Rozwiązywanie problemu: moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną” na stronie 193, a następnie spróbować tych wskazówek.

1. Dodaj parametr -verbose do parametrów w produkcie *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\java.properties. Z poziomu tego dziennika można sprawdzić, czy klasa została załadowana pomyślnie.

   Dane wyjściowe są zapisywane w produkcie *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr.stderr.

2. W produkcie *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\mqxr.log należy szukać wyjątków, które są zgłaszane w produkcie MyJAASLoginModule. Jeśli na przykład podejmowana jest próba wyjścia passwordo wartości NULL, która jest tablicą znakową, a nie łańcuchem, zostanie zgłoszony wyjątek.

3. Patrz *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\errors\AMQERR01.log. Jeśli nazwa użytkownika w produkcie userName nie jest autoryzowana do uzyskiwania dostępu do zasobów menedżera kolejek, a kanał jest skonfigurowany bez opcji MCAUSER lub USECLTID , to w tym miejscu są zgłaszane wszystkie błędy.

4. Sprawdź, czy nazwa sekcji w programie *WMQ_DATA_PATH*\Qmgrs\MQXR_SAMPLE_QM\mqxr\jaas.config jest taka sama, jak nazwa w kanale MQTT , który jest skonfigurowany dla portu, z którym klient Sample próbuje nawiązać połączenie.

5. Sprawdź, czy ścieżka w sekcji jest zgodna z ścieżką do klasy `MyJAASLoginModule` w produkcie Eclipse, na przykład:

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

6. Aby wyeliminować, czy błąd znajduje się w ścieżce klasy w pliku `service.env` w programie *WMQ_DATA_PATH* , nie jest poprawnie pobierany, należy zmienić wiersz "`set CLASSPATH=%MQXRCLASSPATH%;%CLASSPATH%`" w programie `%MQ_FILE_PATH%\mqxr\bin\controlMQXR.BAT` w taki sposób, aby zawierał ścieżkę klasy. Można również powtórzyć ścieżkę klasy. Jednak ścieżka klasy nie zawiera ścieżki klasy, która jest ustawiona w programie `service.env`, więc ta ścieżka działa tylko wtedy, gdy zostanie zmodyfikowany plik `controlMQXR.BAT` .

Pojęcia pokrewne

“Zabezpieczenia produktu MQTT” na stronie 54
Trzy pojęcia są fundamentalne dla bezpieczeństwa produktu MQTT : tożsamość, uwierzytelnianie i autoryzacja. Tożsamość określa nazwę klienta, który jest autoryzowany i nadawany. Uwierzytelnianie polega na potwierdzeniu tożsamości klienta, a autoryzacja polega na zarządzaniu prawami, które nadawane są klientowi.

“Konfiguracja kanału telemetrycznego JAAS” na stronie 120
Skonfiguruj usługę JAAS w taki sposób, aby uwierzytelniała `Nazwę użytkownika` wysłanego przez klienta.

Zadania pokrewne

“Rozwiązywanie problemu: moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną” na stronie 193
Sprawdź, czy moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną (MQXR), a następnie skonfiguruj usługę JAAS w celu usunięcia problemu.

“Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java” na stronie 58
W oparciu o przykład Windows można uruchomić i uruchomić aplikację Java z przykładową bezpieczną przykładową aplikacją w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java"

“Łączenie przykładowej aplikacji Java klienta MQTT w systemie Android przy użyciu protokołu SSL” na stronie 66
Pobieranie i uruchamianie przykładowego klienta Android MQTT połączonego z programem IBM WebSphere MQ za pośrednictwem protokołu SSL.

Informacje pokrewne

Dodatkowe zmienne środowiskowe

Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets

Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

Zanim rozpoczniesz

1. Użytkownik musi mieć dostęp do serwera MQTT version 3 , który obsługuje MQTT protocol za pośrednictwem serwera WebSockets.
2. Przeglądarka musi obsługiwać protokół SSL i WebSocket protocol. Patrz sekcja “Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL” na stronie 181.

3. Kanały SSL muszą być uruchomione.

O tym zadaniu

Wykonaj to zadanie, aby uruchomić klienta przesyłania komunikatów produktu MQTT dla przykładowych stron produktu JavaScript za pośrednictwem protokołu SSL. Zadanie to kieruje użytkownika do programu “Generowanie kluczy i certyfikatów” na stronie 100 w celu utworzenia certyfikatów i skonfigurowania produktu IBM WebSphere MQ.

Należy zabezpieczyć kanał SSL za pomocą kluczy podpisanych ośrodka certyfikacji lub kluczy samopodpisanych.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać MQTT protocol nad zabezpieczonym serwerem WebSockets.

   - IBM MessageSighti wersje produktu IBM WebSphere MQ w wersji 7.5.0.1 i nowszych, wykonaj następujące czynności.
2. Opcjonalne: Należy zainstalować pakiet Java Development Kit (JDK) w wersji 7 lub nowszej.

   If you are setting up a test system, and you want to use self-signed certificates, you need to use the JDK Version 7 **keytool** command to certify your certificates. W przypadku konfigurowania systemu produkcyjnego i wysyłania żądań podpisania certyfikatu (CSR) do zewnętrznego ośrodka certyfikacji, pakiet JDK 7 nie jest wymagany.
3. Utwórz i uruchom skrypty w celu wygenerowania par kluczy i certyfikatów, a następnie skonfiguruj produkt IBM WebSphere MQ jako serwer MQTT .

   Wykonaj kroki opisane w sekcji “Generowanie kluczy i certyfikatów” na stronie 100 , aby utworzyć i uruchomić skrypty. Skrypty są również wymienione w sekcji “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 84.

   Nazwa zwykła certyfikatu serwera musi być zgodna z nazwą DNS kanału serwera. Niektóre przeglądarki akceptują certyfikaty, które zawierają listę nazw wspólnych, na przykład:

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

   Inne przeglądarki akceptują tylko jedną wspólną nazwę. Na przykład, Firefox, do wersji 18, akceptuje tylko jedną nazwę zwykłą. Późniejsze wersje mogą być różne.
4. Należy sprawdzić, czy kanały SSL zostały uruchomione i czy zostały skonfigurowane zgodnie z oczekiwaniami.

   W produkcie IBM WebSphere MQ należy wpisać następującą komendę w oknie komend:

   - 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. Zainstaluj certyfikaty w bazie certyfikatów przeglądarki.

   W tym przykładzie należy wybrać jeden z następujących certyfikatów serwera:

   a. W przypadku samopodpisanego certyfikatu serwera certyfikat to `srvcertselfsigned.cer`.

   b. W przypadku certyfikatu serwera podpisanego przez prywatny ośrodek certyfikacji certyfikat to `cacert.cer`.

c. W przypadku certyfikatu serwera podpisanego przez zewnętrzny ośrodek certyfikacji, sprawdź, czy certyfikat główny ośrodka certyfikacji jest już zainstalowany w bazie certyfikatów.

W zależności od wsparcia dostępnego w przeglądarce, zainstaluj produkt `cacert.cer` na liście zaufanych ośrodków certyfikacji (Trusted Root Certification Authorities). Patrz sekcja “Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL” na stronie 181.

6. Opcjonalne: Uwierzytelnij klienta.

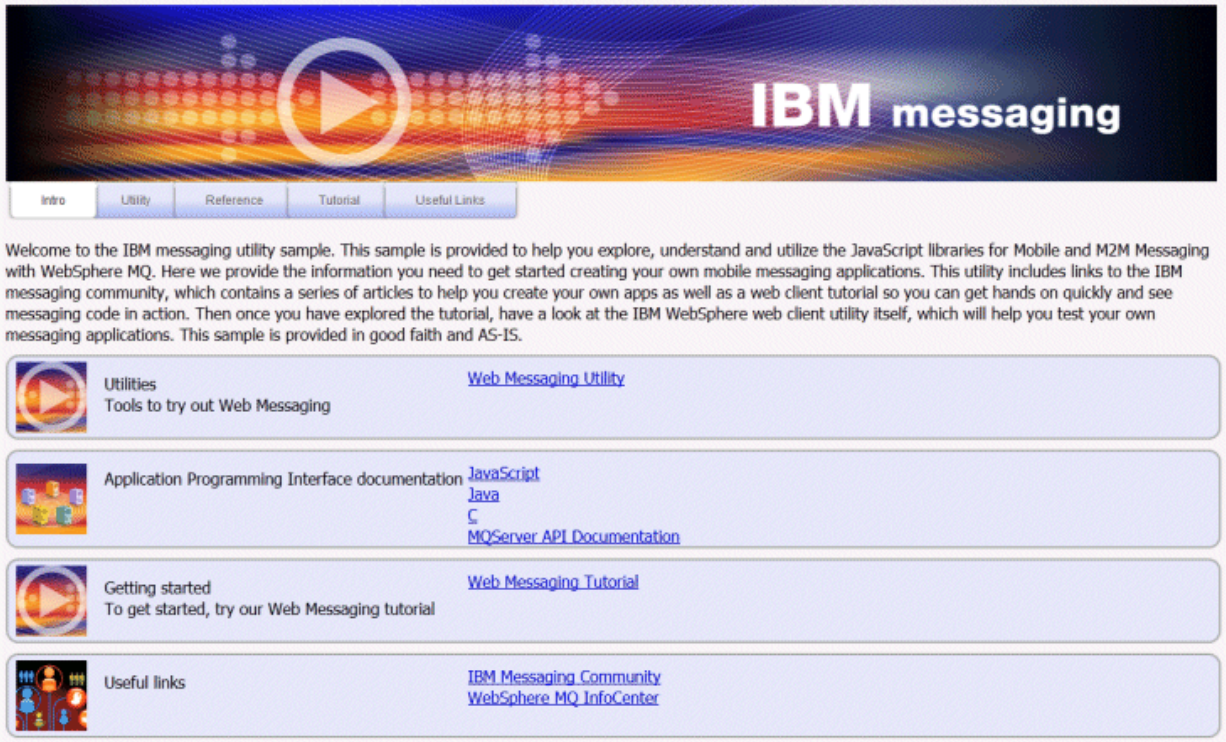
   W przykładzie instalowany jest produkt `cacert.cer` w celu uwierzytelnienia serwera i szyfrowania kanału, ale nie do uwierzytelniania klienta. Aby uwierzytelnić klienta, należy zainstalować magazyn kluczy klienta `cltkeystore.p12w` przeglądarce. Nie wszystkie przeglądarki obsługują uwierzytelnianie klientów. Patrz sekcja “Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL” na stronie 181.

7. Połącz się z bezpiecznym kanałem WebSockets .

   Otwórz przeglądarkę i wpisz adres URL kanału WebSockets na pasku adresu; w tym przykładzie:

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

   IBM WebSphere MQ odpowiada pierwszej stronie Przykładowe strony JavaScript klienta przesyłania komunikatów MQTT.



   Jeśli nawiązanie połączenia nie powiedzie się, a użytkownik uruchomił przykładowe skrypty w celu skonfigurowania przykładowego menedżera kolejek produktu MQTT , spróbuj połączyć się z normalnym kanałem WebSockets na porcie 1886. Sukces na porcie 1886 izoluje awarię połączenia SSL.

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

Pojęcia pokrewne

“Klient przesyłania komunikatów MQTT dla produktu JavaScript i aplikacje WWW” na stronie 122
“Jak programować aplikacje przesyłania komunikatów w produkcie JavaScript” na stronie 126

Zadania pokrewne

“Generowanie kluczy i certyfikatów” na stronie 100

Wykonaj tę procedurę, aby wygenerować klucze i certyfikaty dla klientów Java i C, w tym aplikacji Android i iOS oraz serwerów IBM WebSphere MQ i IBM MessageSight .

“Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript” na stronie 25
Można rozpocząć od Klient przesyłania komunikatów MQTT dla produktu JavaScript , wyświetlając przykładową stronę główną klienta przesyłania komunikatów i przeglądając zasoby, do których odsyłacze te są dostępne. Aby wyświetlić tę stronę główną, należy skonfigurować serwer MQTT w taki sposób, aby akceptować połączenia z serwera Przykładowe strony JavaScript klienta przesyłania komunikatów MQTT, a następnie należy wpisać adres URL skonfigurowany na serwerze w przeglądarce WWW. Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zostanie uruchomiony automatycznie na urządzeniu, a zostanie wyświetlona przykładowa strona główna klienta przesyłania komunikatów. Ta strona zawiera odsyłacze do programów narzędziowych, dokumentacji interfejsu programistycznego, kursu i innych przydatnych informacji.

Odsyłacze pokrewne

“Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL” na stronie 181
Istnieją różnice między różnymi przeglądarkami, na różnych platformach. Zapoznanie się z tymi różnicami ułatwia konfigurowanie aplikacji, ośrodków certyfikacji (CA) i certyfikatów klienta w celu nawiązania połączenia przy użyciu protokołu Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i produktu WebSockets.

Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows

** **

Przykładowe pliki komend tworzą certyfikaty i bazy certyfikatów tak, jak to zostało opisane w krokach w zadaniu. Ponadto konfigurują menedżer kolejek klienta MQTT pod kątem korzystania z bazy certyfikatów serwera. Usuwają i ponownie tworzą menedżer kolejek, wywołując skrypt `SampleMQM.bat` dostarczony z produktem IBM WebSphere MQ.

`initcert.bat`

Skrypt `initcert.bat` ustawia nazwy certyfikatów i innych parametrów oraz ścieżki do nich wymagane przez komendy **keytool** i **openSSL**. Ustawienia zostały opisane w komentarzach w skrypcie.

```
@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

Komendy w skrypcie `cleancert.bat` usuwają menedżer kolejek klienta MQTT, aby mieć pewność, że baza certyfikatów serwera nie będzie zablokowana, a następnie usuwają wszystkie magazyny kluczy i certyfikaty utworzone przez przykładowe skrypty zabezpieczeń.

```
@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

Komendy w skrypcie `genkeys.bat` tworzą pary kluczy dla prywatnych ośrodków certyfikacji, serwera i klienta.

```
@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

Komendy w skrypcie `sscerts.bat` eksportują samopodpisane certyfikaty klienta i serwera z ich magazynów kluczy. Następnie importują certyfikat serwera do magazynu zaufanych certyfikatów klienta, a certyfikat klienta do magazynu kluczy serwera. Serwer nie ma magazynu zaufanych certyfikatów. Komendy tworzą magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt importuje certyfikat główny ośrodka certyfikacji do prywatnych magazynów kluczy. Certyfikat główny ośrodka CA jest potrzebny do utworzenia łańcucha kluczy między certyfikatem głównym a podpisanym certyfikatem. Skrypt `cacerts.bat` eksportuje żądania certyfikatów klienta i serwera z własnych magazynów kluczy. Skrypt podpisuje żądania certyfikatów przy użyciu klucza prywatnego ośrodka certyfikacji znajdującego się w magazynie kluczy `cajkskeystore.jks`, a następnie importuje podpisane certyfikaty z powrotem do tych magazynów kluczy, z których zostały wysłane żądania. Podczas importu tworzony jest łańcuch łączący certyfikat z certyfikatem głównym ośrodka CA. Skrypt tworzy magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt generuje listę magazynów kluczy i certyfikatów w katalogu certyfikatów. Następnie tworzy przykładowy menedżer kolejek produktu MQTT i konfiguruje zabezpieczone kanały pomiarowe.

```
@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
```

Budowanie i uruchamianie zabezpieczonego serwera MQTT client sample C app

Na przykład w oparciu o przykład Windows można pobrać i uruchomić aplikację Secure sample C w dowolnym systemie operacyjnym, dla którego można skompilować źródło C. Sprawdź, czy można uruchomić przykładową aplikację C w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

Zanim rozpoczniesz

1. Użytkownik musi mieć dostęp do serwera MQTT version 3.1 , który obsługuje protokół MQTT protocol przez SSL.
2. Jeśli między klientem a serwerem znajduje się zapora firewall, należy sprawdzić, czy nie blokuje on ruchu MQTT .
3. Wersje binarne bibliotek klienta dla języka C są udostępniane dla wielu systemów operacyjnych. W przypadku niektórych z tych systemów operacyjnych bezpieczną wersję klienta nie są udostępniane jako plik binarny. W przypadku tych systemów operacyjnych należy postępować zgodnie z instrukcjami w sekcji “Budowanie klienta MQTT dla bibliotek C” na stronie 32.
4. W celu rozwiązania problemu dział wsparcia IBM może wymagać uruchomienia klienta MQTT dla języka C na platformie referencyjnej.
5. Kanały SSL muszą być uruchomione.

Przegląd obsługiwanych platform i platform referencyjnych zawiera sekcja Wymagania systemowe produktu IBM Mobile Messaging and M2M Client Pack. Szczegółowe informacje na temat obsługi klienta C można znaleźć w odpowiednich sekcjach Wymagania systemowe produktu WebSphere MQ V7.5 Telemetry.

O tym zadaniu

W tym artykule przedstawiono sposób kompilowania i uruchamiania zabezpieczonego produktu MQTT client sample C app w systemie Windows z poziomu wiersza komend. Na ilustracji produkt Microsoft Visual Studio 2010 jest używany do kompilowania klienta. Użytkownik może zmodyfikować skrypty wiersza komend, aby skompilować i uruchomić przykładową aplikację w innych systemach operacyjnych, takich jak Linux i iOS.

Uwaga:

W skryptach produktu Windows udostępnionych w tym artykule założono, że cały pakiet OpenSSL jest kompilowany ze źródła. Jeśli zostanie wybrana opcja użycia wstępnie skompilowanych bibliotek udostępnianych przez IBM , można również uzyskać prekompilowaną wersję binarną OpenSSL. Wstępnie skompilowane biblioteki nie są dostępne do użycia z produktem iOS.

Należy zabezpieczyć kanał SSL za pomocą kluczy podpisanych ośrodka certyfikacji lub kluczy samopodpisanych.

Procedura

1. Wybierz serwer MQTT , z którym można nawiązać połączenie z aplikacją kliencką.

   Serwer musi obsługiwać protokół MQTT version 3.1 przy użyciu protokołu SSL. Wszystkie serwery MQTT z IBM do tego, w tym IBM WebSphere MQ i IBM MessageSight. Patrz “Pierwsze kroki z serwerami MQTT” na stronie 143.
2. Opcjonalne: Zainstaluj pakiet Java Development Kit (JDK) w wersji 7 lub nowszej.

   7 jest wymagany do uruchomienia komendy **keytool** w celu certyfikacji certyfikatów. Jeśli nie będziesz certyfikować certyfikatów, wersja 7 pakietu JDK nie jest wymagana.
3. Należy zainstalować środowisko programistyczne języka C na platformie, na której przeprowadzane są operacje budowania.

   Pliki makefile w przykładach w tym temacie mają następujące narzędzia:

   - iOS W przypadku produktu iOSw systemie Apple Mac z systemem operacyjnym OS X 10.8.2 z narzędziami programistycznym iOS z poziomu Xcode.
   - Linux W przypadku produktu Linuxw wersji gcc 4.4.6 z produktu Red Hat Enterprise Linux w wersji 6.2.

     Minimalny obsługiwany poziom biblioteki `glibc` C to 2.12, a jądro Linux to 2.6.32.

- Windows W przypadku produktu Microsoft Windowsw wersji Visual Studio 10.0.

4. Pobierz Mobile Messaging and M2M Client Pack i zainstaluj pakiet SDK produktu MQTT .

   Nie ma programu instalacyjnego. Wystarczy rozpakować pobrany plik.

   a. Pobierz Mobile Messaging and M2M Client Pack.

   b. Należy utworzyć folder, w którym ma zostać zainstalowany pakiet SDK.

      Może zaistnieć potrzeba nazwania folderu MQTT. Ścieżka do tego folderu jest określana w tym miejscu jako *sdkroot*.

   c. Rozwiń skompresowaną zawartość pliku Mobile Messaging and M2M Client Pack do programu *sdkroot*. Rozszerzenie tworzy drzewo katalogów, które rozpoczyna się od *sdkroot*\SDK.

5. Opcjonalne: Wykonaj kroki opisane w sekcji “Budowanie klienta MQTT dla bibliotek C” na stronie 32.

   Ten krok należy wykonać tylko wtedy, gdy pakiet SDK produktu MQTT nie zawiera bezpiecznej biblioteki klienta C dla docelowego systemu operacyjnego.

   - Windows Biblioteki są `mqttv3cs.lib` do kompilowania, a `mqttv3cs.dll` do działania.
   - Linux Biblioteka: `libmqttv3cs.so`
   - iOS Biblioteka: `libmqttv3cs.a`

6. Utwórz i uruchom skrypty w celu wygenerowania par kluczy i certyfikatów, a następnie skonfiguruj produkt IBM WebSphere MQ jako serwer MQTT .

   Wykonaj kroki opisane w sekcji “Generowanie kluczy i certyfikatów” na stronie 100 , aby utworzyć i uruchomić skrypty. Skrypty są również wymienione w sekcji “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 94.

7. Należy sprawdzić, czy kanały SSL zostały uruchomione i czy zostały skonfigurowane zgodnie z oczekiwaniami.

   W produkcie IBM WebSphere MQ należy wpisać następującą komendę w oknie komend:

   - 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. Utwórz skrypty w celu zbudowania i uruchomienia zabezpieczonego produktu MQTT client sample C app.

   a) Utwórz i uruchom program `sscclient.bat` , aby przetestować kanał SSL, który jest zabezpieczony samopodpisanymi certyfikatami.

   b) Utwórz i uruchom program `cacclient.bat` , aby przetestować kanał SSL, który jest zabezpieczony podpisanymi certyfikatami ośrodka certyfikacji.

Wyniki

Wyniki są podobne do uruchomienia niezabezpieczonego klienta.

```
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
```

Rysunek 16. Bezpieczny subskrybent

```
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 . . .
```

Rysunek 17. Bezpieczny publikator

Skrypty do uruchamiania zabezpieczonego produktu MQTT client sample C app

Uruchom skrypty w produkcie “Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows” na stronie 94 przed uruchomieniem tych skryptów.

Zabezpiecz MQTT client sample C app za pomocą samopodpisanych certyfikatów.
Uruchom ten skrypt z samopodpisanymi certyfikatami, które zostały utworzone przez uruchomienie skryptu `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
```

Rysunek 18. `sscclient.bat`

Uruchom przykładową aplikację C klienta bezpiecznego MQTT z certyfikatami podpisanymi przez ośrodek certyfikacji.

Uruchom ten skrypt przy użyciu certyfikatów podpisanych przez ośrodek certyfikacji, które zostały utworzone przez uruchomienie skryptu `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
```

Rysunek 19. `cacclient.bat`

Pojęcia pokrewne

“Zabezpieczenia produktu MQTT” na stronie 54
Trzy pojęcia są fundamentalne dla bezpieczeństwa produktu MQTT : tożsamość, uwierzytelnianie i autoryzacja. Tożsamość określa nazwę klienta, który jest autoryzowany i nadawany. Uwierzytelnianie polega na potwierdzeniu tożsamości klienta, a autoryzacja polega na zarządzaniu prawami, które nadawane są klientowi.

Zadania pokrewne

“Generowanie kluczy i certyfikatów” na stronie 100
Wykonaj tę procedurę, aby wygenerować klucze i certyfikaty dla klientów Java i C, w tym aplikacji Android i iOS oraz serwerów IBM WebSphere MQ i IBM MessageSight .

Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows

Przykład

Przykładowe pliki komend tworzą certyfikaty i bazy certyfikatów tak, jak to zostało opisane w krokach w zadaniu. Ponadto konfigurują menedżer kolejek klienta MQTT pod kątem korzystania z bazy certyfikatów serwera. Usuwają i ponownie tworzą menedżer kolejek, wywołując skrypt `SampleMQM.bat` dostarczony z produktem IBM WebSphere MQ.

`initcert.bat`

Skrypt `initcert.bat` ustawia nazwy certyfikatów i innych parametrów oraz ścieżki do nich wymagane przez komendy **keytool** i **openSSL**. Ustawienia zostały opisane w komentarzach w skrypcie.

```
@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

Komendy w skrypcie `cleancert.bat` usuwają menedżer kolejek klienta MQTT, aby mieć pewność, że baza certyfikatów serwera nie będzie zablokowana, a następnie usuwają wszystkie magazyny kluczy i certyfikaty utworzone przez przykładowe skrypty zabezpieczeń.

```
@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

Komendy w skrypcie `genkeys.bat` tworzą pary kluczy dla prywatnych ośrodków certyfikacji, serwera i klienta.

```
@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

Komendy w skrypcie `sscerts.bat` eksportują samopodpisane certyfikaty klienta i serwera z ich magazynów kluczy. Następnie importują certyfikat serwera do magazynu zaufanych certyfikatów klienta, a certyfikat klienta do magazynu kluczy serwera. Serwer nie ma magazynu zaufanych certyfikatów. Komendy tworzą magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt importuje certyfikat główny ośrodka certyfikacji do prywatnych magazynów kluczy. Certyfikat główny ośrodka CA jest potrzebny do utworzenia łańcucha kluczy między certyfikatem głównym a podpisanym certyfikatem. Skrypt `cacerts.bat` eksportuje żądania certyfikatów klienta i serwera z własnych magazynów kluczy. Skrypt podpisuje żądania certyfikatów przy użyciu klucza prywatnego ośrodka certyfikacji znajdującego się w magazynie kluczy `cajkskeystore.jks`, a następnie importuje podpisane certyfikaty z powrotem do tych magazynów kluczy, z których zostały wysłane żądania. Podczas importu tworzony jest łańcuch łączący certyfikat z certyfikatem głównym ośrodka CA. Skrypt tworzy magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt generuje listę magazynów kluczy i certyfikatów w katalogu certyfikatów. Następnie tworzy przykładowy menedżer kolejek produktu MQTT i konfiguruje zabezpieczone kanały pomiarowe.

```
@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
```

Generowanie kluczy i certyfikatów

Wykonaj tę procedurę, aby wygenerować klucze i certyfikaty dla klientów Java i C, w tym aplikacji Android i iOS oraz serwerów IBM WebSphere MQ i IBM MessageSight .

Zanim rozpoczniesz

1. Użytkownik musi mieć kopię komendy **keytool** . Nie wszystkie wersje produktu **keytool** obsługują konwersję plików kluczy z magazynu kluczy Java (JKS) do systemu kryptograficznego klucza publicznego (Public Key Cryptographic System-PKCS) lub podpisywanie żądań certyfikatów. W tym przykładzie użyto komendy **keytool** w produkcie JDK w wersji 7.0, która obsługuje obie te możliwości.
2. Jeśli użytkownik zamierza generować klucze i certyfikaty dla klienta dla języka C, które znajdują się w formacie PEM (Privacy-Enhanced Mail), to musi mieć kopię komendy **openSSL** . Wykonaj kroki opisane w sekcji “Budowanie klienta MQTT dla bibliotek C” na stronie 32 , aby zbudować pakiet openSSL .
3. Zmień wartości parametrów w skrypcie `initcert.bat` , aby zaspokoić potrzeby użytkownika. W szczególności można pominąć parametry hasła w celu uniknięcia zapisywania haseł w dół. Komenda **keytool** wyświetla zachętę do wprowadzenia brakujących haseł.

O tym zadaniu

Wymagane są klucze i certyfikaty do tworzenia bezpiecznych połączeń SSL między klientami i serwerami MQTT . To zadanie przedstawia dwa różne sposoby tworzenia kluczy i certyfikatów, które są wymagane: samopodpisany i podpisany przez własny ośrodek certyfikacji. Stosowana metoda zależy od tego, jak planujesz zarządzać magazynami kluczy i certyfikatami.

Aby użyć certyfikatów podpisanych przez zewnętrzny ośrodek certyfikacji, należy zastąpić krok podpisywania w `cacerts.bat`, wysyłając żądania certyfikatów do zewnętrznego ośrodka certyfikacji. Oprócz podpisanego certyfikatu ośrodek certyfikacji może zwrócić certyfikat pośredni i główny. Należy postępować zgodnie z wytycznymi udostępnionego przez zewnętrzny ośrodek CA, w którym mają zostać zainstalowane zwrócone certyfikaty.

Serwer IBM WebSphere MQ wyszukuje certyfikaty tylko w bazie certyfikatów określonej przez użytkownika w parametrach konfiguracyjnych kanału telemetrycznego. Nie jest on dodatkowo wyszukiwany w sklepie JSE `cacerts` . Klient Java wyszukuje certyfikaty w magazynie zaufanych certyfikatów, który został określony przez użytkownika. Jeśli magazyn zaufanych certyfikatów nie zostanie określony, będzie on wyszukiwane w magazynie kluczy `cacerts` w katalogu JSE `jre\lib\security` . Klienty Android wyszukają certyfikaty w predefiniowanej bazie certyfikatów na urządzeniu Android . Aplikacje klienckie C i aplikacje produktu iOS wyszukną tylko w bazie certyfikatów, w której znajduje się aplikacja.

Klienty Android i Java wyszuka wstępnie skonfigurowany magazyn zaufanych certyfikatów dla zaufanych certyfikatów. Certyfikaty główne ośrodka CA są przechowywane w zaufanej bazie certyfikatów produktu Android oraz w sklepie JSE `jre\lib\security\cacerts` . Jeśli certyfikat główny ośrodka certyfikacji (CA), który certyfikował certyfikat serwera, jest już zainstalowany w wstępnie skonfigurowanym magazynie zaufanych certyfikatów, nie należy definiować magazynu zaufanych certyfikatów klienta. Jedyna wymagana konfiguracja polega na ustawieniu portu TCP/IP dla zabezpieczonego kanału serwera MQTT .

Narzędzia do tworzenia kluczy i certyfikatów oraz zarządzania wszystkimi różnymi formatami, nie są proste w użyciu. Mają one wiele parametrów do zarządzania, a produkt **openSSL** wymaga pliku konfiguracyjnego, `openssl.cnf`oraz parametrów wiersza komend. Żadne narzędzie nie udostępnia wszystkich funkcji wymaganych do zarządzania kluczami i certyfikatami dla aplikacji, które działają zarówno w języku C, jak i w produkcie Java. Kanały telemetryczne w produkcie IBM WebSphere MQ wymagają magazynu kluczy JKS, a więc przykłady korzystają głównie z narzędzi certyfikatów Java , **ikeyman** i **keytool**. Jednak narzędzia Java nie obsługują formatu PEM, który jest wymagany dla aplikacji klienckich C. Aby utworzyć magazyny kluczy w formacie PEM, należy uruchomić narzędzie **openSSL** . Narzędzie **openSSL** przekształca pliki kluczy z formatu PKCS12 w format PEM, a program **keytool** przekształca pliki kluczy między formatem JKS a formatem PKCS12 . Do konwersji magazynu kluczy nie jest wymagany żaden plik `openssl.cnf` . Produkt **openSSL** jest wymagany tylko wtedy, gdy planowane jest budowanie aplikacji klienckich C lub aplikacji iOS . Jeśli wolisz pracować z produktem **openSSL**, można użyć go do podpisywania certyfikatów zamiast podpisywania certyfikatów za pomocą produktu **keytool**.

Procedura

1. Otwórz okno komend, aby uruchomić następujące skrypty.
2. Utwórz i uruchom skrypt `initcert.bat` , aby ustawić parametry, które są wymagane do uruchomienia bezpiecznych klientów przykładowych produktu MQTT .
3. Utwórz i uruchom skrypt `cleancert.bat` , aby wyczyścić środowisko gotowe do tworzenia nowych magazynów kluczy i certyfikatów.
4. Utwórz i uruchom skrypt `genkeys.bat` w celu wygenerowania par kluczy wymaganych przez użytkownika.
5. Wykonaj jedną z następujących czynności:
   - Utwórz i uruchom skrypt `sscerts.bat` , aby utworzyć samopodpisane certyfikaty.
   - Utwórz i uruchom skrypt `cacerts.bat` w celu utworzenia łańcuchów certyfikatów podpisanych przez ośrodek certyfikacji.
6. Utwórz i uruchom skrypt `mqcerts.bat` , aby utworzyć menedżer kolejek produktu `MQXR_SAMPLE_QM` i skonfigurować jego kanały telemetryczne.

Zadania pokrewne

“Budowanie i uruchamianie zabezpieczonego serwera MQTT client sample C app” na stronie 89
Na przykład w oparciu o przykład Windows można pobrać i uruchomić aplikację Secure sample C w dowolnym systemie operacyjnym, dla którego można skompilować źródło C. Sprawdź, czy można uruchomić przykładową aplikację C w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

“Budowanie i uruchamianie zabezpieczonego serwera Przykładowa aplikacja klienta MQTT Java” na stronie 58
W oparciu o przykład Windows można uruchomić i uruchomić aplikację Java z przykładową bezpieczną przykładową aplikacją w systemie IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT. Aplikację Klient MQTT dla produktu Java można uruchomić na dowolnej platformie z produktem JSE 1.5 lub nowejwersji, która jest kompatybilna z produktem "Java"

Przykładowe skrypty służące do konfigurowania certyfikatów SSL dla produktu Windows

Przykład

Przykładowe pliki komend tworzą certyfikaty i bazy certyfikatów tak, jak to zostało opisane w krokach w zadaniu. Ponadto konfigurują menedżer kolejek klienta MQTT pod kątem korzystania z bazy certyfikatów serwera. Usuwają i ponownie tworzą menedżer kolejek, wywołując skrypt `SampleMQM.bat` dostarczony z produktem IBM WebSphere MQ.

initcert.bat

Skrypt `initcert.bat` ustawia nazwy certyfikatów i innych parametrów oraz ścieżki do nich wymagane przez komendy **keytool** i **openSSL**. Ustawienia zostały opisane w komentarzach w skrypcie.

```
@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

Komendy w skrypcie `cleancert.bat` usuwają menedżer kolejek klienta MQTT, aby mieć pewność, że baza certyfikatów serwera nie będzie zablokowana, a następnie usuwają wszystkie magazyny kluczy i certyfikaty utworzone przez przykładowe skrypty zabezpieczeń.

```
@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

Komendy w skrypcie `genkeys.bat` tworzą pary kluczy dla prywatnych ośrodków certyfikacji, serwera i klienta.

```
@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

Komendy w skrypcie `sscerts.bat` eksportują samopodpisane certyfikaty klienta i serwera z ich magazynów kluczy. Następnie importują certyfikat serwera do magazynu zaufanych certyfikatów klienta, a certyfikat klienta do magazynu kluczy serwera. Serwer nie ma magazynu zaufanych certyfikatów. Komendy tworzą magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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

Skrypt importuje certyfikat główny ośrodka certyfikacji do prywatnych magazynów kluczy. Certyfikat główny ośrodka CA jest potrzebny do utworzenia łańcucha kluczy między certyfikatem głównym a podpisanym certyfikatem. Skrypt `cacerts.bat` eksportuje żądania certyfikatów klienta i serwera z własnych magazynów kluczy. Skrypt podpisuje żądania certyfikatów przy użyciu klucza prywatnego ośrodka certyfikacji znajdującego się w magazynie kluczy `cajkskeystore.jks`, a następnie importuje podpisane certyfikaty z powrotem do tych magazynów kluczy, z których zostały wysłane żądania. Podczas importu tworzony jest łańcuch łączący certyfikat z certyfikatem głównym ośrodka CA. Skrypt tworzy magazyn zaufanych certyfikatów klienta w formacie PEM na podstawie magazynu zaufanych certyfikatów JKS klienta.

```
@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
Skrypt generuje listę magazynów kluczy i certyfikatów w katalogu certyfikatów. Następnie tworzy przykładowy menedżer kolejek produktu MQTT i konfiguruje zabezpieczone kanały pomiarowe.

```
@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
```

Identyfikacja klienta MQTT, autoryzacja i uwierzytelnianie

Usługa telemetryczna (MQXR) publikuje lub subskrybuje tematy produktu WebSphere MQ w imieniu klientów MQTT za pomocą kanałów MQTT. Administrator produktu WebSphere MQ konfiguruje tożsamość kanału MQTT, która jest używana dla autoryzacji produktu WebSphere MQ . Administrator może zdefiniować wspólną tożsamość dla kanału lub użyć wartości `Username` lub `ClientIdentifier` klienta połączonego z kanałem.

Usługa telemetryczna (MQXR) może uwierzytelnić klienta przy użyciu `nazwy użytkownika` dostarczonej przez klienta lub za pomocą certyfikatu klienta. `Nazwa użytkownika` jest uwierzytelniana przy użyciu hasła udostępnionego przez klienta.

Podsumowując: Identyfikacja klienta to wybór tożsamości klienta. W zależności od kontekstu klient jest identyfikowany przez `ClientIdentifier`(identyfikator klienta), `Username`(Nazwa użytkownika), wspólną tożsamość klienta utworzoną przez administratora lub certyfikat klienta. Identyfikator klienta używany do sprawdzania autentyczności nie musi być tym samym identyfikatorem, który jest używany do autoryzacji.

Programy klienckie MQTT ustawiają wartości `Nazwa użytkownika` i `Hasło` , które są wysyłane do serwera przy użyciu kanału MQTT. Mogą także ustawiać właściwości protokołu SSL, które są wymagane do szyfrowania i uwierzytelniania połączenia. Administrator decyduje o tym, czy uwierzytelnić kanał MQTT, a także określić sposób uwierzytelniania kanału.

Aby autoryzować klienta MQTT w celu uzyskania dostępu do obiektów IBM WebSphere MQ , autoryzuj `ClientIdentifier`lub `Username` klienta albo autoryzuj wspólną tożsamość klienta. Aby umożliwić klientowi nawiązanie połączenia z serwerem IBM WebSphere MQ, należy uwierzytelnić nazwę użytkownika `Nazwa użytkownika`lub użyć certyfikatu klienta. Skonfiguruj usługę JAAS w celu uwierzytelnienia `nazwy użytkownika`i skonfiguruj protokół SSL w celu uwierzytelnienia certyfikatu klienta.

Jeśli w kliencie zostanie ustawione `Hasło` , zaszyfruj połączenie za pomocą sieci VPN lub skonfiguruj kanał MQTT tak, aby używał protokołu SSL, aby zachować hasło prywatne.

Zarządzanie certyfikatami klientów jest trudne. Z tego powodu, jeśli ryzyko związane z uwierzytelnianiem hasłem jest akceptowalne, uwierzytelnianie za pomocą hasła jest często używane do uwierzytelniania klientów.

Jeśli istnieje bezpieczny sposób zarządzania i przechowywania certyfikatu klienta, możliwe jest oparcie się na uwierzytelnianiu certyfikatów. Rzadko jednak zdarza się, że certyfikaty mogą być bezpiecznie zarządzane w różnych środowiskach, w których używana jest telemetria. Zamiast tego uwierzytelnianie urządzeń za pomocą certyfikatów klienta jest uzupełniane przez uwierzytelnianie haseł klientów na serwerze. Ze względu na dodatkową złożoność korzystanie z certyfikatów klientów jest ograniczone do wysoce wrażliwych aplikacji. Użycie dwóch form uwierzytelniania jest nazywane uwierzytelnianiem dwuskładniowym. Użytkownik musi znać jeden z czynników, na przykład hasło, i mieć inne, takie jak certyfikat.

W wysoce czułych aplikacjach, takich jak urządzenie chipowe i przypinane, urządzenie jest zamykane w trakcie produkcji, aby zapobiec ingerowaniu w wewnętrzny sprzęt i oprogramowanie. Zaufany, ograniczony czas, certyfikat klienta jest kopiowany do urządzenia. Urządzenie jest wdrażane w miejscu, w którym ma być używane. Dalsze uwierzytelnianie jest wykonywane za każdym razem, gdy urządzenie jest używane, albo przy użyciu hasła, albo na innym certyfikacie z karty Smart Card.

Tożsamość i autoryzacja klienta MQTT

Aby uzyskać dostęp do obiektów produktu WebSphere MQ , należy użyć wartości `ClientIdentifier`, `Username`lub wspólnej tożsamości klienta.

Administrator produktu IBM WebSphere MQ ma trzy opcje wyboru tożsamości kanału MQTT. Administrator dokonuje wyboru przy definiowaniu lub modyfikowaniu kanału MQTT używanego przez klienta. Tożsamość jest używana do autoryzowania dostępu do tematów produktu IBM WebSphere MQ . Dostępne opcje:

1. Identyfikator klienta.
2. Tożsamość, którą administrator udostępnia dla kanału.
3. Wartość `Nazwa użytkownika` przekazana z klienta MQTT.

`Nazwa użytkownika` jest atrybutem klasy opcji MqttConnect. Musi on zostać ustawiony przed nawiążniem połączenia z usługą. Jego wartość domyślna to NULL.

Komenda IBM WebSphere MQ **`setmqaut`** służy do wybierania obiektów, które mają być używane przez tożsamość powiązaną z kanałem MQTT, i które mają być używane. Na przykład, aby autoryzować tożsamość kanału, `MQTTClient`, udostępnionego przez administratora menedżera kolejek, `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
```

Informacje pokrewne

Autoryzowanie klientów MQTT w celu uzyskania dostępu do obiektów produktu WebSphere MQ

Uwierzytelnianie klienta MQTT przy użyciu hasła

Uwierzytelniaj `nazwę użytkownika` przy użyciu hasła klienta. Istnieje możliwość uwierzytelnienia klienta przy użyciu innej tożsamości do tożsamości używanej do autoryzowania klienta do publikowania i subskrybowania tematów.

Usługa telemetryczna (MQXR) korzysta z usługi JAAS w celu uwierzytelnienia klienta `Nazwa użytkownika`. Usługa JAAS korzysta z `hasła` dostarczonego przez klienta MQTT.

Administrator produktu IBM WebSphere MQ podejmuje decyzję o tym, czy uwierzytelnić `nazwę użytkownika`, czy nie w ogóle uwierzytelnić, konfigurując kanał połączenia MQTT, z którym łączy się klient. Klienci mogą być przypisani do różnych kanałów, a każdy kanał może być skonfigurowany do uwierzytelniania swoich klientów na różne sposoby. Za pomocą usługi JAASmożna skonfigurować, które metody muszą uwierzytelniać klienta, i które opcjonalnie mogą uwierzytelnić klienta.

Wybór tożsamości do uwierzytelniania nie ma wpływu na wybór tożsamości do autoryzacji. Użytkownik może chcieć skonfigurować wspólną tożsamość w celu autoryzacji dla wygody administratora, ale uwierzytelnia każdego użytkownika w celu użycia tej tożsamości. Poniższa procedura przedstawia kroki w celu uwierzytelnienia poszczególnych użytkowników w celu użycia wspólnej tożsamości:

1. Administrator produktu IBM WebSphere MQ ustawia tożsamość kanału MQTT na dowolną nazwę, taką jak `MQTTClientUser`, za pomocą programu IBM WebSphere MQ Explorer.
2. Administrator produktu IBM WebSphere MQ autoryzuje produkt `MQTTClient` do publikowania i subskrybowania dowolnego tematu:

```
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. Programista aplikacji klienckiej MQTT tworzy obiekt `MqttConnectOptions` i ustawia wartości `Username` i `Password` przed nawiązywaniem połączenia z serwerem.
4. Programista zabezpieczeń tworzy usługę JAAS `LoginModule` w celu uwierzytelnienia `nazwy użytkownika` przy użyciu `hasła` i dołącza ją do pliku konfiguracyjnego JAAS .
5. Administrator produktu IBM WebSphere MQ konfiguruje kanał MQTT w celu uwierzytelniania `UserName` klienta przy użyciu usługi JAAS.

Uwierzytelnianie klienta MQTT przy użyciu protokołu SSL

Połączenia między klientem MQTT a menedżerem kolejek są zawsze inicjowane przez klient MQTT. Klient MQTT jest zawsze klientem SSL. Uwierzytelnianie klienta serwera i uwierzytelnianie serwera klienta MQTT są opcjonalne.

Dzięki udostępnieniu klienta z prywatnym podpisanym certyfikatem cyfrowym można uwierzytelnić klienta MQTT w produkcie IBM WebSphere MQ. Administrator produktu IBM WebSphere MQ może wymusić na klientach MQTT uwierzytelnianie się w menedżerze kolejek przy użyciu protokołu SSL. Tylko w ramach uwierzytelniania wzajemnego można zażądać uwierzytelnienia klienta.

Alternatywę dla protokołu SSL stanowią niektóre rodzaje wirtualnej sieci prywatnej (VPN), takie jak IPsec, uwierzytelniające punkty końcowe połączenia TCP/IP. Sieć VPN szyfruje każdy pakiet IP przepływający przez sieć. Po ustanowieniu połączenia VPN uzyskana sieć jest zaufana. Klienty MQTT można połączyć z kanałami pomiarowymi przy użyciu połączeń TCP/IP za pośrednictwem sieci VPN.

Uwierzytelnianie klienta przy użyciu protokołu SSL wymaga, aby klient miał klucz tajny. Klucz tajny to klucz prywatny klienta stosowany na potrzeby certyfikatu samopodpisanego lub klucz udostępniony przez ośrodek certyfikacji. Ten klucz służy do podpisywania certyfikatu cyfrowego klienta. Odpowiedni klucz publiczny umożliwia weryfikację certyfikatu cyfrowego. Certyfikaty mogą być zaufanymi certyfikatami. Jeśli są połączone w łańcuch, można prześledzić łańcuch certyfikatów aż do zaufanego certyfikatu głównego. Podczas weryfikacji klienta wszystkie certyfikaty tworzące łańcuch certyfikatów udostępnione przez klienta są wysyłane na serwer. Serwer sprawdza łańcuch certyfikatów, aż znajdzie zaufany certyfikat. Zaufany certyfikat to certyfikat publiczny wygenerowany na podstawie certyfikatu samopodpisanego lub certyfikat główny wydany (zwykle) przez ośrodek certyfikacji. W ramach opcjonalnego ostatniego kroku zaufany certyfikat można porównać z aktualną listą odwołań certyfikatów.

Zaufanym certyfikatem może być certyfikat wystawiony przez ośrodek certyfikacji, który został już uwzględniony w bazie certyfikatów środowiska JRE. Może to być certyfikat samopodpisany lub dowolny certyfikat, który został dodany do magazynu kluczy kanału pomiarowego jako zaufany certyfikat.

Uwaga: Kanał pomiarowy ma połączony magazyn kluczy / magazyn zaufanych certyfikatów, w którym przechowywane są klucze prywatne do jednego lub większej liczby kanałów pomiarowych oraz wszystkie certyfikaty publiczne wymagane na potrzeby uwierzytelniania klientów. Baza certyfikatów środowiska JRE nigdy nie jest przywoływana, ponieważ kanał SSL musi mieć magazyn kluczy i jest to ten sam plik, który jest używany na potrzeby magazynu zaufanych certyfikatów kanału. Oznacza to, że jeśli uwierzytelnianie klienta wymaga certyfikatu głównego ośrodka CA, należy umieścić certyfikat główny w magazynie kluczy tego kanału nawet wtedy, gdy główny certyfikat ośrodka CA znajduje się już w bazie certyfikatów środowiska JRE. Baza certyfikatów środowiska JRE nigdy nie jest przywoływana.

Należy pomyśleć o zagrożeniach, jakim ma przeciwdziałać uwierzytelnianie klienta, oraz o rolach klienta i serwera w procesie przeciwdziałania tym zagrożeniom. Uwierzytelnienie samego certyfikatu klienta nie blokuje w pełni możliwości dostępu do systemu bez uprawnień. Jeśli inna osoba wejdzie w posiadanie urządzenia z klientem, urządzenie z klientem nie będzie działało z upoważnienia właściciela certyfikatu. Nigdy nie należy polegać na jednej formie obrony przed niepożądanymi atakami. Należy stosować przynajmniej uwierzytelnianie dwuskładnikowe i oprócz posiadania certyfikatu wymagać znajomości

prywatnych informacji. Na przykład można zastosować usługę JAAS i uwierzytelniać klient przy użyciu hasła wystawionego przez serwer.

Dostanie się w niepowołane ręce stanowi główne zagrożenie dla certyfikatu klienta. Certyfikat jest przechowywany w kliencie w magazynie kluczy zabezpieczonym hasłem. W jaki sposób jest on umieszczany w magazynie kluczy? W jaki sposób klient MQTT uzyskuje hasło do magazynu kluczy? Jak wysoki poziom bezpieczeństwa zapewnia zabezpieczenie hasłem? Często urządzenie pomiarowe można łatwo odłączyć. W takiej sytuacji haker może włamać się do urządzenia w miejscu prywatnym. Czy sprzęt urządzenia musi być zabezpieczony przed manipulacją? Uznaje się, że rozpowszechnianie i ochrona certyfikatów po stronie klienta to trudne zadanie – jest to tzw. problem z zarządzaniem kluczami.

Dodatkowe zagrożenie stanowi nieprawidłowe używanie urządzenia w celu uzyskiwania dostępu do serwerów w niezamierzony sposób. Na przykład zmanipulowanie aplikacji MQTT umożliwia wykorzystanie luki w konfiguracji serwera przy użyciu uwierzytelnionej tożsamości klienta.

Aby móc uwierzytelniać klienta MQTT przy użyciu protokołu SSL, należy skonfigurować kanał pomiarowy i klient.

•
•

Konfiguracja klienta MQTT dla uwierzytelniania klienta przy użyciu protokołu SSL

Aby uwierzytelnić klienta MQTT za pomocą protokołu SSL, klient łączy się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port TCP odpowiadający kanałowi telemetrycznego, który jest skonfigurowany do uwierzytelniania klientów SSL.

Na przykład na kliencie:

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

Maszyna JVM klienta musi używać standardowej fabryki gniazd z biblioteki JSSE. Jeśli korzystasz z Java ME, musisz upewnić się, że pakiet JSSE jest załadowany. Jeśli używana jest wersja Java SE, JSSE została dołączona do środowiska JRE od wersji Java 1.4.1.

Przed nawiązaniem połączenia SSL należy skonfigurować szereg właściwości SSL. Właściwości można ustawić, przekazując je do maszyny JVM za pomocą przełącznika `-D` lub można ustawić właściwości przy użyciu metody `MqttConnectionOptions.setSSLProperties` .

Jeśli zostanie załadowana niesstandardowa fabryka gniazd, wywołując metodę `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, wówczas sposób przekazywania ustawień SSL do gniazda sieciowego jest zdefiniowany przez aplikację.

Dodaj certyfikat cyfrowy klienta, podpisany za pomocą klucza prywatnego klienta lub ośrodka CA, do magazynu kluczy zabezpieczonego hasłem na kliencie. Jeśli certyfikat ma łańcuch kluczy, można dodać certyfikaty z łańcucha kluczy do sklepu. Gdy serwer weryfikuje certyfikat klienta, korzysta z certyfikatów wysyłanych przez klienta w celu dopasowania ich do certyfikatów w jego magazynie kluczy. Szuka on pierwszego dopasowania w łańcuchu kluczy z certyfikatem, który posiada. Pozostała część łańcucha kluczy jest ignorowana.

Klient MQTT wysyła wszystkie certyfikaty z magazynu kluczy do serwera. Jeśli serwer uwierzytelnia wszystkie łańcuchy kluczy, które wysyła klient, klient zostanie uwierzytelniony.

Do uwierzytelniania klientów można również używać zestawów algorytmów szyfrowania SSL. Poniżej znajduje się alfabetyczna lista obsługiwanych obecnie zestawów algorytmów szyfrowania SSL:

- `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 Jeśli planowane jest korzystanie z zestawów algorytmów szyfrowania SHA-2 , należy zapoznać się z “Wymagania systemowe dotyczące korzystania z zestawów algorytmów szyfrowania SHA-2 z klientami MQTT” na stronie 180.

Pojęcia pokrewne

“Konfiguracja klienta MQTT dla uwierzytelniania kanału przy użyciu protokołu SSL” na stronie 112
Aby uwierzytelnić kanał pomiarowy przy użyciu protokołu SSL, klient musi połączyć się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port odpowiadający kanałowi telemetrycznego,

który jest skonfigurowany dla protokołu SSL. Konfiguracja musi zawierać zabezpieczony magazyn kluczy chroniony hasłem, który zawiera certyfikat cyfrowy podpisany przez prywatnie serwer.

Uwierzytelnianie kanału telemetrycznego przy użyciu protokołu SSL

Połączenia między klientem MQTT a menedżerem kolejek są zawsze inicjowane przez klient MQTT. Klient MQTT jest zawsze klientem SSL. Uwierzytelnianie klienta serwera i uwierzytelnianie serwera klienta MQTT są opcjonalne.

Klient zawsze będzie próbował uwierzytelnić serwer, chyba że klient został skonfigurowany pod kątem używania specyfikacji szyfrowania obsługującej połączenia anonimowe. Jeśli uwierzytelnianie nie powiedzie się, połączenie nie zostanie nawiązane.

Alternatywę dla protokołu SSL stanowią niektóre rodzaje wirtualnej sieci prywatnej (VPN), takie jak IPsec, uwierzytelniające punkty końcowe połączenia TCP/IP. Sieć VPN szyfruje każdy pakiet IP przepływający przez sieć. Po ustanowieniu połączenia VPN uzyskana sieć jest zaufana. Klienty MQTT można połączyć z kanałami pomiarowymi przy użyciu połączeń TCP/IP za pośrednictwem sieci VPN.

Uwierzytelnianie serwera za pomocą protokołu SSL uwierzytelnia serwer, do którego użytkownik chce wysłać informacje poufne. Klient wykonuje sprawdzenia zgodne z certyfikatami wysłanym z serwera, z certyfikatami umieszczonym w magazynie zaufanych certyfikatów lub w jego składnicy JRE `cacerts` .

Składnica certyfikatów środowiska JRE to plik JKS `cacerts`. Znajduje się on w katalogu JRE `InstallPath\lib\security\`. Jest on instalowany z domyślnym hasłem `changeit`. Zaufane certyfikaty mogą być przechowywane w bazie certyfikatów środowiska JRE lub w magazynie zaufanych certyfikatów klienta. Nie można używać obu składnic. Należy skorzystać z magazynu zaufanych certyfikatów, jeśli certyfikaty publiczne i zaufane certyfikaty klienta mają być przechowywane w innym miejscu niż certyfikaty używane przez inne aplikacje Java. Należy skorzystać z bazy certyfikatów środowiska JRE, jeśli wszystkie aplikacje Java działające na kliencie mają używać wspólnej bazy certyfikatów. W przypadku wybrania bazy certyfikatów środowiska JRE należy przejrzeć certyfikaty w niej zawarte, aby upewnić się, że są to zaufane certyfikaty.

Konfigurację JSSE można zmodyfikować, podając innego dostawcę zaufania. Istnieje możliwość dostosowania dostawcy zaufania tak, aby wykonywał inne operacje sprawdzania certyfikatu. W niektórych środowiskach OGSi, które korzystały z klienta MQTT, środowisko udostępnia innego dostawcę zaufania.

Aby uwierzytelnić kanał pomiarowy przy użyciu protokołu SSL, należy skonfigurować serwer i klient.

•

Pojęcia pokrewne

“Konfiguracja klienta MQTT dla uwierzytelniania kanału przy użyciu protokołu SSL” na stronie 112
Aby uwierzytelnić kanał pomiarowy przy użyciu protokołu SSL, klient musi połączyć się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port odpowiadający kanałowi telemetrycznego, który jest skonfigurowany dla protokołu SSL. Konfiguracja musi zawierać zabezpieczony magazyn kluczy chroniony hasłem, który zawiera certyfikat cyfrowy podpisany przez prywatnie serwer.

Konfiguracja klienta MQTT dla uwierzytelniania kanału przy użyciu protokołu SSL

Aby uwierzytelnić kanał pomiarowy przy użyciu protokołu SSL, klient musi połączyć się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port odpowiadający kanałowi telemetrycznego, który jest skonfigurowany dla protokołu SSL. Konfiguracja musi zawierać zabezpieczony magazyn kluczy chroniony hasłem, który zawiera certyfikat cyfrowy podpisany przez prywatnie serwer.

Na przykład na kliencie:

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

Maszyna JVM klienta musi używać standardowej fabryki gniazd z biblioteki JSSE. Jeśli korzystasz z Java ME, musisz upewnić się, że pakiet JSSE jest załadowany. Jeśli używana jest wersja Java SE, JSSE została dołączona do środowiska JRE od wersji Java 1.4.1.

Przed nawiązaniem połączenia SSL należy skonfigurować szereg właściwości SSL. Właściwości można ustawić, przekazując je do maszyny JVM za pomocą przełącznika `-D` lub można ustawić właściwości przy użyciu metody `MqttConnectionOptions.setSSLProperties` .

Jeśli zostanie załadowana niesstandardowa fabryka gniazd, wywołując metodę `MqttConnectOptions.setSocketFactory(javax.net.SocketFactory)`, wówczas sposób przekazywania ustawień SSL do gniazda sieciowego jest zdefiniowany przez aplikację.

Kod klienta, który ma być połączony z kanałem pomiarowym za pomocą protokołu SSL, a następnie skonfiguruj klienta w taki sposób, aby ufał certyfikatowi serwera na jeden z trzech sposobów:

Korzystanie z certyfikatu serwera podpisanego przez powszechnie znane uprawnienia do certyfikatów w sklepie `cacerts` .

Brak dodatkowej konfiguracji, jeśli serwer wysyła wszystkie klucze pośrednie w łańcuchu certyfikatów. Zaleca się, aby przejrzeć certyfikaty w składnicy `cacerts` klienta JRE i zmienić hasło do sklepu `cacerts` .

Pozostałe świadectwa

Zapisz certyfikaty zaufane w magazynie zaufanych certyfikatów na kliencie. Należy zapisać co najmniej jeden certyfikat w łańcuchu certyfikatów w magazynie zaufanych certyfikatów, a następnie ustawić parametry magazynu zaufanych certyfikatów w produkcie `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStore`
- `com.ibm.ssl.trustStorePassword`

Korzystanie z niestandardowego menedżera zaufania

Zaimplementuj zaufany dostawca i przekaz mu nazwę używanego algorytmu. Ustaw nazwę klasy dostawcy i algorytm, który ma być używany w produkcie `MqttConnectionOptions.SSLProperty`.

- `com.ibm.ssl.trustStoreProvider`
- `com.ibm.ssl.trustStoreManager`

Do uwierzytelniania kanału można również używać zestawów algorytmów szyfrowania SSL. Poniżej znajduje się alfabetyczna lista obsługiwanych obecnie zestawów algorytmów szyfrowania SSL:

- `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 Jeśli planowane jest korzystanie z zestawów algorytmów szyfrowania SHA-2 , należy zapoznać się z “Wymagania systemowe dotyczące korzystania z zestawów algorytmów szyfrowania SHA-2 z klientami MQTT” na stronie 180.

Pojęcia pokrewne

“Konfiguracja klienta MQTT dla uwierzytelniania klienta przy użyciu protokołu SSL” na stronie 110
Aby uwierzytelnić klienta MQTT za pomocą protokołu SSL, klient łączy się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port TCP odpowiadający kanałowi telemetrycznego, który jest skonfigurowany do uwierzytelniania klientów SSL.

Prywatność publikacji na kanałach telemetrycznych

Prywatność publikacji MQTT wysyłanych w obu kierunkach w kanałach telemetrycznych jest zabezpieczona za pomocą protokołu SSL w celu szyfrowania transmisji przez połączenie.

Klienty MQTT łączające się z kanałami telemetrycznymi korzystają z protokołu SSL w celu zabezpieczenia prywatności publikacji przesyłanych w kanale za pomocą kryptografii klucza symetrycznego. Ponieważ punkty końcowe nie są uwierzytelniane, szyfrowanie kanału nie jest możliwe tylko w przypadku szyfrowania. Połączenie zabezpieczania prywatności z serwerem lub uwierzytelniania wzajemnego.

Alternatywę dla protokołu SSL stanowią niektóre rodzaje wirtualnej sieci prywatnej (VPN), takie jak IPsec, uwierzytelniające punkty końcowe połączenia TCP/IP. Sieć VPN szyfruje każdy pakiet IP przepływający przez sieć. Po ustanowieniu połączenia VPN uzyskana sieć jest zaufana. Klienty MQTT można połączyć z kanałami pomiarowymi przy użyciu połączeń TCP/IP za pośrednictwem sieci VPN.

W przypadku typowej konfiguracji, która szyfruje kanał i uwierzytelnia serwer, należy zapoznać się z “Uwierzytelnianie kanału telemetrycznego przy użyciu protokołu SSL” na stronie 112.

Szyfrowanie połączeń SSL bez uwierzytelniania serwera naraża połączenie z atakami typu man-in-the-middle. Mimo, że informacje, które wymieniasz, są chronione przed podsłuchami, nie wiesz, z kim się wymieniasz. Jeśli sieć nie jest sterowana, użytkownik jest narażony na przechwytywanie transmisji danych IP i maskowanie jako punkt końcowy.

Szyfrowane połączenie SSL można utworzyć bez uwierzytelniania serwera przy użyciu wymiany kluczy Diffie-Hellman CipherSpec , która obsługuje anonimowy protokół SSL. Główny klucz tajny, współużytkowany między klientem i serwerem i używany do szyfrowania transmisji SSL, jest ustanawiany bez wymiany certyfikatu serwera prywatnego.

Ponieważ anonimowe połączenia są niezabezpieczone, większość implementacji protokołu SSL nie jest domyślnie używana do używania anonimowych specyfikacji CipherSpecs. Jeśli żądanie klienta dla połączenia SSL jest akceptowane przez kanał pomiarowy, kanał musi mieć magazyn kluczy chroniony hasłem. Domyślnie, ponieważ implementacje protokołu SSL nie używają anonimowych specyfikacji CipherSpecs, magazyn kluczy musi zawierać podpisany przez klienta certyfikat, który może być uwierzytelniony przez klienta.

Jeśli używana jest anonimowa opcja CipherSpecs, magazyn kluczy serwera musi istnieć, ale nie musi zawierać żadnych prywatnych certyfikatów podpisanych.

Innym sposobem nawiązania szyfrowanego połączenia jest zastąpienie dostawcy zaufanego na kliencie we własnej implementacji. Dostawca zaufania nie uwierzytelniłby certyfikatu serwera, ale połączenie będzie szyfrowane.

Konfiguracja SSL klientów MQTT i kanałów telemetrycznych

MQTT clients and the WebSphere MQ Telemetry (MQXR) service use Java Secure Socket Extension (JSSE) to connect telemetry channels using SSL. Klienty MQTT C, a także demon WebSphere MQ Telemetry dla urządzeń nie obsługują protokołu SSL.

Skonfiguruj protokół SSL, aby uwierzytelnić kanał pomiarowy i klienta MQTT, a także zaszyfruj przesyłanie komunikatów między klientami i kanałem telemetryczny.

Alternatywę dla protokołu SSL stanowią niektóre rodzaje wirtualnej sieci prywatnej (VPN), takie jak IPsec, uwierzytelniające punkty końcowe połączenia TCP/IP. Sieć VPN szyfruje każdy pakiet IP przepływający przez sieć. Po ustanowieniu połączenia VPN uzyskana sieć jest zaufana. Klienty MQTT można połączyć z kanałami pomiarowymi przy użyciu połączeń TCP/IP za pośrednictwem sieci VPN.

Istnieje możliwość skonfigurowania połączenia między klientem Java MQTT a kanałem telemetryczny w celu użycia protokołu SSL przez protokół TCP/IP. To, co jest zabezpieczone, zależy od tego, jak skonfigurować protokół SSL do korzystania z rozszerzenia JSSE. Począwszy od konfiguracji najbardziej zabezpieczonej, można skonfigurować trzy różne poziomy zabezpieczeń:

1. Zezwól tylko zaufanym klientom MQTT na połączenie. Połącz klienta MQTT tylko z zaufanym kanałem telemetryczny. Szyfruj komunikaty między klientem i menedżerem kolejek; patrz “Uwierzytelnianie klienta MQTT przy użyciu protokołu SSL” na stronie 109.
2. Połącz klienta MQTT tylko z zaufanym kanałem telemetryczny. Szyfruj komunikaty między klientem i menedżerem kolejek; patrz “Uwierzytelnianie kanału telemetrycznego przy użyciu protokołu SSL” na stronie 112.
3. Szyfruj komunikaty między klientem i menedżerem kolejek; patrz “Prywatność publikacji na kanałach telemetrycznych” na stronie 114.

Parametry konfiguracyjne JSSE

Zmodyfikuj parametry JSSE w taki sposób, aby zmieniły sposób konfigurowania połączenia SSL. Parametry konfiguracyjne JSSE są rozmieszczone w trzech zestawach:

1. IBM WebSphere MQ Kanał telemetryczny
2. Klient Java MQTT

3. JRE (środowisko JRE)

Skonfiguruj parametry kanału telemetrycznego przy użyciu programu IBM WebSphere MQ Explorer. Ustaw parametry klienta Java MQTT w atrybucie `MqttConnectionOptions.SSLProperties` . Zmodyfikuj parametry zabezpieczeń środowiska JRE, edytując pliki w katalogu zabezpieczeń środowiska JRE zarówno na kliencie, jak i na serwerze.

IBM WebSphere MQ kanał telemetryczny

Ustaw wszystkie parametry protokołu SSL kanału telemetrycznego przy użyciu programu WebSphere MQ Explorer.

`ChannelName`

Parametr `ChannelName` jest wymaganym parametrem we wszystkich kanałach.

Nazwa kanału identyfikuje kanał powiązany z określonym numerem portu. Nazwy kanałów, które ułatwiają administrowanie zestawami klientów MQTT.

`PortNumber`

Parametr `PortNumber` jest opcjonalnym parametrem we wszystkich kanałach. Domyślnie jest to 1883 dla kanałów TCP, a 8883 dla kanałów SSL.

Numer portu TCP/IP powiązany z tym kanałem. Klienty MQTT są połączone z kanałem przez określenie portu zdefiniowanego dla kanału. Jeśli kanał ma właściwości SSL, klient musi nawiązać połączenie przy użyciu protokołu SSL, na przykład:

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

`KeyFileNazwa`

Parametr `KeyFileName` (Nazwa pliku kluczy) jest wymaganym parametrem dla kanałów SSL. Należy go pominąć w przypadku kanałów TCP.

`KeyFileNazwa` to ścieżka do magazynu kluczy Java zawierającego certyfikaty cyfrowe, które są dostępne. Użyj JKS, JCEKS lub PKCS12 jako typu magazynu kluczy na serwerze.

Zidentyfikuj typ magazynu kluczy, używając jednego z następujących rozszerzeń:

```
.jks
.jceks
.p12
.pkcs12
```

Zakłada się, że magazyn kluczy z innym rozszerzeniem nazwy pliku jest plikiem kluczy JKS.

Istnieje możliwość połączenia jednego typu magazynu kluczy na serwerze z innymi typami magazynów kluczy na kliencie.

Umieść certyfikat prywatny serwera w magazynie kluczy. Certyfikat jest znany jako certyfikat serwera. Certyfikat może być samopodpisany lub częścią łańcucha certyfikatów, który jest podpisany przez ośrodek podpisujący.

Jeśli używany jest łańcuch certyfikatów, należy umieścić powiązane certyfikaty w magazynie kluczy serwera.

Certyfikat serwera i wszystkie certyfikaty w łańcuchu certyfikatów są wysyłane do klientów w celu uwierzytelnienia tożsamości serwera.

Jeśli parametr `ClientAuth` został ustawiony na wartość `Required`, magazyn kluczy musi zawierać wszystkie certyfikaty niezbędne do uwierzytelnienia klienta. Klient wysyła samopodpisany certyfikat lub łańcuch certyfikatów, a klient jest uwierzytelniany przez pierwszą weryfikację tego materiału względem certyfikatu w magazynie kluczy. Korzystając z łańcucha certyfikatów, jeden certyfikat może weryfikować wiele klientów, nawet jeśli są one wystawiane z różnymi certyfikatami klienta.

PassPhrase

Parametr PassPhrase jest wymaganym parametrem dla kanałów SSL. Należy go pominąć w przypadku kanałów TCP.

Fraza hasła używana jest do ochrony magazynu kluczy.

ClientAuth

Parametr ClientAuth jest opcjonalnym parametrem SSL. Domyślnie nie jest to uwierzytelnianie klienta. Należy go pominąć w przypadku kanałów TCP.

Ustaw wartość ClientAuth , jeśli usługa telemetryczna (MQXR) ma być uwierzytelniana przez klienta, zanim klient będzie mógł nawiązać połączenie z kanałem telemetryczny.

Jeśli zostanie ustawiona wartość ClientAuth, klient musi połączyć się z serwerem za pomocą protokołu SSL i uwierzytelnić serwer. W odpowiedzi na ustawienie parametru ClientAuthklient wysyła swój certyfikat cyfrowy do serwera i wszystkie inne certyfikaty w magazynie kluczy. Jego certyfikat cyfrowy jest znany jako certyfikat klienta. Certyfikaty te są uwierzytelniane na certyfikatach znajdujących się w magazynie kluczy kanału i w sklepie JRE cacerts .

CipherSuite

Parametr CipherSuite jest opcjonalnym parametrem SSL. Wartością domyślną jest próba wszystkich włączonych specyfikacji CipherSpecs. Należy go pominąć w przypadku kanałów TCP.

Aby użyć konkretnej specyfikacji szyfrowania CipherSpec, należy ustawić właściwość CipherSuite na nazwę CipherSpec , która musi być używana do nawiązania połączenia SSL.

Usługa telemetryczna i klient MQTT negocjują wspólną specyfikację CipherSpec ze wszystkich obiektów CipherSpecs , które są włączone na każdym końcu. Jeśli określony parametr CipherSpec jest określony w obu lub obu końcach połączenia, musi być zgodny z atrybutem CipherSpec na drugim końcu.

Zainstaluj dodatkowe szyfry, dodając dodatkowych dostawców do rozszerzenia JSSE.

Federal Information Processing Standards (FIPS)

Tryb FIPS jest ustawieniem opcjonalnym. Domyślnie nie jest ona ustawiona.

Na panelu właściwości menedżera kolejek lub za pomocą opcji **runmqsc**ustaw wartość SSLFIPS. Opcja SSLFIPS określa, czy mają być używane tylko algorytmy z certyfikatem FIPS.

Lista nazw odwołań

Lista nazw odwołań jest ustawieniem opcjonalnym. Domyślnie nie jest ona ustawiona.

Na panelu właściwości menedżera kolejek lub za pomocą komendy **runmqsc**ustaw wartość SSLCRLNL. SSLCRLNL określa listę nazw obiektów informacji uwierzytelniających, które są używane do udostępniania położeń odwołań certyfikatów.

Nie są używane żadne inne parametry menedżera kolejek, które ustawiają właściwości SSL.

Klient Java MQTT

Ustaw właściwości SSL dla klienta Java w katalogu MqttConnectionOptions.SSLProperties, na przykład:

```
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);
```

Nazwy i wartości konkretnych właściwości są opisane w dokumentacji interfejsu API dla produktu MqttConnectOptions. W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT.

Protokół

Protokół jest opcjonalny.

Protokół jest wybierany w trakcie negocjacji z serwerem telemetryczny. Jeśli wymagany jest konkretny protokół, można wybrać jeden z nich. Jeśli serwer telemetryczny nie obsługuje protokołu, połączenie nie powiedzie się.

ContextProvider

Element ContextProvider jest opcjonalny.

KeyStore

Opcja KeyStore jest opcjonalna. Skonfiguruj ją, jeśli na serwerze ustawiona jest wartość ClientAuth , aby wymusić uwierzytelnianie klienta.

Umieść certyfikat cyfrowy klienta, podpisanego przy użyciu jego klucza prywatnego, do magazynu kluczy. Podaj ścieżkę do magazynu kluczy i hasło. Typ i dostawca są opcjonalne. Domyślnym typem dostawcy jest JKS, a dostawca IBMJCE jest dostawcą domyślnym.

Określ inny dostawca magazynu kluczy, aby odwołował się do klasy, która dodaje nowego dostawcę magazynu kluczy. Przekaz nazwę algorytmu używanego przez dostawcę magazynu kluczy do utworzenia instancji KeyManagerFactory przez ustawienie nazwy menedżera kluczy.

TrustStore

Opcja TrustStore jest opcjonalna. Użytkownik może umieścić wszystkie certyfikaty zaufane w sklepie JRE cacerts .

Skonfiguruj magazyn zaufanych certyfikatów, jeśli chcesz mieć inny magazyn zaufanych certyfikatów dla klienta. Magazyn zaufanych certyfikatów może nie zostać skonfigurowany, jeśli serwer używa certyfikatu wystawionego przez znany ośrodek CA, który ma już certyfikat główny zapisany w produkcie cacerts.

Dodaj do magazynu zaufanych certyfikatów publicznie podpisany certyfikat serwera lub certyfikat główny, a następnie podaj ścieżkę i hasło magazynu zaufanych certyfikatów. Domyślnym typem dostawcy jest JKS, a dostawca IBMJCE jest dostawcą domyślnym.

Określ inny dostawca magazynu zaufanych certyfikatów, aby odwołował się do klasy, która dodaje nowego dostawcę magazynu zaufanych certyfikatów. Przekaz nazwę algorytmu używanego przez dostawcę magazynu zaufanych certyfikatów do utworzenia instancji fabryki TrustManagerFactory przez ustawienie nazwy menedżera zaufania.

JRE (środowisko JRE)

Inne aspekty zabezpieczeń produktu Java , które mają wpływ na działanie protokołu SSL zarówno na kliencie, jak i na serwerze, są konfigurowane w środowisku JRE. Pliki konfiguracyjne w systemie Windows znajdują się w katalogu *Java Installation Directory*\jre\lib\security. Jeśli używane jest środowisko JRE dostarczane z produktem IBM WebSphere MQ , ścieżka jest wyświetlana w następującej tabeli:

Tabela 3. Ścieżki do plików przy użyciu platformy dla plików konfiguracyjnych SSL środowiska JRE

Platforma	**Ścieżka do pliku**
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux dla 32-bitowego systemu System x	*WMQ Installation Directory*/java/jre/lib/security
Inne platformy UNIX and Linux	*WMQ Installation Directory*/java/jre64/jre/lib/security

Powszechnie znane uprawnienia do certyfikatów

Plik `cacerts` zawiera certyfikaty główne powszechnie znanych ośrodków certyfikacji. `cacerts` jest używany domyślnie, chyba że zostanie określony magazyn zaufanych certyfikatów. Jeśli używany jest sklep `cacerts` lub magazyn zaufanych certyfikatów nie jest udostępniany, należy przejrzeć i edytować listę osób podpisujących w produkcie `cacerts` , aby spełnić wymagania bezpieczeństwa.

Produkt `cacerts` można otworzyć, korzystając z komendy WebSphere MQ `strmqikm`, która uruchamia program narzędziowy IBM Key Management. Otwórz plik `cacerts` jako plik JKS, korzystając z hasła `changeit`. Zmodyfikuj hasło, aby zabezpieczyć plik.

Konfigurowanie klas zabezpieczeń

Użyj pliku `java.security` , aby zarejestrować dodatkowych dostawców zabezpieczeń i inne domyślne właściwości zabezpieczeń.

Uprawnienia

Użyj pliku `java.policy` , aby zmodyfikować uprawnienia nadane zasobom. `javaws.policy` nadaje uprawnienia do produktu `javaws.jar`

Siła szyfrowania

Niektóre statki JREs z obniżonym szyfrowaniem wytrzymałości. Jeśli nie można zaimportować kluczy do magazynów kluczy, przyczyną może być zredukowane szyfrowanie mocy. Należy spróbować uruchomić produkt **ikeyman** przy użyciu komendy **strmqikm** lub pobrać silne, ale ograniczone pliki jurysdykcji z produktu Zestawy programistyczne produktu IBM , informacje o zabezpieczeniach.

Ważne: Twój kraj pochodzenia może mieć ograniczenia w imporcie, posiadaniu, użyciu lub powrotnym wywozie do innego kraju, oprogramowania szyfrującego. Przed pobraniem lub korzystaniem z nieograniczonej strategii należy sprawdzić prawa swojego kraju. Sprawdź jego regulacje, a także jego strategie dotyczące importu, posiadania, używania i ponownego eksportu oprogramowania szyfrującego, w celu ustalenia, czy jest to dozwolone.

Zmodyfikuj zaufany dostawca, aby umożliwić klientowi nawiązanie połączenia z dowolnym serwerem.

W tym przykładzie przedstawiono sposób dodawania dostawcy zaufania i odwołania go z kodu klienta MQTT. W tym przykładzie nie jest przeprowadzane uwierzytelnianie klienta lub serwera. Wynikowe połączenie SSL jest szyfrowane bez uwierzytelniania.

Fragment kodu w produkcie Rysunek 20 na stronie 119 ustawia dostawcę zaufania `AcceptAllProviders` i menedżera zaufania dla klienta 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);
```

Rysunek 20. Fragment kodu klienta 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());
    }
```

Rysunek 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() };
    }
```

Rysunek 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);
  }
}
```

Rysunek 23. AcceptAllX509TrustManager.java

Konfiguracja kanału telemetrycznego JAAS

Skonfiguruj usługę JAAS w taki sposób, aby uwierzytelniała `Nazwę użytkownika` wysłanego przez klienta.

Administrator produktu WebSphere MQ konfiguruje, które kanały MQTT wymagają uwierzytelniania klienta przy użyciu usługi JAAS. Podaj nazwę konfiguracji JAAS dla każdego kanału, który ma wykonać uwierzytelnianie JAAS . Kanały mogą korzystać z tej samej konfiguracji JAAS lub mogą używać różnych konfiguracji JAAS . Konfiguracje są definiowane w produkcie *WMQData directory*`\qmgrs\`*qMgrName*`\mqxr\jaas.config`.

Plik `jaas.config` jest organizowany przez nazwę konfiguracji JAAS . Pod każdą nazwą konfiguracji znajduje się lista konfiguracji logowania; patrz Rysunek 24 na stronie 121.

JAAS udostępnia cztery standardowe moduły logowania. Standardowe moduły logowania NT i UNIX mają ograniczoną wartość.

Moduł JndiLogin
Uwierzytelnia się przed usługą katalogową skonfigurowaną w interfejsie JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Uwierzytelnia się przy użyciu protokołów Kerberos .

NTLoginModule
Uwierzytelnia się przy użyciu informacji o zabezpieczeniach systemu NT dla bieżącego użytkownika.

Moduł UnixLogin
Uwierzytelnia się przy użyciu informacji o zabezpieczeniach systemu UNIX dla bieżącego użytkownika.

Problem z używaniem produktu `NTLoginModule` lub `UnixLoginModule` polega na tym, że usługa telemetryczna (MQXR) jest uruchamiana z tożsamością `mqm` , a nie tożsamością kanału MQTT. `mqm` jest

tożsamością przekazytą do produktu NTLoginModule lub UnixLoginModule w celu uwierzytelnienia, a nie tożsamości klienta.

Aby rozwiązać ten problem, napisz własny moduł logowania lub użyj innych standardowych modułów logowania. Przykładowy produkt JAASLoginModule.java jest dostarczany razem z produktem WebSphere MQ Telemetry. Jest to implementacja interfejsu javax.security.auth.spi.LoginModule . Użyj go do opracowania własnej metody uwierzytelniania.

Wszystkie nowe klasy LoginModule , które zostaną określone, muszą znajdować się w ścieżce klasy usługi telemetrii (MQXR). Nie należy umieszczać klas w katalogach produktu WebSphere MQ , które znajdują się w ścieżce klasy. Utwórz własne katalogi i zdefiniuj całą ścieżkę klasy dla usługi telemetrycznej (MQXR).

Ścieżkę klas używaną przez usługę telemetryczną (MQXR) można rozszerzyć, ustawiając ścieżkę klasy w pliku service.env . CLASSPATH musi być skapitalizowana, a instrukcja ścieżki klasy może zawierać tylko literały. W zmiennej CLASSPATH nie można używać zmiennych; na przykład wartość CLASSPATH=%CLASSPATH% jest niepoprawna. Usługa telemetryczna (MQXR) ustawia własną ścieżkę klasy. Zostanie do niego dodana ścieżka CLASSPATH zdefiniowana w produkcie service.env .

Usługa telemetryczna (MQXR) udostępnia dwa wywołania zwrotne, które zwracają wartości Nazwa użytkownika i Hasło dla klienta połączonego z kanałem MQTT. W obiekcie MqttConnectOptions ustawiane są wartości Nazwa użytkownika i Hasło . Zapoznaj się z przykładem dostępu do opcji Nazwa użytkownika i Hasłow sekcji Rysunek 25 na stronie 122 .

Przykłady

Przykład pliku konfiguracyjnego JAAS z jedną nazwaną konfiguracją, 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";
};
```

Rysunek 24. Przykładowy plik jaas.config

Przykład modułu logowania JAAS zakodowanego w taki sposób, aby otrzymywać wartości Nazwa użytkownika i Hasło udostępniane przez klienta 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;
}
```

Rysunek 25. Przykładowa metoda JAASLoginModule.Login()

Pojęcia dotyczące programowania klienta

Pojęcia opisane w tej sekcji ułatwiają zrozumienie klienta Java dla wersji 3.1 produktu MQTT protocol. Pojęcia uzupełniają dokumentację API dołączoną do pakietu com.ibm.micro.client.mqttv3.

Produkt com.ibm.micro.client.mqttv3 zawiera klasy udostępniające metody publiczne dla implementacji Java w wersji 3.1 produktu MQTT . Pakiet com.ibm.micro.client.mqttv3 wraz z pakietami towarzyszącymi, które implementują protokół dla produktu Java SE i ME, są dostarczane wraz z instalacją produktu IBM WebSphere MQ Telemetry.

Aby utworzyć i uruchomić klienta MQTT , należy skopiować lub zainstalować te pakiety na urządzeniu klienckim. Nie ma potrzeby instalowania odrębnego środowiska wykonawczego klienta.

Warunki licencjonowania dla klientów są powiązane z serwerem, z którym łączą się klienty.

Klient Java jest implementacją referencyjną w wersji 3.1 produktu MQTT protocol. Istnieje możliwość zaimplementowania własnych klientów w różnych językach odpowiednich dla różnych platform urządzeń. Szczegółowe informacje na ten temat można znaleźć w sekcji MQ Telemetry Transport format and protocol .

Dokumentacja interfejsu API klienta dla pakietu com.ibm.micro.client.mqttv3 nie zawiera żadnych założeń dotyczących serwera, z którym klient jest połączony. W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT. Zachowanie klienta może się nieznacznie różnić w przypadku połączenia z różnymi serwerami. W poniższych opisach opisano zachowanie klienta w połączeniu z usługą telemetryczną IBM WebSphere MQ (MQXR).

Klient przesyłania komunikatów MQTT dla produktu JavaScript i aplikacje WWW

Do niedawna programowanie aplikacji internetowych i tworzenie aplikacji do przesyłania wiadomości były odrębnymi dyscyplinami. Bez względu na to, gdzie znajdują się wcześniejsze doświadczenia, istnieją znaczące korzyści z używania produktu JavaScript i przesyłania komunikatów razem. Po zakodowaniu

aplikacji do przesyłania wiadomości jako aplikacji WWW można ją wciągać i uruchamiać w dowolnej aktualnej przeglądarce. W przypadku zmiany aplikacji, najnowsza wersja jest pobierana za każdym razem, gdy przeglądarka jest odświeżana. Przeglądarka wygląda również po zabezpieczeniach, a niezawodne przesyłanie komunikatów.

Jak korzystanie z aplikacji WWW umożliwia wdrożenie aplikacji

Jeśli użytkownik ma doświadczenie w tworzeniu i wdrażaniu tradycyjnych aplikacji przesyłania komunikatów (na przykład w produkcie IBM WebSphere MQ), może być zaznajomiony z następującym procesem wdrażania:

1. Administrator systemu instaluje lub osadza bibliotekę klienta.
2. Administrator systemu organizuje dystrybucję aplikacji do przesyłania komunikatów do użytkowników końcowych i zainstalowaną w ich systemach lokalnych.
3. Gdy kod ulegnie zmianie, administrator systemu powtarza poprzednie kroki (tak więc zarządzanie zmianami jest złożone).

Jeśli aplikacja przesyłania komunikatów została zakodowana jako aplikacja WWW, jest to proces wdrażania:

1. Administrator systemu służy do obsługi aplikacji WWW i biblioteki klienta pod adresem URL.
2. Przeglądarka użytkownika końcowego korzysta razem z aplikacji WWW i biblioteki klienta.
3. Gdy kod ulegnie zmianie, zaktualizowana wersja jest odbierana po odświeżeniu przeglądarki (tak więc zarządzanie zmianami jest proste).

Dlaczego warto korzystać z przesyłania wiadomości bezpośrednio z poziomu przeglądarki w aplikacjach WWW

Jeśli użytkownik ma doświadczenie w programowaniu aplikacji w programie JavaScript, może być zainteresowany poznaniem korzyści udostępnianych przez systemy przesyłania komunikatów, takie jak IBM WebSphere MQ:

- Jeśli komunikaty są wysyłane i odbierane za pośrednictwem systemu przesyłania komunikatów, ten system jest odpowiedzialny za zapewnienie dostarczania komunikatów.
- Ponieważ system przesyłania komunikatów jest wyszukiwawczy po dostarczeniu, aplikacja WWW może "zwolnić i zapomnieć". To bardzo upraszcza logikę programowania. Jeśli wiadomości są dostarczane dla Ciebie, Twój kod nie musi sprawdzić, czy tam dotarły. Aplikacja nie musi już obsługiwać potwierdzenia odbioru ani zapisywać niedostarczanych wiadomości i ponawia je później.
- Systemy przesyłania komunikatów udostępniają komunikaty sterowane zdarzeniami. Aplikacja kliencka nie musi już wysyłać żądania, a następnie stale odpytywać w celu uzyskania odpowiedzi. Zamiast tego serwer przesyłania komunikatów wysyła komunikat do aplikacji klienckiej, gdy wystąpi interesujące zdarzenie. Oznacza to również, że aplikacja kliencka jest powiadamiana natychmiast po wystąpieniu zdarzenia, zamiast czekać do następnego odpytywania serwera przez aplikację.
- Przesyłanie komunikatów sterowane zdarzeniami powoduje również masowe zmniejszenie obciążenia urządzenia udostępniającego aplikację kliencką użytkownika, ruch sieciowy między przeglądarką a serwerem przesyłania komunikatów oraz obciążenie serwera przesyłania komunikatów. To się coraz bardziej liczy, coraz więcej systemów działa na urządzeniach mobilnych, a łącząc się między sieciami bezprzewodowymi.

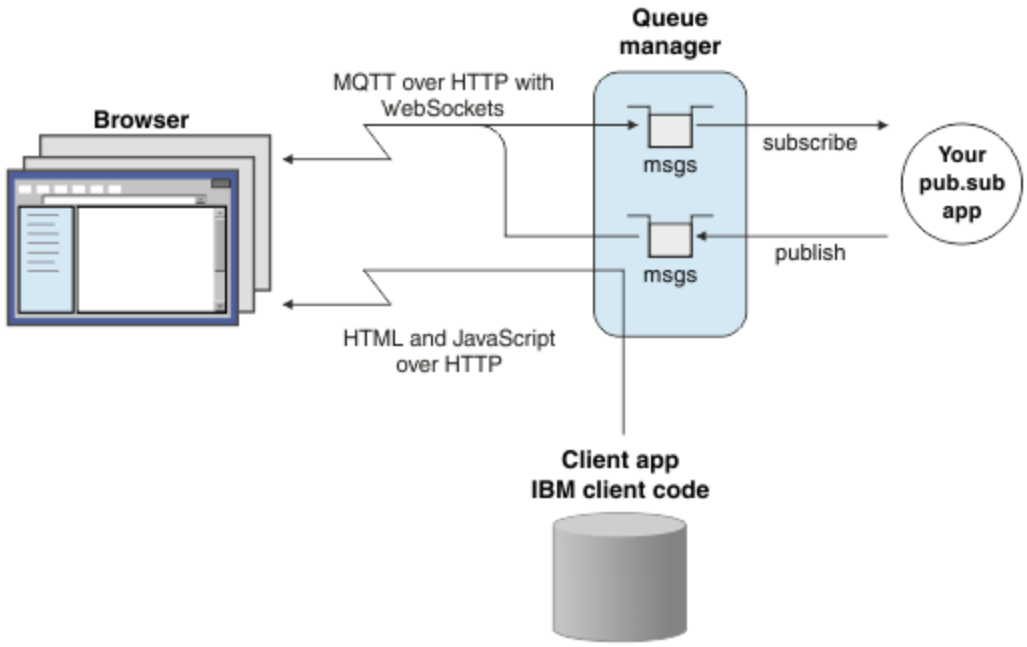
Wzajemne relacje elementów

Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zawiera bibliotekę klienta oraz przykładową aplikację WWW, która korzysta z biblioteki. Należy zakodować własną aplikację WWW, która korzysta z biblioteki. Aplikacja WWW i biblioteka klienta są następnie udostępniane pod wybranym adresem URL, na przykład przez menedżer kolejek produktu MQ (tak jak na poniższym diagramie) lub przez serwer aplikacji. Przeglądarka pobiera w aplikacji WWW i bibliotece klienta, a aplikacja WWW

następnie używa przeglądarki w celu połączenia się z serwerem MQTT, na przykład IBM WebSphere MQ Telemetry lub IBM MessageSight, oraz do wymiany komunikatów z serwerem MQTT.

Są to przepływy:

1. Każda instancja przeglądarki odświeża połączenie z adresem URL, na którym jest dostępna aplikacja WWW, a na przeglądarce załadowana jest aktualna wersja aplikacji WWW i biblioteki klienta.
2. Aplikacja WWW łączy się z menedżerem kolejek przy użyciu protokołu MQTT na serwerze WebSocket protocoli subskrybuje temat interesującego go tematu.
3. Menedżer kolejek używa tego samego połączenia do wysyłania komunikatów, które są zgodne z subskrypcją, z powrotem do aplikacji WWW.



Rysunek 26. Używanie produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z przesyłaniem komunikatów w trybie publikowania i subskrypcji

Aplikacja WWW zawiera logikę aplikacji, a także adres URL serwera MQTT. Po otwarciu w przeglądarce aplikacja nawiązuje połączenie z serwerem MQTT, tworzy wymagane subskrypcje, a następnie czeka na otrzymywanie alertów sterowanych zdarzeniami i działanie na ich podstawie.

Aplikacja WWW łączy się za pomocą protokołu MQTT jako protokołu transportowego działającego w produkcie WebSockets. Większość nowoczesnych przeglądarek może nawiązać połączenia WebSockets . Za pomocą programu WebSocketsaplikacja WWW może przekazywać komunikaty za pośrednictwem firewalli akceptanych HTTP i WebSocket protocol, a także wysyłać pakiety danych (nazywane "ramkami"), tak jak przy użyciu protokołu TCP w okresie IP.

Gdy komunikat wysyłany przez aplikację WWW dociera do serwera MQTT, aplikacja po stronie serwera po prostu widzi ją jako komunikat. Nie ma informacji o tym, że komunikat pochodzi z przeglądarki.

Administrowanie i sterowanie serwerem MQTT

Serwer MQTT obsługuje złożoność przesyłania komunikatów po stronie serwera. Zapewnia ona dostarczanie wiadomości, które otrzymuje z aplikacji WWW, a także udostępnia aplikację publikowania i subskrypcji, która odpowiada na aplikację WWW. W przypadku dowolnego serwera MQTT należy wykonać następujące kroki:

- Utwórz serwer.
- Wybierz port.

- Zdefiniuj nowy kanał MQTT.
- Skonfiguruj aplikację WWW klienta, aby połączyć się z wybranym portem w nowym kanale MQTT.

Konieczne jest również obsługenie pliku wykonywalnego aplikacji WWW JavaScript w przeglądarce. Jeśli używany jest produkt IBM WebSphere MQ Telemetry, domyślnie serwer MQTT wykonuje tę operację za pomocą tego samego kanału MQTT, który jest używany przez aplikację WWW do nawiązywania połączenia z serwerem MQTT. Jeśli próbujesz wypróbować MQTT, może to pomóc w szybkim uruchamianiu i szybkim uruchomieniu. W przypadku zastosowań produkcyjnych, w szczególności w środowiskach o wysokiej przepustowości, użytkownik może preferować obsługę kodu wykonywalnego aplikacji WWW JavaScript w osobnym kanale, korzystając z dedykowanego serwera aplikacji, takiego jak WebSphere Application Server.

Uwaga: Ponieważ jest on zaprojektowany dla środowisk o wysokiej przepustowości, produkt IBM MessageSight oczekuje, że zostanie to wykonalne.

Jeśli na przykład używany jest produkt IBM WebSphere MQ Telemetry, należy użyć kreatora IBM WebSphere MQ Explorer **Nowy kanał telemetryczny** w celu wykonania następujących kroków:

1. Utwórz serwer.
2. Wybierz port (domyślnie 1883).
3. Zdefiniuj nowy kanał MQTT.
4. Skonfiguruj aplikację WWW klienta, aby połączyć się z wybranym portem w nowym kanale MQTT.

Plik wykonywalny aplikacji WWW JavaScript jest (opcjonalnie) obsługiwany również przez menedżer kolejek w tym samym kanale. Aby to zrobić, menedżer kolejek musi obsługiwać zarówno protokół MQTT, jak i HTTP. Jeśli menedżer kolejek jest już skonfigurowany dla produktu MQTT, można użyć narzędzia wiersza komend MQSC, aby zmienić protokół w definicji kanału w celu obsługi zarówno protokołu MQTT, jak i protokołu HTTP. Patrz ALTER CHANNEL.

Aplikacja WWW i biblioteka klienta produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript są przechowywane na dysku w strukturze zdefiniowanej przez serwer aplikacji lub menedżer kolejek. Jeśli używany jest produkt IBM WebSphere MQ Telemetry, aplikacja WWW i biblioteka klienta są przechowywane w następującej strukturze katalogów:

```
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)
```

Przykładowa aplikacja WWW i biblioteka klienta są przechowywane w katalogu *MQINSTALL*/mqxr/SDK/WebContent . Materiały znajdujące się w tym katalogu są obsługiwane przez wszystkie menedżery kolejek. Jeśli użytkownik nie chce, aby użytkownicy mogli wyświetlać i używać całego tego materiału, należy utworzyć własną wersję aplikacji. Aby udostępnić tę aplikację lub własną aplikację zastępczą, dostępną w konkretnych menedżerach kolejek, należy umieścić tę aplikację w katalogu *MQINSTALL*/qmgrs/*qmgr_name*/mqxr/WebContent . Aby wybrać aplikację i powiązane klasy produktu JavaScript w celu obsługi adresu URL, menedżer kolejek będzie wyglądać najpierw w swoim katalogu WebContent , a następnie w globalnym katalogu WebContent . W poprzednim przykładowym drzewie katalogów menedżer kolejek obsługuje *aplikacja_klienta_klienta* i globalną kopię klas JavaScript .

Aby zatrzymać menedżer kolejek udostępniający pliki wykonywalne aplikacji WWW lub zmodyfikować miejsce, w którym menedżer kolejek szuka plików wykonywalnych, należy skonfigurować właściwość **webcontentpath** i dodać ją do pliku `mqxr.properties` . Patrz: właściwości MQXR.

Pojęcia pokrewne

“Jak programować aplikacje przesyłania komunikatów w produkcie JavaScript” na stronie 126

Zadania pokrewne

“Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets” na stronie 81

Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

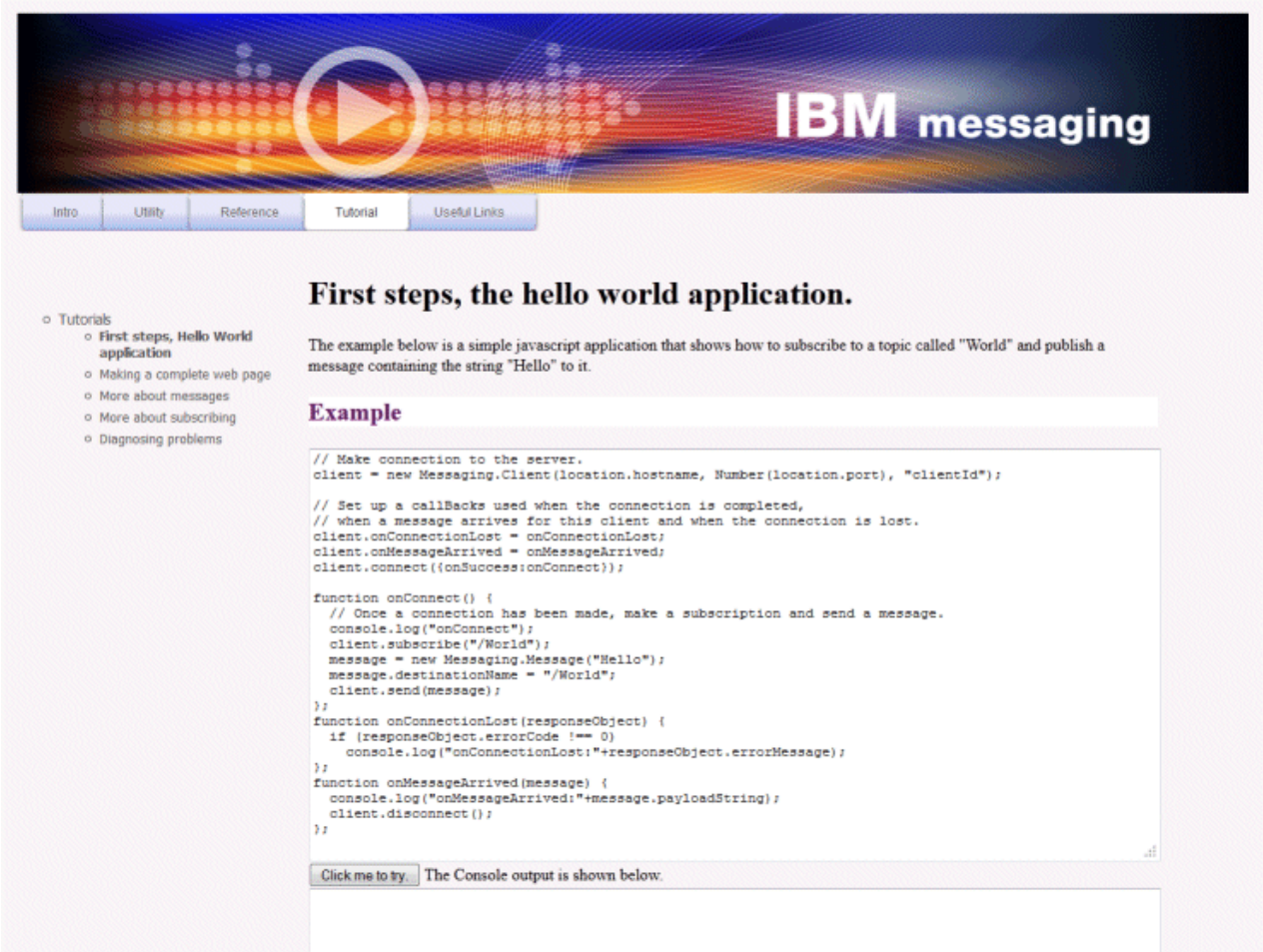
“Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript” na stronie 25

Można rozpocząć od Klient przesyłania komunikatów MQTT dla produktu JavaScript , wyświetlając przykładową stronę główną klienta przesyłania komunikatów i przeglądając zasoby, do których odsyłacze te są dostępne. Aby wyświetlić tę stronę główną, należy skonfigurować serwer MQTT w taki sposób, aby akceptować połączenia z serwera Przykładowe strony JavaScript klienta przesyłania komunikatów MQTT, a następnie należy wpisać adres URL skonfigurowany na serwerze w przeglądarce WWW. Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zostanie uruchomiony automatycznie na urządzeniu, a zostanie wyświetlona przykładowa strona główna klienta przesyłania komunikatów. Ta strona zawiera odsyłacze do programów narzędziowych, dokumentacji interfejsu programistycznego, kursu i innych przydatnych informacji.

Jak programować aplikacje przesyłania komunikatów w produkcie JavaScript

Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zawiera kurs demonstrujący sposób tworzenia prostej aplikacji WWW publikowania i subskrypcji. Eksplorując kod aplikacji "Pierwsze kroki, Hello world", można uzyskać podstawową wiedzę na temat mechaniki programowej aplikacji webowych do przesyłania wiadomości.

Jeśli dotychczasu dotychczasu doświadczenia zajmował się głównie tworzeniem i wdrażaniem tradycyjnych aplikacji przesyłania komunikatów, pomocne może być również znalezienie sekcji “Wskazówki dotyczące kodowania produktu JavaScript” na stronie 127 . Jeśli jest to doświadczony programista produktu JavaScript , który jest nowością w przesyłaniu komunikatów, w sekcji “Podstawy przesyłania komunikatów” na stronie 129 można znaleźć krótkie wprowadzenie do kluczowych pojęć związanych z przesyłaniem komunikatów.



Wskazówki dotyczące kodowania produktu JavaScript

Jeśli używane są aplikacje przesyłania komunikatów, ale nowe aplikacje WWW, pomocne mogą być następujące wskazówki:

Zawijanie kodu dla każdego zdarzenia w wywołaniu zwrotnym onSuccess

Podczas tworzenia kodu aplikacji przesyłania komunikatów należy zakodować następujące zdarzenia w następującej kolejności:

1. connect
2. subskrybować
3. publikować
4. odebranie komunikatu

Interfejs API produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript jest w pełni asynchroniczny, co oznacza, że wątek aplikacji nie blokuje podczas oczekiwania na wywołania, takie jak połączenie lub subskrypcja, które ma zostać zastosowane. Zamiast tego wywołania te sygnalią ich zakończenie, wywołując wywołanie zwrotne `onSuccess` lub `onFailure` . Aby mieć pewność, że każde zdarzenie zostało zakończone przed wyzwoleniu następnego zdarzenia, należy zawinować kod dla każdego zdarzenia w wywołaniu zwrotnym `onSuccess` . Na przykład aplikacja JavaScript może zwrócić się z wywołania połączenia przed utworzeniem połączenia. Aby mieć pewność, że połączenie miało miejsce przed zasubskrybowanie, należy umieścić kod subskrypcji w wywołaniu zwrotnym `onSuccess` dla połączenia.

Kod aplikacji "Pierwsze kroki, Hello world" korzysta z tego podejścia.

Osadzanie kodu aplikacji w języku znaczników HTML

Oto przykładowa strona JavaScript :

Poniżej znajduje się źródło poprzedniej strony, w celu pokazania, w jaki sposób kod aplikacji jest osadzony w obrębie znacznika HTML-up:

```
<!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>
```

Podstawy przesyłania komunikatów

Poniżej znajduje się kilka informacji o przesyłaniu komunikatów w tle dla programistów aplikacji WWW, które są nowe w przesyłaniu komunikatów:

Asynchroniczne i odpalające przesyłanie komunikatów.
Protokół MQTT obsługuje przesyłanie gwarantowanych dostaw i przeciwpożarowych oraz-i-o-zapominanie. W protokole dostarczanie komunikatów jest asynchroniczne: aplikacja przekazuje komunikat do interfejsu API klienta i nie podejmuje dalszych działań w celu zapewnienia, że komunikat jest dostarczany. Jest to metoda znana jako *fire-and-forget*. Gdy odpowiedź jest dostępna, jest ona automatycznie wysyłana do aplikacji.

Dostarczanie asynchroniczne zwalnia aplikację z dowolnego połączenia z serwerem i oczekuje na komunikaty. Model interakcji jest podobny do poczty elektronicznej, ale zoptymalizowany pod kątem programowania aplikacji.

Patrz także sekcja "MQTT protocol" (protokół MQTT) w produkcie “Wprowadzenie do produktu MQTT” na stronie 5 .

Przegląd przesyłania komunikatów w trybie publikowania i subskrypcji.
Dostawca informacji jest nazywany *publikatorem*. Publikator dostarcza informacji o temacie, bez potrzeby poznawania tych informacji o aplikacjach, które są zainteresowane tą informacją. Publikator wybiera *temat*, który jest kontenerem dla komunikatów w konkretnym temacie. Następnie publikator generuje każdy fragment informacji dla tego podmiotu jako komunikat, nazywany *publikacją* i publikuje go w powiązanym temacie.

Konsument informacji jest nazywany *subskrybentem*. Subskrybent tworzy *subskrypcję* dla tematu, który jest zainteresowany. Po wysłaniu nowego komunikatu do tematu komunikat jest przekazywany do wszystkich subskrybentów tematu. Subskrybenci mogą wykonywać wiele subskrypcji i mogą otrzymywać informacje od wielu różnych publikatorów.

Patrz także Wprowadzenie do przesyłania komunikatów w trybie publikowania/subskrypcji produktu IBM WebSphere MQ

Sposób dopasowania subskrypcji i tematów.

Jeśli jako serwer MQTT używany jest produkt IBM WebSphere MQ , należy zrozumieć, w jaki sposób produkt IBM WebSphere MQ określa tematy. W produkcie IBM WebSphere MQpublikator tworzy komunikat i publikuje go za pomocą łańcucha tematu, który najlepiej pasuje do tematu publikacji. W celu otrzymywania publikacji subskrybent tworzy subskrypcję przy użyciu łańcucha tematu zgodnego z wzorcem w celu wybrania tematów publikacji. Menedżer kolejek dostarcza publikacje do subskrybentów, którzy mają subskrypcje zgodne z tematem publikacji, a także są uprawnieni do otrzymywania publikacji.

Zwykle tematy są zorganizowane hierarchicznie, w drzewa tematów, przy użyciu znaku/, aby utworzyć podtematy w łańcuchu tematu. Tematy są węzłami w drzewie tematów. Tematy mogą być węzłów-liści bez dalszych podtematów lub węzłów pośrednich z podtematami. Subskrybenci mogą używać znaków wieloznacznych w celu zasubskrybowania więcej niż jednego tematu naraz. Na przykład subskrypcja programu `/sport/tennis` pobiera tylko komunikaty publikowane w podtemacie tenisa, natomiast subskrypcja programu `/sport/#` pobiera komunikaty publikowane w dowolnym podtemacie produktu `/sport`.

Patrz także: Tematy, Drzewa tematówi Programy wieloznaczne.

Pojęcia pokrewne

“Klient przesyłania komunikatów MQTT dla produktu JavaScript i aplikacje WWW” na stronie 122

Zadania pokrewne

“Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets” na stronie 81

Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

“Pierwsze kroki z interfejsem typu Klient przesyłania komunikatów MQTT dla produktu JavaScript” na stronie 25

Można rozpocząć od Klient przesyłania komunikatów MQTT dla produktu JavaScript , wyświetlając przykładową stronę główną klienta przesyłania komunikatów i przeglądając zasoby, do których odsyłacze te są dostępne. Aby wyświetlić tę stronę główną, należy skonfigurować serwer MQTT w taki sposób, aby akceptować połączenia z serwera Przykładowe strony JavaScript klienta przesyłania komunikatów MQTT, a następnie należy wpisać adres URL skonfigurowany na serwerze w przeglądarce WWW. Produkt Klient przesyłania komunikatów MQTT dla produktu JavaScript zostanie uruchomiony automatycznie na urządzeniu, a zostanie wyświetlona przykładowa strona główna klienta przesyłania komunikatów. Ta strona zawiera odsyłacze do programów narzędziowych, dokumentacji interfejsu programistycznego, kursu i innych przydatnych informacji.

Procedury zwrotne i synchronizacja w aplikacjach klienckich MQTT

W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , na tyle, na ile mogą, z opóźnień w przesyłaniu komunikatów do i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wywołania zwrotne

Interfejs `MqttCallback` ma trzy metody wywołania zwrotnego. Patrz przykładowa implementacja w pliku Callback.java.

connectionLost(java.lang.Throwable cause)

Program `connectionLost` jest wywoływany po wystąpieniu błędu komunikacyjnego w celu usunięcia połączenia. Jest ona również wywoływana, jeśli serwer usunie połączenie w wyniku błędu na serwerze po nawiązaniu połączenia. Błędy serwera są rejestrowane w dzienniku błędów menedżera kolejek. Serwer usuwa połączenie z klientem, a klient wywołuje `MqttCallback.connectionLost`.

Jedyne błędy zdalne zgłaszane jako wyjątki w tym samym wątku, co aplikacja kliencka, są wyjątkami od produktu `MqttClient.connect`. Błędy wykryte przez serwer po nawiązaniu połączenia są zgłaszane z powrotem do metody wywołania zwrotnego `MqttCallback.connectionLost` jako `throwables`.

Typowe błędy serwera, których wynikiem jest `connectionLost` , to błędy autoryzacji. Na przykład serwer telemetryczny podejmuje próbę opublikowania w temacie w imieniu klienta, który nie ma uprawnień do publikowania w tym temacie. Wszystkie wyniki w kodzie warunku `MQCC_FAIL` zwracane do serwera telemetrycznego mogą spowodować usunięcie połączenia.

deliveryComplete(MqttDeliveryToken token)

Program `deliveryComplete` jest wywoływany przez klienta MQTT w celu przekazania tokenu dostarczania do aplikacji klienckiej. Patrz sekcja “Znaczniki dostawy” na stronie 135. Za pomocą znacznika dostarczania wywołanie zwrotne może uzyskać dostęp do opublikowanego komunikatu przy użyciu metody `token.getMessage`.

Gdy wywołanie zwrotne aplikacji zwraca sterowanie do klienta MQTT po wywołaniu przez metodę `deliveryComplete` , dostarczenie jest zakończone. Do czasu zakończenia dostarczania komunikaty z funkcją QoS 1 lub 2 są zachowywane przez klasę trwałości.

Wywołanie funkcji `deliveryComplete` jest punktem synchronizacji między aplikacją a klasą trwałości. Metoda `deliveryComplete` nigdy nie jest wywoływana dwukrotnie dla tego samego komunikatu.

Gdy wywołanie zwrotne aplikacji zwraca wartość z `deliveryComplete` do klienta MQTT , klient wywołuje `MqttClientPersistence.remove` dla komunikatów z funkcją QoS 1 lub 2. Program `MqttClientPersistence.remove` usuwa lokalnie zapisaną kopię opublikowanego komunikatu.

Z perspektywy przetwarzania transakcji wywołanie funkcji `deliveryComplete` jest transakcją jednofazową, która zatwierdza dostawę. Jeśli przetwarzanie nie powiedzie się podczas wywołania zwrotnego, po ponownym uruchomieniu klienta `MqttClientPersistence.remove` zostanie ponownie wywołana w celu usunięcia lokalnej kopii opublikowanego komunikatu. Wywołanie zwrotne nie jest ponownie wywoływane. Jeśli do przechowywania dziennika dostarczonych komunikatów używana jest procedura zwrotna, nie można zsynchronizować dziennika z klientem MQTT . Jeśli dziennik ma być niezawodnie przechowywany, należy zaktualizować dziennik w klasie `MqttClientPersistence` .

Znacznik dostawy i komunikat są przywoływane przez główny wątek aplikacji i klienta MQTT . Klient MQTT wyrejestrowuje obiekt `MqttMessage` po zakończeniu dostarczania, a także obiekt znacznika dostarczania, gdy klient rozłącza się. Obiekt `MqttMessage` może być czyszczony po zakończeniu dostarczania, jeśli aplikacja kliencka wyłuskuje ją. Znacznik dostarczania może być czyszczony po rozłączeniu sesji.

Atrybuty `MqttDeliveryToken` i `MqttMessage` można pobrać po opublikowaniu komunikatu. Jeśli po opublikowaniu komunikatu zostanie podjęta próba ustawienia atrybutów `MqttMessage` , wynik nie zostanie zdefiniowany.

Klient MQTT kontynuuje przetwarzanie potwierdzeń dostarczenia, jeśli klient ponownie nawiąże połączenie z poprzednią sesją przy użyciu tego samego `ClientIdentifier`. Patrz “Czyste sesje” na stronie 133. Aplikacja kliencka MQTT musi ustawić `MqttClient.CleanSession` na `false` dla poprzedniej sesji, a następnie ustawić ją na `false` w nowej sesji. Klient MQTT tworzy nowe znaczniki dostarczania i obiekty komunikatów w nowej sesji dla oczekujących dostaw. Odtwarza obiekty za pomocą klasy `MqttClientPersistence` . Jeśli klient aplikacji nadal ma odwołania do starych tokenów dostawy i komunikatów, wyłuskuj je. Wywołanie zwrotne aplikacji jest wywoływane w nowej sesji dla wszystkich dostaw zainicjowanych w poprzedniej sesji i zakończonych w tej sesji.

Wywołanie zwrotne aplikacji jest wywoływane po nawiązaniu połączenia przez klient aplikacji po zakończeniu dostarczania oczekującego. Zanim klient aplikacji połączy się, będzie mógł pobrać oczekujące dostawy za pomocą metody `MqttClient.getPendingDeliveryTokens` .

Zauważ, że aplikacja kliencka pierwotnie utworzyła opublikowany obiekt komunikatu i jego tablicę bajtów ładunku. Klient MQTT odwołuje się do tych obiektów. Obiekt komunikatu zwrócony przez znacznik dostarczania w metodzie `token.getMessage` nie musi być tym samym obiektem komunikatu tworzonym przez klienta. Jeśli nowa instancja klienta MQTT ponownie odtwarza znacznik dostarczania, klasa `MqttClientPersistence` odtwarza obiekt `MqttMessage` . For consistency `token.getMessage` returns `null` if `token.isCompleted` is `true`, regardless of whether the message object was created by the application client or the `MqttClientPersistence` class.

`messageArrived(MqttTopic topic, MqttMessage message)`

Program `messageArrived` jest wywoływany po nadejściu publikacji dla klienta, który jest zgodny z tematem subskrypcji. `temat` to temat publikowania, a nie filtr subskrypcji. Dwa mogą być różne, jeśli filtr zawiera znaki wieloznaczne.

Jeśli temat pasuje do wielu subskrypcji utworzonych przez klienta, klient otrzymuje wiele kopii tej publikacji. Jeśli klient publikuje w temacie, do którego również subskrybuje, otrzymuje kopię własnej publikacji.

Jeśli komunikat jest wysyłany z usługą QoS 1 lub 2, komunikat jest zapisywany przez klasę `MqttClientPersistence` przed wywołaniami `messageArrived`klienta MQTT . `messageArrived` zachowuje się jak `deliveryComplete`: jest wywoływana tylko raz dla publikacji, a lokalna kopia publikacji jest usuwana przez program `MqttClientPersistence.remove` , gdy program `messageArrived` powróci do klienta MQTT . Klient MQTT usuwa odwołania do tematu i komunikatu, gdy program `messageArrived` powróci do klienta MQTT . Obiekty tematów i komunikatów są czyszczeniem pamięci, jeśli klient aplikacji nie ma odwołania do odwołania do obiektów.

Synchronizacja aplikacji zwrotnych, wielowątkowych i aplikacji klienckich

Klient MQTT wywołuje metodę wywołania zwrotnego w osobnym wątku do głównego wątku aplikacji. Aplikacja kliencka nie tworzy wątku dla wywołania zwrotnego, jest on tworzony przez klienta MQTT .

Klient MQTT synchronizuje metody wywołania zwrotnego. W danej chwili działa tylko jedna instancja metody wywołania zwrotnego. Synchronizacja ułatwia aktualizację obiektu, który utalentowany jest, które publikacje zostały dostarczone. Jedna instancja serwera `MqttCallback.deliveryComplete` jest uruchamiana jednocześnie, a więc można bezpiecznie aktualizować ją bez dalszej synchronizacji. Jest to również przypadek, że tylko jedna publikacja dociera w danym momencie. Kod w metodzie `messageArrived` może aktualizować obiekt bez synchronizacji. Jeśli odwołujesz się do "tally" lub obiektu, który jest aktualizowany, w innym wątku, zsynchronizuj ten sam obiekt lub obiekt.

Znacznik dostarczania udostępnia mechanizm synchronizacji między głównym wątkiem aplikacji a publikacją. Metoda `token.waitForCompletion` oczekuje na zakończenie dostarczania konkretnej publikacji lub do momentu utraty ważności opcjonalnego limitu czasu. Produkt `token.waitForCompletion` może być używany w kilku prostych sposobach przetwarzania jednej publikacji w danym momencie:

1. Aby wstrzymać klient aplikacji do czasu dostarczenia publikacji, należy zapoznać się z publikacją PubSync.java.
2. Aby przeprowadzić synchronizację z metodą `MqttCallback.deliveryComplete` . Tylko w przypadku, gdy program `MqttCallback.deliveryComplete` powróci do programu MQTT , zostanie wznowiony `token.waitForCompletion` . Za pomocą tego mechanizmu można synchronizować działający kod w produkcie `MqttCallback.deliveryComplete` , zanim kod zostanie uruchomiony w głównym wątku aplikacji.

Co, jeśli chcesz opublikować bez czekania na każdą publikację, która ma być dostarczona, ale chcesz potwierdzić, kiedy wszystkie publikacje zostały dostarczone? W przypadku publikowania w pojedynczym wątku ostatnia publikacja, która ma zostać wysłana, jest również ostatnią dostarczoną.

Synchronizacja żądań wysłanych do serwera

Tabela 4. Zachowanie synchronizacji metod, których wynikiem są żądania do serwera.

Ta tabela zawiera listę metod w kliencie Java produktu MQTT , które wysyłają żądanie do serwera. Dla każdej metody w tabeli opisano warunki, w jakich metoda oczekuje lub zwraca, oraz od czasu oczekiwania metody.

Metoda	Synchronizacja	Limit czasu
`MqttClient.Connect`	Oczekuje na nawiązanie połączenia z serwerem.	Domyślnie 30 sekund lub wartość ustawiona przez parametr.
`MqttClient.Disconnect`	Oczekuje, że klient MQTT zakończy pracę, którą musi wykonać, a sesja TCP/IP zostanie rozłączona.	Domyślnie 30 sekund lub wartość ustawiona przez parametr.
`MqttClient.Subscribe`	Oczekuje na zakończenie żądania subskrypcji.	Domyślnie 30 sekund lub wartość ustawiona przez parametr.
`MqttClient.UnSubscribe`	Oczekuje na zakończenie żądania anulowania subskrypcji.	Domyślnie 30 sekund lub wartość ustawiona przez parametr.
`MqttClient.Publish`	Powraca natychmiast do wątku aplikacji po przekazaniu żądania do klienta MQTT .	Brak.
`MqttDeliveryToken.waitForCompletion`	Oczekuje na zwrócenie znacznika dostarczania.	Nieokreślone domyślnie lub jako ustawione przez parametr.

Czyste sesje

Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są wykorzystywane w celu zapewnienia "co najmniej raz" i "dokładnie raz" dostarczenia, oraz "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT. Użytkownik może wybrać opcję uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Po połączeniu aplikacji klienckiej MQTT za pomocą metody `MqttClient.connect` klient identyfikuje połączenie przy użyciu identyfikatora klienta i adresu serwera. Serwer sprawdza, czy informacje o sesji zostały zeskładowane z poprzedniego połączenia z serwerem. Jeśli poprzednia sesja nadal istnieje, a następnie `cleanSession=true`, to poprzednie informacje o sesji na kliencie i serwerze zostaną wyczyszczone. Jeśli `cleanSession=false` poprzednia sesja została wznowiona. Jeśli żadna wcześniejsza sesja nie istnieje, zostanie uruchomiona nowa sesja.

Uwaga: Administrator produktu WebSphere MQ może wymusić zamknięcie otwartej sesji i usunięcie wszystkich informacji o sesji. Jeśli klient ponownie otworzy sesję z programem `cleanSession=false`, uruchamiana jest nowa sesja.

Publikacje

Jeśli przed nawiązaniem połączenia z klientem zostanie użyta wartość domyślna `MqttConnectOptions`lub zostanie ustawiona wartość `MqttConnectOptions.cleanSession` na

`true` , wszystkie oczekujące dostawy publikacji dla klienta zostaną usunięte po nawiązaniu połączenia przez klienta.

Ustawienie sesji czyszczenia nie ma wpływu na publikacje wysłane z produktem `QoS=0`. W przypadku produktów `QoS=1` i `QoS=2`użycie produktu `cleanSession=true` może spowodować utratę publikacji.

Subskrypcje

Jeśli przed nawiązaniem połączenia z klientem zostanie użyta wartość domyślna `MqttConnectOptions`lub zostanie ustawiona wartość `MqttConnectOptions.cleanSession` na `true` , wówczas wszystkie stare subskrypcje klienta zostaną usunięte po nawiązaniu połączenia przez klienta. Wszystkie nowe subskrypcje dokonane przez klienta podczas sesji zostaną usunięte po rozłączeniu.

Jeśli serwer `MqttConnectOptions.cleanSession` zostanie ustawiony na `false` przed nawiązaniem połączenia, wszystkie subskrypcje tworzone przez klienta zostaną dodane do wszystkich subskrypcji, które istniały dla klienta przed jego połączeniem. Wszystkie subskrypcje pozostaną aktywne po rozłączeniu połączenia z klientem.

Atrybut `cleanSession` można także rozpatrywać jako atrybut modalny, jeśli chodzi o jego wpływ na subskrypcje. W domyślnym trybie atrybutu `cleanSession=true` klient tworzy subskrypcje i odbiera publikacje tylko w zasięgu sesji. W alternatywnym trybie atrybutu `cleanSession=false` subskrypcje są trwałe. Można nawiązywać połączenia z klientem i je rozłączać, a jego subskrypcje pozostają aktywne. Po ponownym nawiązaniu połączenia z klientem odbiera on wszystkie niedostarczone publikacje. Po nawiązaniu połączenia klient może modyfikować zestaw subskrypcji aktywnych na jego potrzeby.

Przed nawiązaniem połączenia należy ustawić tryb `cleanSession` . Tryb ten trwa cały czas trwania sesji. Aby zmienić ustawienie trybu, należy rozłączyć połączenie z klientem, a następnie ponownie je nawiązać. Jeśli tryby zostaną zmienione z `cleanSession=false` na `cleanSession=true`, wszystkie poprzednie subskrypcje klienta oraz wszystkie publikacje, które nie zostały odebrane, zostaną usunięte.

Identyfikator klienta

Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Identyfikator klienta musi być unikalny w obrębie wszystkich klientów nawiązujących połączenie z serwerem. Ponadto nie może on być taki sam jak nazwa menedżera kolejek na serwerze. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Identyfikator klienta jest używany w administrowaniu systemem MQTT . Z potencjalnie setkami tysięcy klientów do administrowania, konieczne jest szybkie zidentyfikowanie konkretnego klienta. Załóżmy na przykład, że urządzenie nie działa, a użytkownik jest powiadamiany, być może przez klienta dzwoniący do stanowiska pomocy. W jaki sposób klient identyfikuje urządzenie i w jaki sposób korelować tę identyfikację z serwerem, który jest zwykle połączony z klientem? Czy konieczne jest zapoznanie się z bazą danych, która odwzorowuje każde urządzenie na identyfikator klienta i na serwer? Czy nazwa urządzenia identyfikuje serwer, do którego jest przyłączony? Podczas przeglądania za pośrednictwem połączeń klienckich MQTT każde połączenie jest oznaczone identyfikatorem klienta. Czy chcesz wyszukać tabelę w celu odwzorowania identyfikatora klienta na urządzenie fizyczne?

Czy identyfikator klienta identyfikuje określone urządzenie, użytkownika lub aplikację działającą na kliencie? Czy jeśli klient zastąpi wadliwe urządzenie nowym urządzeniem, to czy nowe urządzenie ma ten sam identyfikator co stare urządzenie? Czy przydzielić nowy identyfikator? Jeśli zmienisz urządzenie fizyczne, ale zachowaj ten sam identyfikator, pozostałe publikacje i aktywne subskrypcje zostaną automatycznie przesłane do nowego urządzenia.

W jaki sposób identyfikatory klientów są unikalne? Podobnie jak w przypadku systemu generowania unikalnych identyfikatorów, należy mieć niezawodny proces ustawiania identyfikatora na kliencie. Być może urządzenie klienckie to "black-box", bez interfejsu użytkownika. Czy wytwarza się urządzenie o identyfikatorze klienta-takie jak przy użyciu jego adresu MAC? A może jest to proces instalacji i konfiguracji oprogramowania, który konfiguruje urządzenie przed jego aktywowaniem?

Identyfikator klienta można utworzyć na podstawie adresu MAC urządzenia 48-bitowego, aby identyfikator był krótki i unikalny. Jeśli wielkość transmisji nie jest newralgicznym problemem, można użyć pozostałych 17 bajtów, aby ułatwić administrowanie adresem.

Znaczniki dostawy

Gdy klient publikuje w temacie nowy znacznik dostarczania, zostanie utworzony. Za pomocą znacznika dostarczania można monitorować dostarczanie publikacji lub blokować aplikację kliencką do czasu zakończenia dostawy.

Token jest obiektem `MqttDeliveryToken` . Jest on tworzony przez wywołanie metody MqttTopic.publish () i jest zachowywany przez klienta MQTT do momentu, gdy sesja klienta zostanie rozłączona i dostarczenie zostanie zakończone.

Normalny sposób użycia tokenu polega na sprawdzeniu, czy dostawa jest kompletna. Zablokuj aplikację kliencką do czasu zakończenia dostarczania za pomocą zwróconego znacznika do wywołania `token.waitForCompletion`. Alternatywnie można podać procedurę obsługi produktu `MqttCallBack` . Po otrzymaniu przez klienta MQTT wszystkich potwierdzeń, których oczekuje w ramach dostarczania publikacji, wywołuje on `MqttCallBack.deliveryComplete` przekazując znacznik dostarczania jako parametr.

Dopóki dostawa nie zostanie zakończona, można sprawdzić publikację za pomocą zwróconego znacznika dostarczania, wywołując komendę `token.getMessage`.

Zrealizowane dostawy

Zakończenie dostaw jest asynchroniczne i zależy od jakości usługi powiązanej z publikacją.

Najwyżej raz

`QoS=0`
Dostarczanie jest kompletne natychmiast po powrocie z produktu `MqttTopic.publish`. `MqttCallback.deliveryComplete` jest wywoływana natychmiast.

Co najmniej raz

`QoS=1`
Dostarczanie jest kompletne, gdy potwierdzenie publikacji zostało odebrane z menedżera kolejek. `MqttCallback.deliveryComplete` jest wywoływana po odebraniu potwierdzenia. Komunikat może zostać dostarczony więcej niż jeden raz przed wywołaniem programu `MqttCallback.deliveryComplete` , jeśli komunikacja jest powolna lub niewiarygodna.

Dokładnie jeden raz

`QoS=2`
Dostarczanie jest kompletne, gdy klient otrzymuje komunikat o zakończeniu, że publikacja została opublikowana w subskrybentach. `MqttCallback.deliveryComplete` jest wywoływana natychmiast po odebraniu komunikatu publikacji. Nie czeka na komunikat o zakończeniu.

W rzadkich przypadkach aplikacja kliencka może nie wrócić do klienta MQTT z `MqttCallback.deliveryComplete` normalnie. Wiadomo, że dostarczenie zostało zakończone, ponieważ wywołano program `MqttCallback.deliveryComplete` . Jeśli klient zrestartuje tę samą sesję, program `MqttCallback.deliveryComplete` nie zostanie wywołany ponownie.

Dostawy niekompletne

Jeśli dostawa nie zostanie zakończona po rozłączeniu sesji klienta, można ponownie nawiązać połączenie z klientem i zakończyć dostarczanie. Dostarczanie komunikatu można zakończyć tylko wtedy, gdy komunikat został opublikowany w sesji z atrybutem `MqttConnectionOptions` ustawionym na wartość `false`.

Utwórz klienta przy użyciu tego samego identyfikatora klienta i adresu serwera, a następnie nawiąże połączenie, ustawiając ponownie atrybut `cleanSession` `MqttConnectionOptions` na wartość `false` .

Jeśli parametr `cleanSession` zostanie ustawiony na wartość `true`, oczekujące tokeny dostarczania zostaną odrzucone.

Istnieje możliwość sprawdzenia, czy istnieją oczekujące dostawy, wywołując komendę `MqttClient.getPendingDeliveryTokens`. Program `MqttClient.getPendingDeliveryTokens` można wywołać przed nawiązaniem połączenia z klientem.

Publikacja ostatniego testamentu i testamentu

Jeśli połączenie klienta MQTT niespodziewanie kończy się, można skonfigurować produkt WebSphere MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go przed nawiązaniem połączenia z klientem.

Utwórz temat dla ostatniej testamentu i testamentu. Użytkownik może utworzyć temat, taki jak `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Skonfiguruj "ostatni testament i testament" przy użyciu metody `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` .

Rozważ utworzenie znacznika czasu w komunikacie `lastWillPayload` . Dołączanie innych informacji o kliencie, które pomagają w identyfikacji klienta i okolicznościach połączenia. Przekaz obiekt `MqttConnectionOptions` do konstruktora `MqttClient` .

Set `lastWillQos` to 1 or 2, to make the message persistent in IBM WebSphere MQ, and to ensure delivery. Aby zachować ostatnie utracone informacje o połączeniu, ustaw wartość `lastWillRetained` na `true`.

Publikacja "last will and testament" jest wysyłana do subskrybentów, jeśli połączenie zostanie nieoczekiwanie zakończone. Jest on wysyłany, jeśli połączenie kończy się bez wywołania metody `MqttClient.disconnect` przez klienta.

Aby monitorować połączenia, należy uzupełnić publikację "ostatni testament i testament" z innymi publikacjami, aby rejestrować połączenia i programowane rozłączenia.

Trwałość komunikatu w klientach MQTT

Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz" lub "dokładnie jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

W produkcie MQTTtrwałość komunikatu ma dwa aspekty: sposób przesyłania komunikatu oraz informację o tym, czy jest on umieszczany w kolejce na serwerze MQTT jako komunikat trwały.

1. Trwałość komunikatu dla par klienta MQTT z jakością usługi. W zależności od tego, jaką jakość usługi wybierzesz dla komunikatu, komunikat jest trwały. Trwałość komunikatu jest niezbędna do zaimplementowania wymaganej jakości usługi.

   Jeśli zostanie określona wartość "at most once" (najwyżej raz), `QoS=0`(klient), klient odrzuci komunikat natychmiast po jego opublikowaniu. Jeśli w procesie przetwarzania komunikatu wystąpi awaria, komunikat nie zostanie wysłany ponownie. Nawet jeśli klient nadal będzie aktywny, komunikat nie zostanie wysłany ponownie. Zachowanie komunikatów produktu `QoS=0` jest takie samo, jak szybkie komunikaty nietrwałe produktu IBM WebSphere MQ .

   Jeśli komunikat jest publikowany przez klienta z funkcją QoS 1 lub 2, jest on trwały. Komunikat jest przechowywany lokalnie i usuwany z klienta tylko wtedy, gdy nie jest już potrzebny do zapewnienia "co najmniej raz", `QoS=1`lub "dokładnie raz", `QoS=2`, dostarczania.
2. Jeśli komunikat jest oznaczony jako `QoS 1` lub 2, jest on umieszczany w kolejce jako komunikat trwały. Jeśli jest ona oznaczona jako `QoS=0`, to jest umieszczana w kolejce jako komunikat nietrwały. W produkcie IBM WebSphere MQ komunikaty nietrwałe są przesyłane między menedżerami kolejek

"dokładnie jeden raz", chyba że dla kanału komunikatów atrybut NPMSPEED jest ustawiony na wartość FAST.

Trwała publikacja jest przechowywana na kliencie, dopóki nie zostanie odebrana przez aplikację kliencką. W przypadku systemu QoS=2publikacja jest odrzucana od klienta, gdy wywołanie zwrotne aplikacji zwraca element sterujący. W przypadku QoS=1 aplikacja może otrzymać publikację ponownie, jeśli wystąpi awaria. W przypadku systemu QoS=0wywołanie zwrotne odbiera publikację nie więcej niż raz. Może ona nie otrzymać publikacji, jeśli wystąpi awaria lub jeśli klient został odłączony w momencie publikacji.

Po zasubskrybowaniu tematu można zmniejszyć QoS , za pomocą którego subskrybent odbiera komunikaty, aby dopasować jego możliwości trwałości. Publikacje utworzone w wyższej jakości QoS są wysyłane z najwyższą jakością QoS , o którą zażądano subskrybentowi.

Zapisywanie komunikatów

Implementacja przechowywania danych na małych urządzeniach zmienia się bardzo wiele. Model tymczasowego zapisywania trwałych komunikatów w pamięci masowej, który jest zarządzany przez klienta MQTT , może być zbyt wolny lub wymagać zbyt dużej ilości pamięci masowej. W urządzeniach przenośnych mobilny system operacyjny może udostępniać usługę pamięci masowej, która jest idealna dla komunikatów MQTT .

Aby zapewnić elastyczność w spełnianiu ograniczeń dotyczących małych urządzeń, klient MQTT ma dwa interfejsy trwałości. Interfejsy definiują operacje, które są związane z zapisywaniem trwałych komunikatów. Interfejsy są opisane w dokumentacji interfejsu API dla Klient MQTT dla produktu Java. W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT. Interfejsy można zaimplementować w taki sposób, aby odpowiadał urządzeniu. Klient MQTT , który działa w środowisku Java SE, ma domyślną implementację interfejsów, które przechowują trwałe komunikaty w systemie plików. Używa pakietu `java.io` . Klient ma również domyślną implementację dla mechanizmu Java ME, `MqttDefaultMIDPPersistence`.

Klasy trwałości

`MqttClientPersistence`
Przekaz instancji implementacji produktu `MqttClientPersistence` do klienta MQTT jako parametr konstruktora `MqttClient` . Jeśli parametr `MqttClientPersistence` zostanie pominięty z konstruktora `MqttClient` , klient MQTT zapisuje komunikaty trwałe przy użyciu klasy `MqttDefaultFilePersistence` lub `MqttDefaultMIDPPersistence`.

`MqttPersistable`
Program `MqttClientPersistence` pobiera i umieszcza obiekty `MqttPersistable` przy użyciu klucza pamięci masowej. Należy udostępnić implementację produktu `MqttPersistable` oraz implementację produktu `MqttClientPersistence` , jeśli nie jest używany produkt `MqttDefaultFilePersistence` lub `MqttDefaultMIDPPersistence`.

`MqttDefaultFilePersistence`

Klient MQTT udostępnia klasę `MqttDefaultFilePersistence` . W przypadku tworzenia instancji produktu `MqttDefaultFilePersistence` w aplikacji klienckiej można udostępnić katalog, w którym będą przechowywane komunikaty trwałe jako parametr konstruktora `MqttDefaultFilePersistence` .

Alternatywnie klient MQTT może utworzyć instancję produktu `MqttDefaultFilePersistence` i umieścić pliki w katalogu domyślnym. Nazwa katalogu to *`client identifier`*`-tcp` *`hostname portnumber`*. Produkty `"\", "\\", "/", ":"` i `" "` są usuwane z łańcucha nazwy katalogu.

Ścieżka do katalogu jest wartością właściwości systemowej `rcp.data`. Jeśli parametr `rcp.data` nie jest ustawiony, to ścieżka jest wartością właściwości systemowej `usr.data`.

`rcp.data` to właściwość powiązana z instalacją platformy OSGi lub Eclipse Rich Client Platform (RCP).

usr.data to katalog, w którym uruchomiono komendę Java, która uruchomiła aplikację.

MqttDefaultMIDPPersistence
MqttDefaultMIDPPersistence ma konstruktor domyślny i nie ma żadnych parametrów. Używa on pakietu javax.microedition.rms.RecordStore do przechowywania komunikatów.

Publikacje

Publikacje to instancje produktu MqttMessage , które są powiązane z łańcuchem tematu. Klient MQTT może tworzyć publikacje, które mają być wysyłane do produktu IBM WebSphere MQ, a następnie zasubskrybować tematy w produkcie IBM WebSphere MQ MQ w celu otrzymywania publikacji.

MqttMessage ma tablicę bajtów jako jej ładunek. Dążą do tego, aby wiadomości były jak najmniejsze. Maksymalna długość komunikatu dozwolona przez protokół MQTT wynosi 250 MB.

Zwykle program kliencki MQTT używa produktu java.lang.String lub java.lang.StringBuffer do manipulowania treścią komunikatu. Dla wygody klasa MqttMessage ma metodę toString , która umożliwia przekształcenie jej ładunku w łańcuch. Aby utworzyć ładunek tablicy bajtów z poziomu java.lang.String lub java.lang.StringBuffer, należy użyć metody getBytes .

Metoda getBytes przekształca łańcuch w domyślny zestaw znaków dla platformy. Domyślnym zestawem znaków jest zazwyczaj UTF-8. Publikacje dotyczące produktu MQTT , które zawierają tylko tekst, są zwykle kodowane w produkcie UTF-8. Użyj metody getBytes("UTF8") , aby przesłonić domyślny zestaw znaków.

W produkcie IBM WebSphere MQpublikacja MQTT jest odbierana w postaci komunikatu jms-bytes . Komunikat zawiera folder MQRFH2 zawierający <mqtt>oraz folder <mqps> . Folder <mqtt> zawiera elementy clientId i qos, ale ta treść może ulec zmianie w przyszłości.

MqttMessage ma trzy dodatkowe atrybuty: jakość usługi, bez względu na to, czy jest ona zachowywana, oraz czy jest duplikatem. Flaga duplikowania jest ustawiana tylko wtedy, gdy jakość usługi jest "co najmniej raz" lub "dokładnie jeden raz". Jeśli komunikat został wcześniej wysłany i nie został uznany za wystarczająco szybko przez klienta MQTT , komunikat zostanie wysłany ponownie z zduplikowanym atrybutem ustawionym na wartość true.

Publikowanie

Aby utworzyć publikację w aplikacji klienckiej MQTT , należy utworzyć MqttMessage. Ustaw jego ładunek, jakość usługi i to, czy jest on zachowywany, a następnie wywołaj metodę MqttTopic.publish(MqttMessage message) ; MqttDeliveryToken jest zwracany, a zakończenie publikacji jest asynchroniczne.

Alternatywnie klient MQTT może utworzyć tymczasowy obiekt komunikatu dla użytkownika z parametrów metody MqttTopic.publish(byte [] payload, int qos, boolean retained) podczas tworzenia publikacji.

Jeśli w publikacji znajduje się "co najmniej raz" lub "dokładnie jeden raz" jakość usług, QoS=1 lub QoS=2, klient MQTT wywołuje interfejs MqttClientPersistence . Wywołuje on produkt MqttClientPersistence w celu zapisania komunikatu przed zwróceniem znacznika dostarczenia do aplikacji.

Aplikacja może zablokować do momentu dostarczenia wiadomości do serwera za pomocą metody MqttDeliveryToken.waitForCompletion . Alternatywnie, aplikacja może być kontynuowana bez blokowania. Jeśli chcesz sprawdzić, czy publikacje są dostarczane bez blokowania, zarejestruj instancję klasy zwrotnej, która implementuje MqttCallback z klientem MQTT . Klient MQTT wywołuje metodę MqttCallback.deliveryComplete od razu po dostarczyniu publikacji. W zależności od jakości usługi dostawa może być niemal natychmiastowa w przypadku produktu QoS=0lub może potrwać pewien czas w przypadku produktu QoS=2.

Użyj metody MqttDeliveryToken.isComplete , aby odpytać, czy dostawa jest kompletna. Jeśli wartością parametru MqttDeliveryToken.isComplete jest false, można wywołać MqttDeliveryToken.getMessage , aby pobrać treść wiadomości. Jeśli wynikiem wywołania programu MqttDeliveryToken.isComplete jest true, komunikat został usunięty, a wywołanie

metody `MqttDeliveryToken.getMessage` spowodowałoby zgłoszenie wyjątku pustego wskaźnika. Nie ma wbudowanej synchronizacji między `MqttDeliveryToken.getMessage` i `MqttDeliveryToken.isComplete`.

Jeśli klient rozłącza się przed odebraniem wszystkich oczekujących tokenów dostarczania, nowa instancja klienta może wysłać zapytanie do tokenów dostarczania oczekujących przed nawiązaniem połączenia. Dopóki klient nie połączy się, nie zostaną wykonane żadne nowe dostawy i można bezpiecznie zadzwonić do produktu `MqttDeliveryToken.getMessage`. Użyj metody `MqttDeliveryToken.getMessage` , aby dowiedzieć się, które publikacje nie zostały dostarczone. Oczekujące tokeny dostarczania są usuwane w przypadku nawiązania połączenia z `MqttConnectOptions.cleanSession` ustawionym na wartość domyślną `true`.

subskrypcja

Menedżer kolejek lub IBM MessageSight jest odpowiedzialny za tworzenie publikacji, które mają być wysyłane do subskrybenta MQTT . Menedżer kolejek sprawdza, czy filtr tematu w subskrypcji utworzonej przez klienta MQTT jest zgodny z łańcuchem tematu w publikacji. Zgodność może być dokładnym dopasowaniu lub może zawierać znaki wieloznaczne. Zanim opublikowanie zostanie przekazane do subskrybenta przez menedżera kolejek, menedżer kolejek sprawdza atrybuty tematu powiązane z publikacją. Jest ona zgodna z procedurą wyszukiwania opisaną w sekcji Subskrybowanie łańcucha tematu zawierającego znaki wieloznaczne w celu określenia, czy obiekt tematu administracyjnego nadaje uprawnienia użytkownika do subskrybowania.

Gdy klient MQTT otrzymuje publikację z "co najmniej raz" jakością usługi, wywołuje metodę `MqttCallback.messageArrived` w celu przetworzenia publikacji. Jeśli jakość usługi publikacji to "dokładnie jeden raz", `QoS=2`, klient MQTT wywołuje interfejs `MqttClientPersistence` w celu zapisania komunikatu po jego odebraniu. Następnie wywołuje on `MqttCallback.messageArrived`.

Jakość usług świadczonych przez klienta MQTT

Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu WebSphere MQ oraz klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM WebSphere MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Jakość usługi publikacji jest atrybutem produktu `MqttMessage`. Jest ona ustawiana przy użyciu metody `MqttMessage.setQos`.

Metoda `MqttClient.subscribe` może zmniejszyć jakość usługi stosowaną do publikacji wysyłanych do klienta w temacie. Jakość usługi publikacji przekazanej abonentowi może być inna niż jakość usługi w publikacji. Niższa z tych dwóch wartości jest używana do przekazywania publikacji.

Najwyżej raz
`QoS=0`

Komunikat jest dostarczany nie więcej niż jeden raz lub nie jest dostarczany w ogóle. Dostarczenie komunikatu za pośrednictwem sieci nie jest potwierdzane.

Komunikat nie jest zapisany. Komunikat może zostać utracony, jeśli klient zostanie rozłączony lub nastąpi awaria serwera.

`QoS=0` jest najszybszym trybem przesyłania. Czasami nazywa się "ogień i zapomnij".

Protokół MQTT nie wymaga, aby serwery przekazywały publikacje pod adresem `QoS=0` do klienta. Jeśli klient zostanie odłączony od chwili otrzymania przez serwer publikacji, może ona zostać odrzucona w zależności od serwera. Usługa telemetryczna (MQXR) nie usuwa komunikatów wysłanych z produktem `QoS=0`. Są one zapisywane jako komunikaty nietrwałe i są usuwane tylko w przypadku zatrzymania menedżera kolejek.

Co najmniej raz
`QoS=1`

`QoS=1` jest domyślnym trybem przesyłania.

Komunikat jest zawsze dostarczany co najmniej raz. Jeśli nadawca nie otrzyma potwierdzenia, komunikat zostanie wysłany ponownie z ustawioną opcją DUP , dopóki nie zostanie odebrany potwierdzenie. Odbiornik wyników może wielokrotnie wysyłać ten sam komunikat i może przetwarzać go wiele razy.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy, dopóki nie zostanie przetworzony.

Komunikat zostanie usunięty z odbiornika po przetworzeniu komunikatu. Jeśli odbiornik jest brokerem, komunikat jest publikowany do jego subskrybentów. Jeśli odbiorcą jest klient, komunikat jest dostarczany do aplikacji subskrybenta. Po usunięciu wiadomości odbiorca wysyła potwierdzenie do nadawcy.

Wiadomość zostanie usunięta od nadawcy po odebraniu potwierdzenia od odbiorcy.

Dokładnie jeden raz

QoS=2

Komunikat jest zawsze dostarczany dokładnie jeden raz.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy, dopóki nie zostanie przetworzony.

QoS=2 jest najbezpieczniejszym, ale najwolniejszym trybem przesyłania. Przed usunięciem komunikatu z nadajnika pobiera ona co najmniej dwie pary transmisji między nadawcą i odbiorcą. Komunikat może być przetwarzany u odbiorcy po pierwszej transmisji.

W pierwszej parze transmisji nadawca przesyła wiadomość i otrzymuje potwierdzenie od odbiorcy, że zapisał komunikat. Jeśli nadawca nie otrzyma potwierdzenia, komunikat zostanie wysłany ponownie z ustawioną opcją DUP , dopóki nie zostanie odebrany potwierdzenie.

W drugiej parze transmisji nadawca informuje odbiorcę o tym, że może zakończyć przetwarzanie komunikatu "PUBREL". Jeśli nadawca nie otrzyma potwierdzenia komunikatu "PUBREL" , komunikat "PUBREL" zostanie wysłany ponownie, dopóki nie zostanie odebrany potwierdzenie. Nadawca usuwa komunikat, który został zapisany po odebraniu potwierdzenia do komunikatu produktu "PUBREL" .

Odbiornik może przetworzyć komunikat w pierwszej lub drugiej fazie, pod warunkiem, że nie przeprzetwarzali komunikatu. Jeśli odbiornik jest brokerem, publikuje on komunikat do subskrybentów. Jeśli odbiorcą jest klient, dostarcza on komunikat do aplikacji subskrybenta. Odbiornik wysyła komunikat o zakończeniu z powrotem do nadawcy, który zakończył przetwarzanie komunikatu.

Zachowane publikacje i klienty MQTT

Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, najświeższą zachowaną publikację w tym temacie jest natychmiast przekazywana do użytkownika.

Metoda `MqttMessage.setRetained` służy do określania, czy publikacja na danym temacie jest zachowywana, czy nie.

Aby usunąć zachowaną publikację w programie IBM WebSphere MQ, należy uruchomić komendę MQSC CLEAR TOPICSTR**CLEAR TOPICSTR** .

Jeśli zostanie utworzona publikacja z ładunkiem o wartości NULL, pusta publikacja jest przekazywana do subskrybentów. Inne brokery produktu MQTT mogą nie przesyłać pustej publikacji do subskrybentów.

Jeśli opublikujesz niezachowaną publikację do tematu, który ma zachowaną publikację, zachowana publikacja nie będzie miała wpływu na tę publikację. Aktualni subskrybenci otrzymują nową publikację. Nowi subskrybenci otrzymują po raz pierwszy zachowaną publikację, a następnie otrzymują nowe publikacje.

Po utworzeniu lub zaktualizowaniu zachowanej publikacji należy wysłać publikację za pomocą opcji QoS lub 1 lub 2. Jeśli zostanie ona wysłana z QoS z 0, program IBM WebSphere MQ tworzy nietrwałą zachowaną publikację. Jeśli menedżer kolejek zostanie zatrzymany, publikacja nie zostanie zachowana.

Użyj zachowanych publikacji, aby zarejestrować najnowszą wartość pomiaru. Nowi subskrybenci zatrzymanego tematu natychmiast otrzymują najnowszą wartość pomiaru. Jeśli nie są pobierane żadne nowe pomiary od czasu ostatniego subskrybowania subskrybenta do tematu publikacji, a subskrybent

ponownie subskrybuje subskrybent, subskrybent otrzymuje najnowszą zachowaną publikację w temacie ponownie.

Subskrypcje

Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Utwórz subskrypcje przy użyciu metod `MqttClient.subscribe` , przekazując jeden lub więcej filtrów tematów oraz parametry jakości usług. Parametr jakości usługi ustawia maksymalną jakość usługi, którą subskrybent jest przygotowany do użycia w celu odebrania komunikatu. Wiadomości wysłane do tego klienta nie mogą być dostarczane z wyższą jakością usług. Jakość usługi jest ustawiana na niższą od pierwotnej wartości, gdy komunikat został opublikowany, a poziom określony dla subskrypcji. Domyślną jakością usługi dla odbierania komunikatów jest `QoS=1`, co najmniej raz.

Samo żądanie subskrypcji jest wysyłane z produktem `QoS=1`.

Publikacje są odbierane przez subskrybenta, gdy klient MQTT wywołuje metodę `MqttCallback.messageArrived` . Metoda `messageArrived` przekazuje także łańcuch tematu, z którym komunikat został opublikowany w subskrybencie.

Za pomocą metod `MqttClient.unsubscribe` można usunąć subskrypcję lub zestaw lub subskrypcje.

Komenda WebSphere MQ może usunąć subskrypcję. Lista subskrypcji za pomocą programu WebSphere MQ Explorer lub za pomocą komend **runmqsc** lub PCF. Zostaną nazwane wszystkie subskrypcje klienta MQTT. Nadano im nazwę w postaci: *`ClientIdentifier`*:*`Topic name`*

Jeśli przed nawiązaniem połączenia z klientem zostanie użyta wartość domyślna `MqttConnectOptions`lub zostanie ustawiona wartość `MqttConnectOptions.cleanSession` na `true` , wówczas wszystkie stare subskrypcje klienta zostaną usunięte po nawiązaniu połączenia przez klienta. Wszystkie nowe subskrypcje dokonane przez klienta podczas sesji zostaną usunięte po rozłączeniu.

Jeśli serwer `MqttConnectOptions.cleanSession` zostanie ustawiony na `false` przed nawiązaniem połączenia, wszystkie subskrypcje tworzone przez klienta zostaną dodane do wszystkich subskrypcji, które istniały dla klienta przed jego połączeniem. Wszystkie subskrypcje pozostaną aktywne po rozłączeniu połączenia z klientem.

Atrybut `cleanSession` można także rozpatrywać jako atrybut modalny, jeśli chodzi o jego wpływ na subskrypcje. W domyślnym trybie atrybutu `cleanSession=true` klient tworzy subskrypcje i odbiera publikacje tylko w zasięgu sesji. W alternatywnym trybie atrybutu `cleanSession=false` subskrypcje są trwałe. Można nawiązywać połączenia z klientem i je rozłączać, a jego subskrypcje pozostają aktywne. Po ponownym nawiązaniu połączenia z klientem odbiera on wszystkie niedostarczone publikacje. Po nawiązaniu połączenia klient może modyfikować zestaw subskrypcji aktywnych na jego potrzeby.

Przed nawiązaniem połączenia należy ustawić tryb `cleanSession` . Tryb ten trwa cały czas trwania sesji. Aby zmienić ustawienie trybu, należy rozłączyć połączenie z klientem, a następnie ponownie je nawiązać. Jeśli tryby zostaną zmienione z `cleanSession=false` na `cleanSession=true`, wszystkie poprzednie subskrypcje klienta oraz wszystkie publikacje, które nie zostały odebrane, zostaną usunięte.

Publikacje, które są zgodne z aktywnymi subskrypcjami, są wysyłane do klienta natychmiast po ich opublikowaniu. Jeśli klient zostanie rozłączony, zostaną one wysłane do klienta, jeśli połączy się ponownie z tym samym serwerem z tym samym identyfikatorem klienta i z `MqttConnectOptions.cleanSession` ustawionym na `false`.

Subskrypcje dla konkretnego klienta są identyfikowane przez identyfikator klienta. Można ponownie podłączyć klient z innego urządzenia klienckiego do tego samego serwera i kontynuować te same subskrypcje i otrzymywać niedostarczone publikacje.

Łańcuchy tematów i filtry tematów w klientach MQTT

Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM WebSphere MQ.

Łańcuchy tematów są używane do wysyłania publikacji do subskrybentów. Utwórz łańcuch tematu przy użyciu metody `MqttClient.getTopic(java.lang.String topicString)`.

Filtry tematów są używane do subskrybowania tematów i otrzymywania publikacji. Filtry tematów mogą zawierać znaki wieloznaczne. W przypadku znaków wieloznacznych można zasubskrybować wiele tematów. Aby utworzyć filtr tematów, należy użyć metody subskrypcji, na przykład `MqttClient.subscribe(java.lang.String topicFilter)`.

Łańcuchy tematów

Składnia łańcucha tematu IBM WebSphere MQ jest opisana w sekcji Strings tematów. Składnia łańcuchów tematów produktu MQTT jest opisana w klasie `MqttClient` w dokumentacji interfejsu API dla Klient MQTT dla produktu Java. W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT..

Składnia każdego typu łańcucha tematu jest prawie identyczna. Istnieją cztery niewielkie różnice:

1. Łańcuchy tematów wysyłane do programu IBM WebSphere MQ przez klienty MQTT muszą być zgodne z konwencją dla nazw menedżerów kolejek. W szczególności łańcuchy tematów nie mogą zawierać łączników.
2. Maksymalna długość różni się od siebie. Łańcuchy tematów IBM WebSphere MQ są ograniczone do 10,240 znaków. Klient MQTT może tworzyć łańcuchy tematów o długości do 65535 bajtów.
3. Łańcuch tematu utworzony przez klient MQTT nie może zawierać znaku o kodzie zero.
4. W produkcie WebSphere Message Broker, poziom tematu o wartości NULL, `'...//...'` był niepoprawny. Poziomy tematów o wartości NULL są obsługiwane przez produkt IBM WebSphere MQ.

W przeciwieństwie do publikowania/subskrybowania produktu IBM WebSphere MQ protokół `mqttv3` nie zawiera pojęcia administracyjnego obiektu tematu. Nie można skonstruować łańcucha tematu z obiektu tematu i łańcucha tematu. Jednak łańcuch tematu jest odwzorowywany na temat administracyjny w produkcie WebSphere MQ. Kontrola dostępu powiązana z tematem administracyjnym określa, czy publikacja jest publikowana w temacie, czy odrzucona. Atrybuty, które są stosowane do publikacji, gdy są przekazywane do subskrybentów, mają wpływ na atrybuty tematu administracyjnego.

Filtry tematów

Składnia filtru tematu produktu IBM WebSphere MQ jest opisana w sekcji Schemat znaków wieloznacznych oparty na temacie. Składnia filtrów tematów, które można skonstruować za pomocą klienta MQTT , jest opisana w klasie `MqttClient` w dokumentacji interfejsu API dla Klient MQTT dla produktu Java. W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT.

Składnia każdego typu filtru tematów jest prawie identyczna. Jedyna różnica polega na tym, że różne brokery MQTT interpretują filtr tematów. W produkcie WebSphere Message Broker V6wielopoziomowy znak wieloznaczny może być używany tylko na końcu filtru tematów. W produkcie WebSphere MQwielopoziomowy znak wieloznaczny może być używany na dowolnym poziomie w drzewie tematów, na przykład `USA/#/Dutchess County`.

Skorowidz programistyczny klienta MQTT

Poniżej znajdują się odsyłacze do produktu Mobile Messaging and M2M Client Packoraz do powiązanej dokumentacji interfejsu API klienta.

W produkcie Mobile Messaging and M2M Client Packbiblioteki klienta produktu MQTT są dostarczane wraz z wygenerowaną dokumentacją interfejsu API. Pakiet klienta można pobrać z serwisu IBM messaging community downloads.

Kopie najnowszej dokumentacji interfejsu API można wyświetlić w postaci elektronicznej, wykonując następujące odsyłacze do projektu Eclipse Paho :

- Klient MQTT dla klas Java
- Biblioteka klienta MQTT dla języka C
- Asynchroniczna biblioteka klienta MQTT dla języka C

Uwaga:

1. Połącz aplikacje MQTT Java z pakietem `org.eclipse.paho.client.mqttv3` , a nie z `com.ibm.micro.client.mqttv3`. . Pakiet `com.ibm.micro.client.mqttv3` jest dostarczany w celu obsługi istniejących aplikacji MQTT Java .
2. V 7.5.0.1 Połącz aplikacje klienta MQTT dla języka C z biblioteką `MQTTAsync` , a nie z biblioteką `MQTTClient` . `MQTTClient` jest udostępniany w celu obsługi istniejących aplikacji produktu MQTT dla systemu C.
3. Serwer Klient przesyłania komunikatów MQTT dla produktu JavaScript wymaga serwera MQTT , który obsługuje produkt WebSockets. Na przykład: IBM WebSphere MQ Version 7.5 i nowsze wersje tego działania.

Pierwsze kroki z serwerami MQTT

Serwery przesyłania komunikatów, które obsługują protokół transportowy MQTT , są dostępne w produkcie IBM i innych. Najbardziej podstawowy serwer MQTT umożliwia wymianę komunikatów przez aplikacje mobilne i urządzenia obsługiwane przez biblioteki klienta MQTT . IBM WebSphere MQ and IBM MessageSight are MQTT servers from IBM. Oprócz tego, że działają jako podstawowe serwery MQTT , wymieniają również komunikaty między aplikacjami klienckim MQTT a aplikacjami korporacyjnymi. Wszystkie serwery MQTT z produktu IBM obsługują protokół MQTT version 3.1 i MQTT na serwerze WebSocket protocol.

Bieżące serwery MQTT z programu IBM

IBM WebSphere MQ

- Produkt IBM WebSphere MQ udostępnia funkcję przesyłania komunikatów klasy korporacyjnej. Komponent telemetryczny umożliwia IBM WebSphere MQ również działanie jako serwer MQTT .
- Umożliwia to obsługę aplikacji mobilnych, machine-to-machine (M2M) i urządzeń opartych na urządzeniach, a także umożliwia wymianę komunikatów z aplikacjami do przesyłania komunikatów w przedsiębiorstwie, takimi jak aplikacje IBM WebSphere MQ i JMS .
- Instalacja produktu IBM WebSphere MQ obejmuje kopię pakietu SDK produktu MQTT z produktu IBM. Ten pakiet SDK udostępnia przykładowe aplikacje klienckie produktu MQTT oraz biblioteki klienta produktu MQTT , które obsługują te aplikacje.

  Uwaga: Aby pobrać najbardziej aktualną wersję tego pakietu SDK, należy pobrać Mobile Messaging and M2M Client Pack. Więcej informacji na ten temat zawiera sekcja “Pierwsze kroki z klientami MQTT” na stronie 11.
- Obsługa produktu MQTT została po raz pierwszy uwzględniona w produkcie IBM WebSphere MQ Version 7.0.1. Pełne informacje na temat każdej wersji produktu IBM WebSphere MQmożna znaleźć w następującej dokumentacji produktu:
  - WebSphere MQ Telemetry Wersja 7.5
  - WebSphere MQ Telemetry Wersja 7.1

Krótkie wprowadzenie do produktu IBM WebSphere MQoraz kroki, które należy wykonać w celu rozpoczęcia pracy z komponentem IBM WebSphere MQ Telemetry , znajdują się w sekcji “IBM WebSphere MQ jako serwer MQTT” na stronie 145.

IBM MessageSight

- Produkt IBM MessageSight jest serwerem MQTT opartym na urządzeniu, który może jednocześnie łączyć ogromną liczbę klientów MQTT i zapewnić wydajność i skalowalność niezbędną do pomieszczenia coraz większej liczby urządzeń mobilnych i czujników. Obsługuje on protokół MQTT version 3.1 i MQTT na serwerze WebSocket protocol.

- Główne funkcje i korzyści produktu IBM MessageSight jako serwera MQTT są następujące:
  - Wysokowydajność, niezawodność i skalowalne przesyłanie komunikatów.
  - Zaprojektowany specjalnie dla scenariuszy typu komputer-komputer (M2M) i Internet of Things (Internet of Things), wspierając masowe społeczności dla współbieżnie połączonych punktów końcowych.
  - Łatwość montażu i użytkowania. Może on być włączony i uruchomiony w ciągu 30 minut.
  - Obsługa rodzimych aplikacji mobilnych, które zawierają produkty Android i iOS.
  - Integracja z produktem IBM WebSphere MQ jako brokerem publikowania/subskrypcji.
- Aby uzyskać szybkie wprowadzenie do produktu IBM MessageSight, należy zapoznać się z wprowadzeniem MessageSight w serwisie YouTube i ogłoszeniem MessageSight. Szczegółowe informacje techniczne można znaleźć w dokumentacji produktu MessageSight.

IBM WebSphere MQ Telemetry daemon for devices

- Jest on również znany jako IBM WebSphere MQ Telemetry advanced client for C. Jest to mały serwer MQTT , który zwykle działa w lokalizacjach satelitarnych lub urządzeniach znajdujących się w pobliżu krawędzi sieci; na przykład w set-top bokach, zdalnych jednostkach telemetrycznych lub terminalach punktowych.
- Typowym zastosowaniem jest skoncentrowanie wielu połączeń klienta MQTT , które są następnie połączone z IBM WebSphere MQ przez internet w jednym połączeniu MQTT . Na przykład można zainstalować dużą liczbę czujników w budynku, połączyć je z serwerem IBM WebSphere MQ Telemetry daemon for devicesi połączyć demona z serwerem IBM WebSphere MQ.
- IBM WebSphere MQ Telemetry daemon for devices jest dołączany do produktu IBM WebSphere MQ. Do nawiązania połączenia z produktem IBM WebSphere MQwymagana jest oddzielna licencja. Patrz IBM United States Software Announcement 212-091.

Really Small Message Broker

- Really Small Message Broker (RSMB) to wersja serwera IBM WebSphere MQ Telemetry daemon for devices. Główna różnica jest w użyciu. RSMB to mały serwer testowy, dostępny w produkcie IBM alphaWorksi przeznaczony do użytku podczas oceny lub eksperymentowania z rozwiązaniami MQTT. Protokół RSMB obsługuje produkt MQTT na wielu platformach Linux , w systemie Windows XP, w systemie Apple Mac OS X Leopardoraz w systemie Unslung (Linksys NSLU12) .

Poprzednie serwery MQTT z programu IBM

WebSphere Message Broker ***(znany teraz pod nazwą IBM Integration Bus)***

- Produkt WebSphere Message Broker w wersji 6 udostępnił własny serwer MQTT . Obsługa została wymieniona w produkcie WebSphere Message Broker 7 przez komponent telemetryczny produktu IBM WebSphere MQ.

Inne serwery MQTT

Produkt MQTT.org obsługuje listę serwerów MQTT i brokerów na swojej stronie Oprogramowanie , w tym serwery Open Source.

Zadania pokrewne

"Pierwsze kroki z klientami MQTT" na stronie 11
Użytkownik może rozpocząć tworzenie aplikacji mobilnej lub aplikacji machine-to-machine (M2M) , budując i uruchamiając przykładową aplikację kliencką MQTT , która korzysta z biblioteki klienta MQTT . Przykładowe aplikacje i powiązane z nimi biblioteki klienckie są dostępne w katalogu Mobile Messaging and M2M Client Pack w serwisie IBM. Istnieją wersje aplikacji i bibliotek klienckich napisane w języku Java, w języku JavaScripti w języku C. Te aplikacje można uruchamiać na większości platform i urządzeń, w tym na urządzeniach z systemem Android i produktach firmy Apple.

IBM WebSphere MQ jako serwer MQTT

Wprowadzenie do korzystania z serwera MQTT , który jest dołączony do produktu IBM WebSphere MQ.

Aby rozpocząć, wykonaj kroki opisane w następujących artykułach:

- "instalowanieIBM WebSphere MQ" na stronie 145
- "Konfigurowanie usługi MQTT z poziomu wiersza komend" na stronie 147
- "Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer" na stronie 150

Uwaga: Szybkie rozpoczęcie pracy można rozpocząć za pomocą interfejsu wiersza komend. Jeśli jednak konfiguracja różni się znacząco od przykładu, potrzebujesz większej wiedzy i umiejętności, aby efektywnie korzystać z interfejsu wiersza komend. Za pomocą interfejsu IBM WebSphere MQ Explorer można łatwo uruchamiać i wykonywać standardowe czynności konfiguracyjne.

Kluczowe informacje koncepcyjne dotyczące komponentu IBM WebSphere MQ Telemetry można znaleźć w następujących artykułach w dokumentacji produktu IBM WebSphere MQ :

- Łączenie urządzeń telemetrycznych z menedżerem kolejek
- Usługa telemetryczna (MQXR)
- Kanały pomiarowe

Informacje pokrewne

Konfigurowanie menedżera kolejek dla telemetrii w systemach Linux i AIX
Konfigurowanie menedżera kolejek dla telemetrii w systemie Windows
Konfigurowanie rozproszonego kolejkowania w celu wysyłania komunikatów do klientów MQTT
Administrowanie produktem WebSphere MQ Telemetry

instalowanieIBM WebSphere MQ

Follow these instructions to obtain and install IBM WebSphere MQ and configure IBM WebSphere MQ Telemetry on Windows or Linux.

Zanim rozpoczniesz

W przypadku systemów operacyjnych obsługiwanych przez usługę MQTT działającą w systemie IBM WebSphere MQnależy zapoznać się z Wymagania systemowe produktu IBM WebSphere MQ Telemetry.

Uzyskaj pliki instalacyjne produktu IBM WebSphere MQ oraz licencję, wykonując jedną z następujących czynności:

1. Poproś administratora produktu IBM WebSphere MQ o pliki instalacyjne i zapytaj go, czy możesz zaakceptować umowę licencyjną.
2. Pobierz 90-dniową kopię próbną produktu IBM WebSphere MQ. Więcej informacji na ten temat zawiera sekcja Ocena: IBM WebSphere MQ.
3. Kup produkt IBM WebSphere MQ. Więcej informacji na ten temat zawiera sekcja Strona produktuIBM WebSphere MQ.

O tym zadaniu

Install IBM WebSphere MQ as `root` on Linux, and as an administrator on Windows. W czasie instalacji należy wybrać dodatkowe opcje `Usługa telemetryczna` i `Klienty telemetryczne` , aby zainstalować komponent IBM WebSphere MQ Telemetry . Utwórz ID użytkownika do administrowania produktem IBM WebSphere MQi sprawdź, czy zdefiniowany jest identyfikator użytkownika gościa. Identyfikator użytkownika gościa jest używany w przykładowej konfiguracji usługi MQTT w celu autoryzowania dostępu MQTT do produktu IBM WebSphere MQ.

Po zainstalowaniu produktu IBM WebSphere MQuruchom usługę MQTT , wykonując kroki opisane w sekcji “Konfigurowanie usługi MQTT z poziomu wiersza komend” na stronie 147 lub “Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer” na stronie 150.

Procedura

1. Zaloguj się jako użytkownik `root` w systemie Linuxlub jako administrator w systemie Windows.
2. Zainstaluj produkt IBM WebSphere MQ.

   Postępuj zgodnie z instrukcjami w sekcji Instalowanie serwera WebSphere MQ na serwerze Linux lub Instalowanie serwera WebSphere MQ w systemie Windows. Wybierz opcję `Usługi telemetryczne` i `Klienty telemetryczne` , aby zainstalować komponent IBM WebSphere MQ Telemetry .

   W systemie Linuxnależy zanotować instrukcję w sekcji "Co dalej" , aby dokonać instalacji podstawowej instalacji. Nawet jeśli ta instalacja jest jedyną instalacją produktu IBM WebSphere MQ na stacji roboczej, należy ją utworzyć jako podstawową. Patrz Pojedyncza instalacja produktu WebSphere MQ w wersji 7.1lub nowszej, skonfigurowana jako instalacja podstawowa.

   Aby dokładnie postępować zgodnie z przykładowymi instrukcjami konfiguracji, należy wykonać instalację podstawową.

   Wiele instalacji: Jeśli chcesz pracować z instalacją niepodstawową, uruchom komendę setmqenv . Konfiguruje ono środowisko IBM WebSphere MQ w oknie komend na stacji roboczej użytkownika. Patrz Wiele instalacji.

   Zakładając, że zaakceptowano domyślne miejsce instalacji oferowane przez program instalacyjny, produkt IBM WebSphere MQ jest zainstalowany w następujących katalogach:

   Linux (64-bitowe)

   ```
   /opt/mqm
   ```

   Windows (32-bitowa)

   ```
   C:\Program Files\IBM\WebSphere MQ
   ```

   Windows (64-bitowe)

   ```
   C:\Program Files (x86)\IBM\WebSphere MQ
   ```

   Katalog instalacyjny jest wyświetlany jako *MQ_INSTALLATION_PATH*
3. Opcjonalne: Dodaj użytkownika, który będzie administrować produktem IBM WebSphere MQ , z grupą `mqm` na tej stacji roboczej.

Ten krok jest opcjonalny w przypadku produktu Windows , ponieważ produkt IBM WebSphere MQ można administrować jako administrator produktu Windows . Więcej informacji na ten temat zawiera sekcja Uprawnienia do administrowania produktem WebSphere MQ w systemach UNIX i Windows.

Jeśli stacja robocza Windows jest elementem domeny, należy zapoznać się z domeną Domena Windows 2000 z ustawieniami domyślnymi i domenami Windows 2003 i Windows Server 2008 z domyślnymi uprawnieniami zabezpieczeń.

W systemie Linuxprogram instalacyjny tworzy użytkownika `mqm`, jako członka grupy `mqm`. Nadaj temu użytkownikowi hasło lub utwórz innego użytkownika z produktem `mqm` jako jego grupą podstawową.

4. Opcjonalne: Zaloguj się, korzystając z użytkownika, który został członkiem grupy `mqm` .

   Ten krok jest opcjonalny w przypadku produktu Windows , ponieważ produkt IBM WebSphere MQ można administrować jako administrator produktu Windows .

5. Sprawdź, czy identyfikator użytkownika gościa jest zdefiniowany na stacji roboczej użytkownika.

   The guest user ID is `"guest"` on Windows and `"nobody"` on Linux. Identyfikator użytkownika gościa nie wymaga żadnych uprawnień systemu operacyjnego ani praw.

Wyniki

Produkt IBM WebSphere MQ został zainstalowany na stacji roboczej jako podstawowa instalacja produktu IBM WebSphere MQ i została utworzona grupa `mqm`. Instalacja daje uprawnienia do administrowania produktem IBM WebSphere MQ w stosunku do członków grupy `mqm` . Członkowie grupy administratorów w systemie Windows mają również uprawnienia do administrowania produktem IBM WebSphere MQ.

Co dalej

1. Skonfiguruj usługę MQTT z poziomu wiersza komend lub IBM WebSphere MQ Explorer; patrz “Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer” na stronie 150 lub “Konfigurowanie usługi MQTT z poziomu wiersza komend” na stronie 147.
2. Przetestuj klienty Android, iOS, WebSockets, Javai "C" MQTT .
3. When you finish testing, remove the queue manager and MQTT service by running the command *MQ_INSTALLATION_PATH*`\mqxr\samples\CleanupMQM.bat` on Windows and *MQ_INSTALLATION_PATH*`/mqxr/samples/CleanupMQM.sh` on Linux.

Informacje pokrewne

Instalowanie produktu WebSphere MQ Telemetry

Instalowanie serwera WebSphere MQ w systemie Linux

Instalowanie serwera WebSphere MQ w systemie Windows

Konfigurowanie usługi MQTT z poziomu wiersza komend

Aby uruchomić przykładowe aplikacje produktu IBM WebSphere MQ Telemetry , należy skonfigurować produkt IBM WebSphere MQ przy użyciu wiersza komend. W poniższych krokach przedstawiono sposób uruchamiania skryptu w celu utworzenia usługi MQTT w nowym menedżerze kolejek o nazwie `MQXR_SAMPLE_QM`.

Zanim rozpoczniesz

Aby skonfigurować usługę MQTT , należy mieć dostęp administracyjny do menedżera kolejek produktu IBM WebSphere MQ . Użytkownik ma wiele sposobów uzyskiwania dostępu do menedżera kolejek:

1. Pobierz kopię produktu IBM WebSphere MQ i utwórz menedżer kolejek na własnej stacji roboczej Linux lub Windows . Aby uzyskać i zainstalować produkt IBM WebSphere MQ, postępuj zgodnie z instrukcjami w sekcji “instalowanieIBM WebSphere MQ” na stronie 145 . Należy pamiętać, że podczas instalowania należy również wybrać opcję `Usługi telemetryczne` i `Klienty telemetryczne` . Można również zmodyfikować istniejącą instalację w celu dodania tych opcji.

2. Skontaktuj się z administratorem produktu IBM WebSphere MQ i poproś o dostęp administracyjny do menedżera kolejek na serwerze, na którym zainstalowano opcję IBM WebSphere MQ Telemetry .

   V 7.5.0.1 Oprócz nazwy menedżera kolejek wymagane są co najmniej dwa porty TCP/IP dla produktu MQTT oraz dla produktu MQTT na serwerze WebSockets. Jeśli planowane jest połączenie bezpiecznych klientów, wymagane jest co najmniej dwa porty.

Aby wykonać kroki w zadaniu dokładnie tak, jak są opisane, konieczne jest utworzenie menedżera kolejek o nazwie `MQXR_SAMPLE_QM`, a port TCP/IP 1883 musi być nieużywany.

O tym zadaniu

W tym zadaniu uruchamiany jest skrypt, który tworzy menedżer kolejek, a następnie konfiguruje usługę MQTT w celu nasłuchiwania połączeń klientów MQTT V3.1 na porcie 1883. Konfiguracja daje każdemu uprawnienie do publikowania i subskrybowania dowolnego tematu. Konfiguracja zabezpieczeń i kontroli dostępu jest minimalna. Jest ona przeznaczona tylko dla menedżera kolejek, który znajduje się w zabezpieczonej sieci z ograniczonym dostępem. Aby uruchomić produkty IBM WebSphere MQ i MQTT w niezabezpieczonym środowisku, należy skonfigurować zabezpieczenia. W celu skonfigurowania zabezpieczeń produktów IBM WebSphere MQ i MQTT należy zapoznać się z informacjami dostępnymi po kliknięciu odsyłaczy do stron pokrewnych znajdujących się na końcu tego zadania.

Procedura

1. Zaloguj się z identyfikatorem użytkownika, który ma uprawnienia administracyjne do produktu IBM WebSphere MQ.

   Aby zdefiniować identyfikator użytkownika z uprawnieniami administracyjnymi do produktu IBM WebSphere MQ, należy zapoznać się z krokiem 3 w sekcji “instalowanieIBM WebSphere MQ” na stronie 145.
2. Otwórz okno komend i uruchom przykładowy skrypt komend w celu utworzenia i uruchomienia przykładowego menedżera kolejek o nazwie `MQXR_SAMPLE_QM` i usługi MQTT .

   The path to the sample script is `%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat` on Windows and *`MQ_INSTALLATION_PATH`*`/mqxr/samples/SampleMQM.sh` on Linux.

   Wpisz następującą komendę, aby utworzyć i skonfigurować menedżer kolejek:

   - Windows

     ```
     "%MQ_FILE_PATH%\mqxr\samples\SampleMQM.bat"
     ```

   - Linux

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

Wyniki

W tym przykładzie tworzony jest kanał produktu MQTT o nazwie `PlainText` z następującymi właściwościami w systemie 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
```

Właściwości kanału w systemie Linux są takie same, jak Windows, z wyjątkiem `com.ibm.mq.MQXR.UserName=nobody`.

Klienty MQTT V3.1 , które łączą się z portem 1883, uzyskują dostęp IBM WebSphere MQ z ID użytkownika ustawionym w zmiennej *com.ibm.mq.MQXR.UserName*. Przykładowy skrypt autoryzuje ID użytkownika za pomocą następujących komend produktu 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
```

Pierwsza komenda daje użytkownikowi uprawnienia do publikowania i subskrybowania tematów dziedziczących uprawnienia z tematu podstawowego. Druga komenda daje użytkownikowi uprawnienia do umieszczania komunikatów w kolejce transmisji SYSTEM.MQTT.TRANSMIT.QUEUE . Usługa MQTT wysyła komunikaty na temat SYSTEM.MQTT.TRANSMIT.QUEUE jako publikacje subskrybentów produktu MQTT .

Skrypt uruchamia usługę MQTT w kolejce, aby nasłuchiwać połączeń na porcie 1883.

Co dalej

Wykonaj poniższe kroki, aby przetestować połączenie, uruchamiając przykładową aplikację MQTT V3.1 Java .

Źródło przykładowej aplikacji Java znajduje się w pliku MQTTV3Sample.java .

Do uruchomienia przykładu wymagane są dwa okna komend. Uruchom przykład jako subskrybent w jednym oknie i jako publikator w innym oknie.

- Windows Aby uruchomić subskrybent, uruchom komendę

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat" -a subscriber
  ```

  Aby opublikować komendę, uruchom komendę:

  ```
  "%MQ_FILE_PATH%\mqxr\samples\RunMQTTV3Sample.bat"
  ```

- Linux Aby uruchomić subskrybent, uruchom komendę

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscriber
  ```

  Aby opublikować komendę, uruchom komendę:

  ```
  MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh
  ```

Publikator i subskrybent zapisuje dane wyjściowe do swoich okien komend:

```
Connected to tcp://localhost:1883
Publishing to topic "MQTTV3Sample/Java/v3" qos 2
Disconnected
Press any key to continue . . .
```

Rysunek 27. Dane wyjściowe publikatora

```
Connected to tcp://localhost:1883
Subscribing to topic "MQTTV3Sample/#" qos 2
Press <Enter> to exit
Topic:          MQTTV3Sample/Java/v3
Message:        Message from MQTTv3 Java client
QoS:            2
```

Rysunek 28. Dane wyjściowe subskrybenta

Serwer jest teraz gotowy do przetestowania aplikacji MQTT V3.1 .

Zadania pokrewne

Konfigurowanie usługi MQTT za pomocą programu WebSphere MQ Explorer

Wykonaj poniższe instrukcje, aby skonfigurować produkt IBM WebSphere MQ przy użyciu produktu IBM WebSphere MQ Explorer do uruchamiania przykładowych klientów IBM WebSphere MQ Telemetry . W poniższych krokach przedstawiono sposób tworzenia usługi produktu MQTT przez uruchomienie kreatora konfiguracji produktu `Define sample` .

Informacje pokrewne

WebSphere MQ Telemetry

Tworzenie aplikacji dla produktu WebSphere MQ Telemetry

Administrowanie produktem WebSphere MQ Telemetry

Zabezpieczenia produktu WebSphere MQ Telemetry

Konfigurowanie usługi MQTT przy użyciu produktu IBM WebSphere MQ Explorer

Wykonaj poniższe instrukcje, aby skonfigurować produkt IBM WebSphere MQ przy użyciu produktu IBM WebSphere MQ Explorer do uruchamiania przykładowych klientów IBM WebSphere MQ Telemetry . W poniższych krokach przedstawiono sposób tworzenia usługi produktu MQTT przez uruchomienie kreatora konfiguracji produktu `Define sample` .

Zanim rozpoczniesz

Aby skonfigurować usługę MQTT , należy mieć dostęp administracyjny do menedżera kolejek produktu IBM WebSphere MQ . Użytkownik ma wiele sposobów uzyskiwania dostępu do menedżera kolejek:

1. Pobierz kopię produktu IBM WebSphere MQ i utwórz menedżer kolejek na własnej stacji roboczej Linux lub Windows . Aby uzyskać i zainstalować produkt IBM WebSphere MQ, postępuj zgodnie z instrukcjami w sekcji “instalowanieIBM WebSphere MQ” na stronie 145 . Należy pamiętać, że podczas instalowania należy również wybrać opcję `Usługi telemetryczne` i `Klienty telemetryczne` . Można również zmodyfikować istniejącą instalację w celu dodania tych opcji.
2. Skontaktuj się z administratorem produktu IBM WebSphere MQ i poproś o dostęp administracyjny do menedżera kolejek na serwerze, na którym zainstalowano opcję IBM WebSphere MQ Telemetry .

   V 7.5.0.1 Oprócz nazwy menedżera kolejek wymagane są co najmniej dwa porty TCP/IP dla produktu MQTT oraz dla produktu MQTT na serwerze WebSockets. Jeśli planowane jest połączenie bezpiecznych klientów, wymagane jest co najmniej dwa porty.

Aby wykonać kroki w zadaniu dokładnie tak, jak są opisane, konieczne jest utworzenie menedżera kolejek o nazwie `MQXR_SAMPLE_QM`, a port TCP/IP 1883 musi być nieużywany.

O tym zadaniu

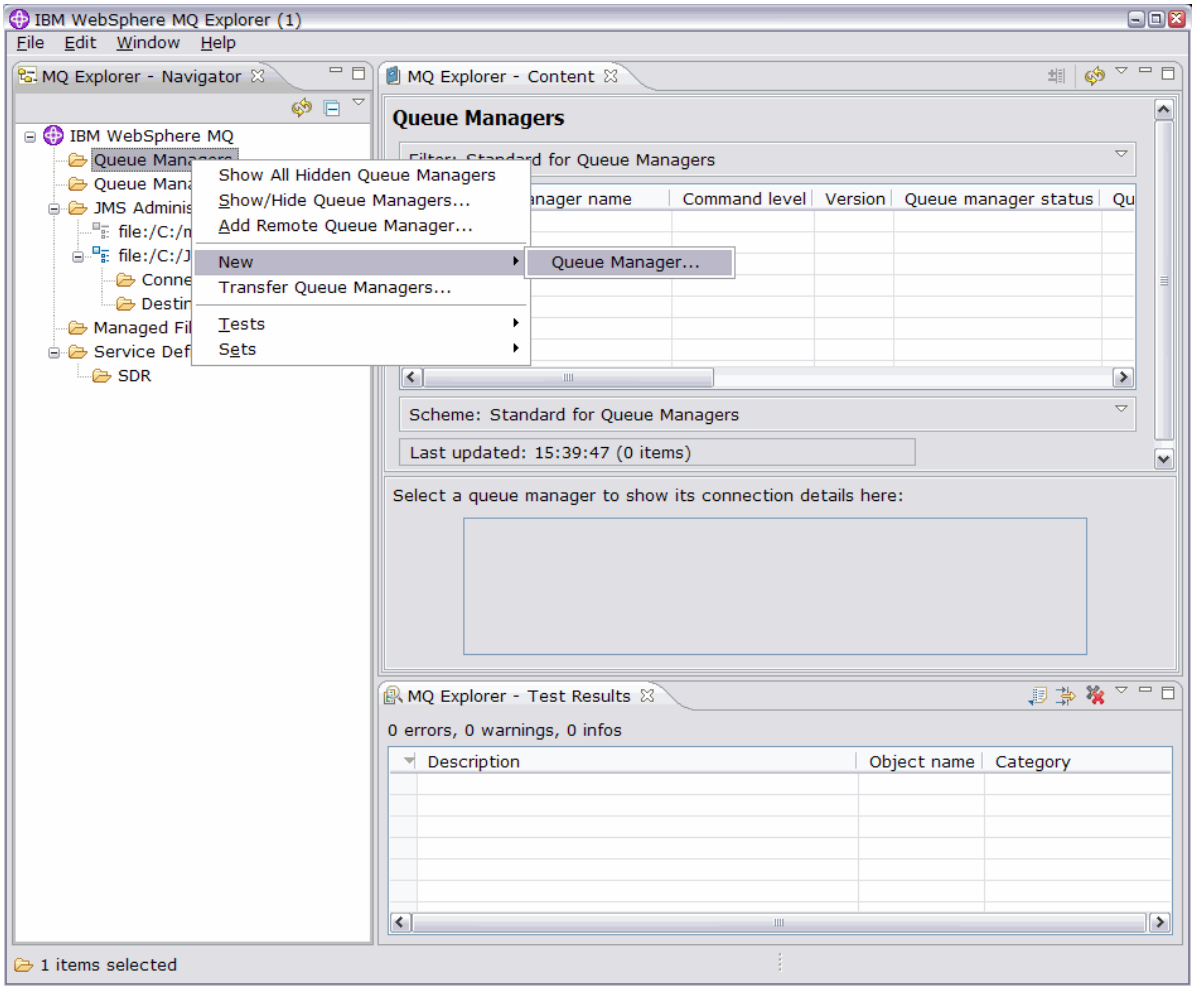
W tym zadaniu należy uruchomić kreatora konfiguracji IBM WebSphere MQ Explorer `Define sample` w celu utworzenia usługi MQTT w celu nasłuchiwania połączeń klientów MQTT V3.1 na porcie 1883. Konfiguracja daje każdemu uprawnienie do publikowania i subskrybowania dowolnego tematu. Konfiguracja zabezpieczeń i kontroli dostępu jest minimalna. Jest ona przeznaczona tylko dla menedżera kolejek, który znajduje się w zabezpieczonej sieci z ograniczonym dostępem. Aby uruchomić produkty IBM WebSphere MQ i MQTT w niezabezpieczonym środowisku, należy skonfigurować zabezpieczenia. W celu skonfigurowania zabezpieczeń produktów IBM WebSphere MQ i MQTT należy zapoznać się z informacjami dostępnymi po kliknięciu odsyłaczy do stron pokrewnych znajdujących się na końcu tego zadania.

Procedura

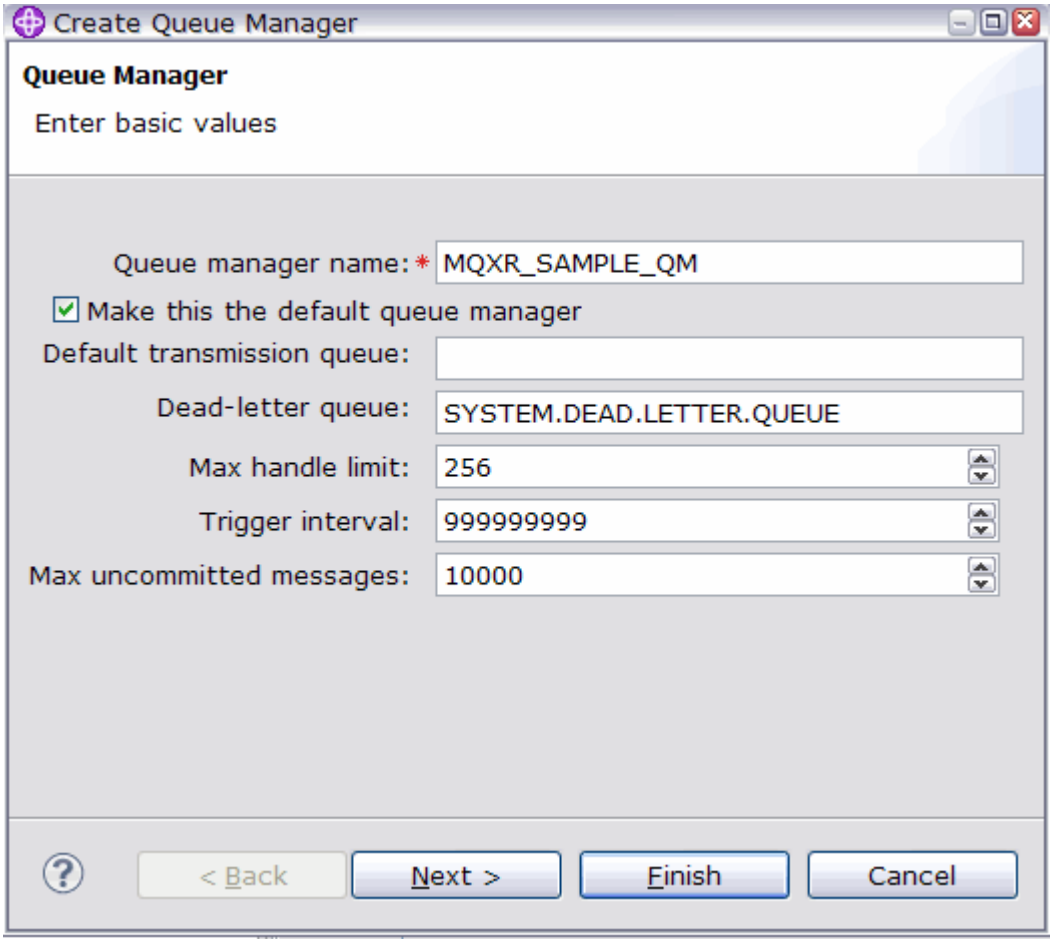
1. Zaloguj się z identyfikatorem użytkownika, który ma uprawnienia administracyjne do produktu IBM WebSphere MQ.

   Aby zdefiniować identyfikator użytkownika z uprawnieniami administracyjnymi do produktu IBM WebSphere MQ, należy zapoznać się z krokiem 3 w sekcji “instalowanieIBM WebSphere MQ” na stronie 145.

2. Otwórz okno komend, a następnie uruchom komendę IBM WebSphere MQ Explorer **strmqcfg** , aby uruchomić program IBM WebSphere MQ Explorer.
3. Tworzenie menedżera kolejek
   a) Uruchom kreator **Nowy menedżer kolejek** .

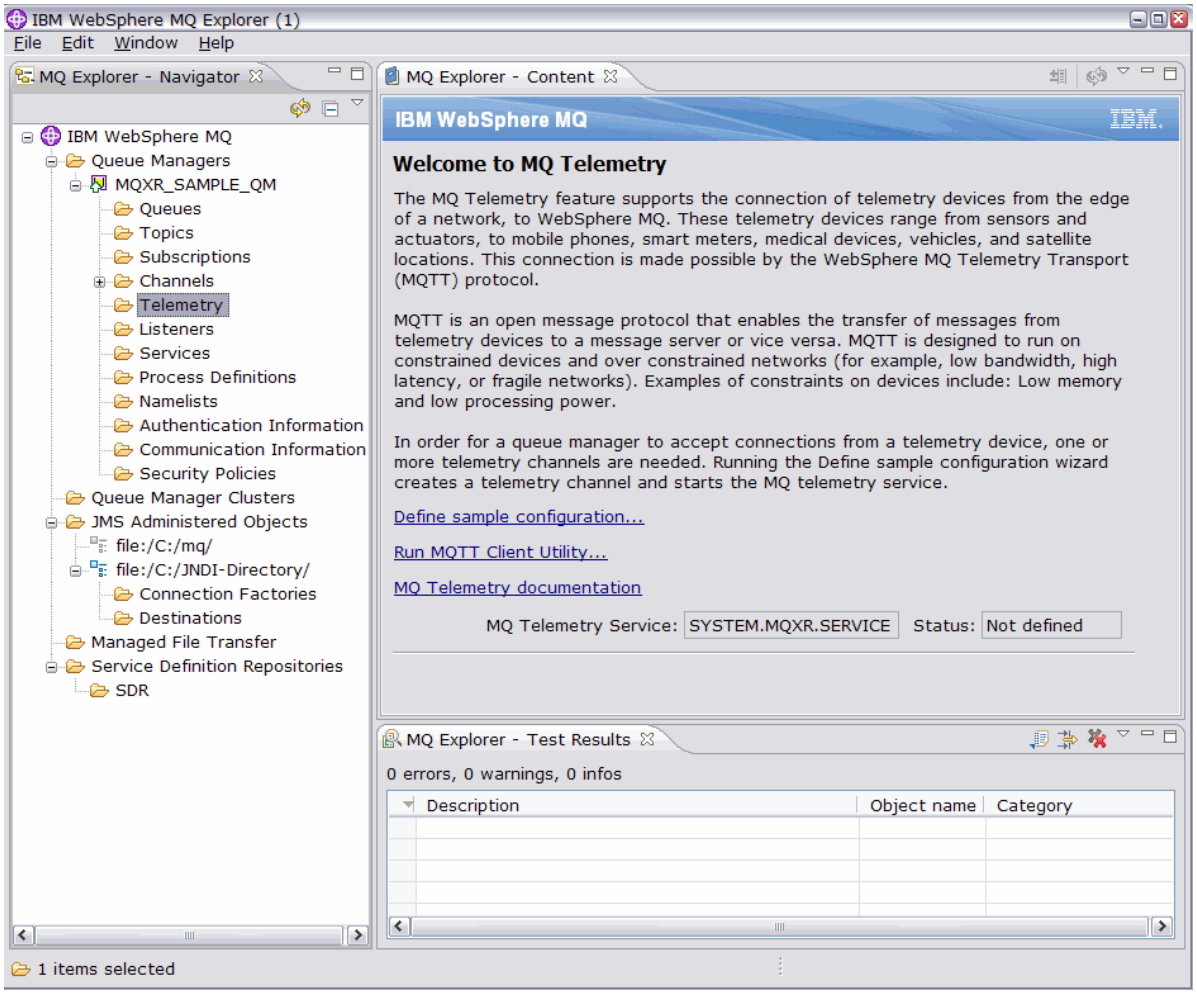


   b) Wpisz wartość w polu **Nazwa menedżera kolejek**i nazwę **kolejki niedostarczanych komunikatów**. Dla wygody, ustaw ją jako domyślny menedżer kolejek. Kliknij opcję **Zakończ**.



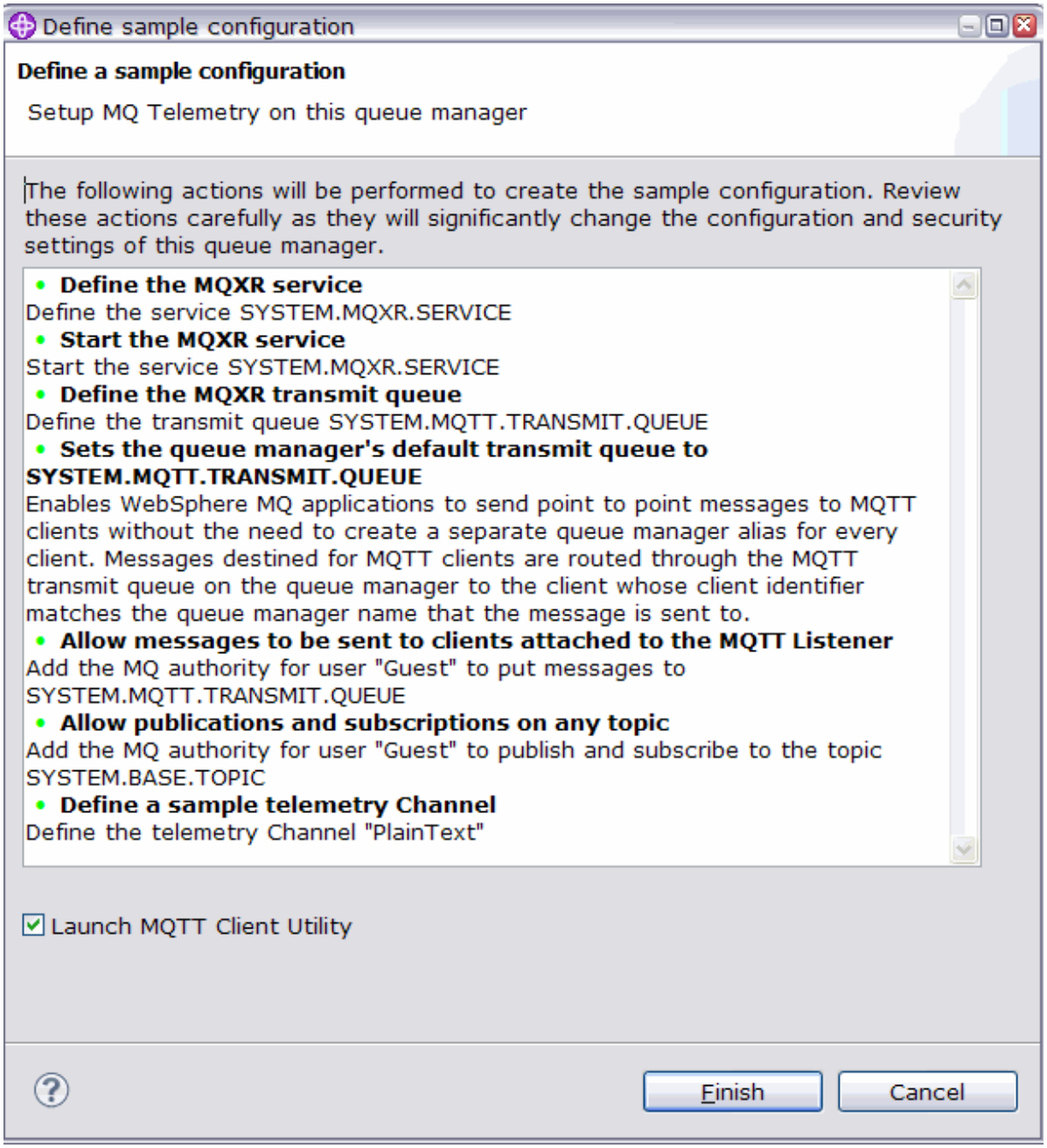
Program IBM WebSphere MQ Explorer tworzy menedżer kolejek i uruchamia go.

4. Uruchom kreator Telemetria **Definiowanie przykładowej konfiguracji** .
   a) Otwórz folder Telemetry dla menedżera kolejek.



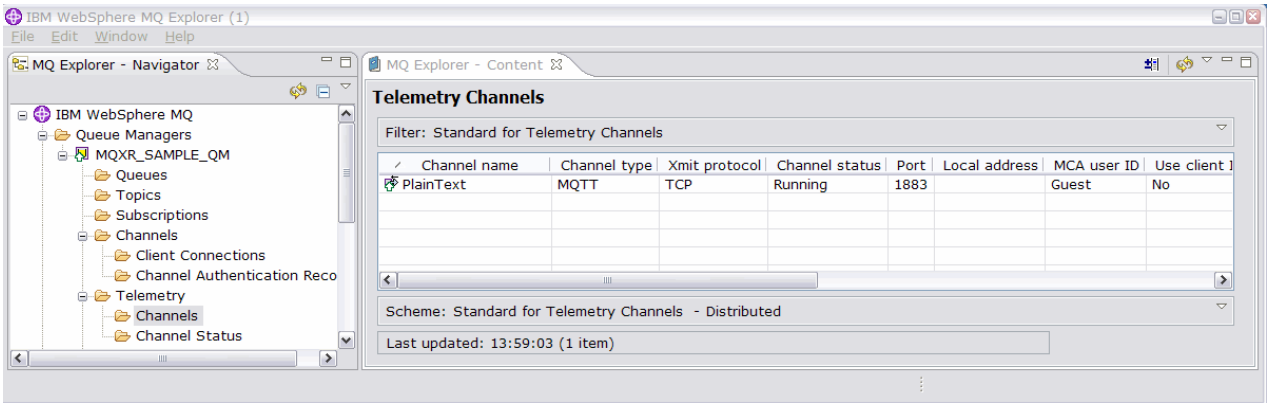
b) Kliknij opcję **Definiuj przykładową konfigurację** , aby uruchomić kreator.

c) Kliknij przycisk **Zakończ** , aby utworzyć usługę telemetryczną i uruchomić program narzędziowy klienta MQTT .



Wyniki

Otwórz folder Kanały telemetryczne, aby wyświetlić listę kanałów przykładowych.



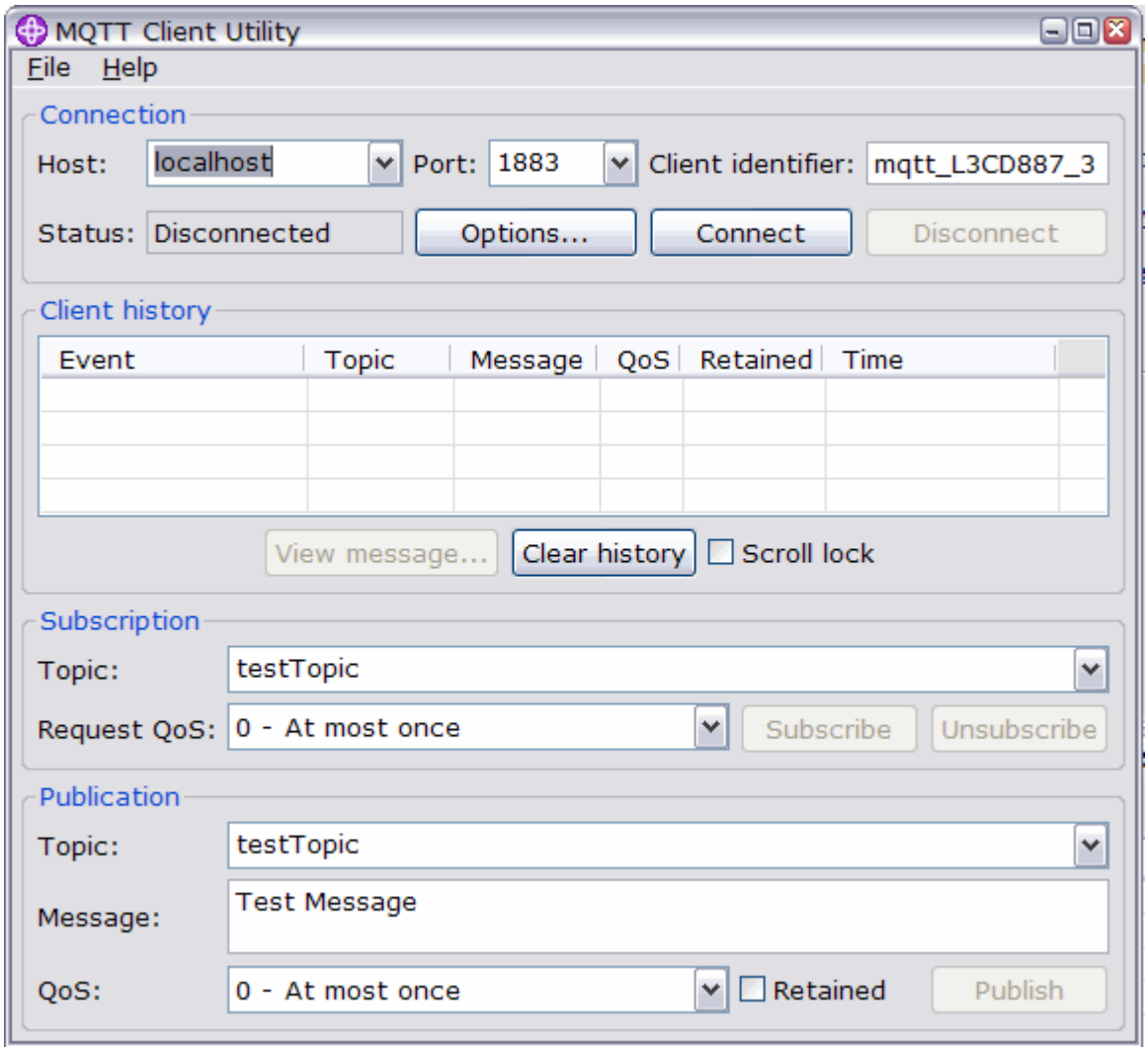
Użytkownik może modyfikować właściwości tego kanału oraz dodawać i usuwać kanały w tym oknie.

Co dalej

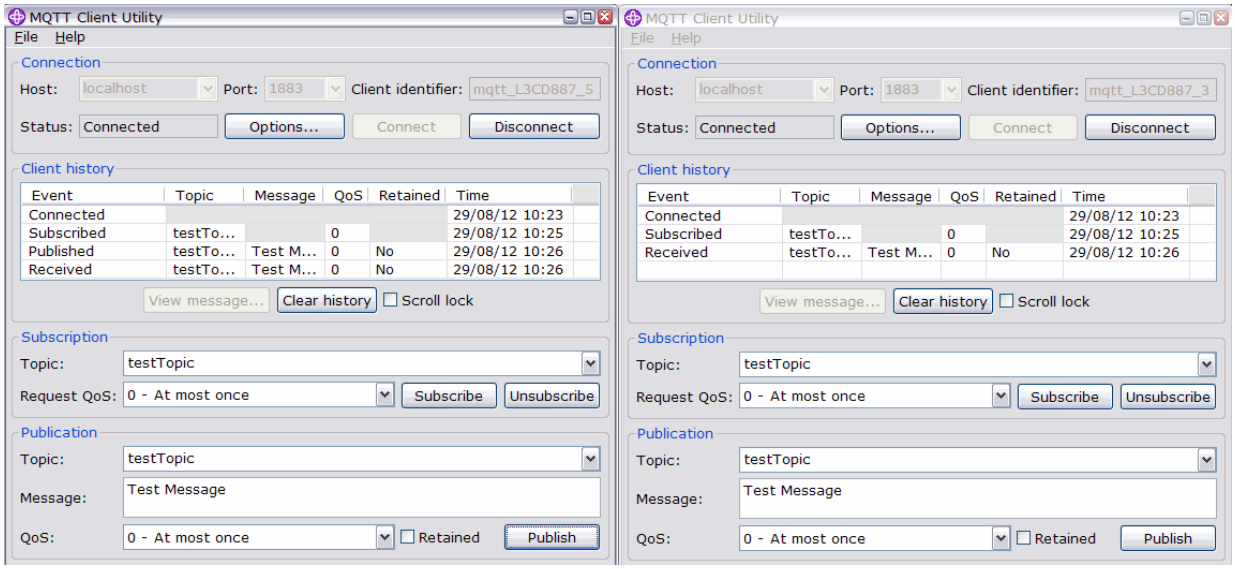
Przetestuj połączenie, uruchamiając program narzędziowy klienta produktu MQTT .

1. Aby uruchomić program narzędziowy klienta, otwórz folder **Telemetria** i dwukrotnie kliknij opcję **Uruchom program narzędziowy klienta MQTT** .

   Dwa okna programu **Program narzędziowy klienta MQTT** są otwarte, identyczne, ale dla różnych identyfikatorów klienta.



2. Kliknij opcję **Połącz** w obu oknach.
3. Kliknij opcję **Subskrybuj** w obu oknach.
4. W dowolnym oknie kliknij opcję **Publikuj** . Wyniki są wyświetlane w programie Rysunek 29 na stronie 156 .



Rysunek 29. Wyniki

5. W obu oknach kliknij opcję **Rozłącz** .

Serwer jest teraz gotowy do przetestowania aplikacji MQTT V3.1 .

Zadania pokrewne

Konfigurowanie usługi MQTT z poziomu wiersza komend

Aby uruchomić przykładowe aplikacje produktu IBM WebSphere MQ Telemetry , należy skonfigurować produkt IBM WebSphere MQ przy użyciu wiersza komend. W poniższych krokach przedstawiono sposób uruchamiania skryptu w celu utworzenia usługi MQTT w nowym menedżerze kolejek o nazwie `MQXR_SAMPLE_QM`.

Administrowanie produktem WebSphere MQ Telemetry

Informacje pokrewne

WebSphere MQ Telemetry

Administrowanie produktem WebSphere MQ Telemetry za pomocą programu WebSphere MQ Explorer

Tworzenie aplikacji dla produktu WebSphere MQ Telemetry

Zabezpieczenia

Zabezpieczenia produktu WebSphere MQ Telemetry

IBM WebSphere MQ Demon telemetryczny dla pojęć związanych z urządzeniami

Demon IBM WebSphere MQ Telemetry dla urządzeń jest zaawansowaną aplikacją kliencką MQTT V3 . Służy do przechowywania i przekazywania komunikatów z innych klientów MQTT. Łączy się on z produktem IBM WebSphere MQ , takim jak klient MQTT, ale można również połączyć z nim inne klienty MQTT.

Demon jest brokerem publikowania/subskrypcji. Klienty MQTT V3 łączą się z nim w celu publikowania i subskrybowania tematów, przy użyciu łańcuchów tematów do publikowania i filtrowania tematów w celu zasubskrybowania. Łańcuch tematu jest hierarchiczny i zawiera poziomy tematów podzielone przez produkt /. Filtry tematów są łańcuchami tematów, które mogą zawierać znaki wieloznaczne + na poziomie pojedynczego poziomu oraz wielopoziomowy znak wieloznaczny # jako ostatni fragment łańcucha tematu.

Uwaga: Znaki wieloznaczne w demonie są zgodne z bardziej restrykcyjnymi regułami produktu WebSphere Message Broker, v6. IBM WebSphere MQ jest inny. Obsługuje on wielopoziomowe znaki wieloznaczne; znaki wieloznaczne mogą znajdować się na dowolnej liczbie poziomów hierarchii, w dowolnym miejscu łańcucha tematu.

Wiele klientów MQTT v3 łączy się z demonem przy użyciu portu nasłuchiwania. Domyślny port nasłuchiwania jest modyfikowalny. Można zdefiniować wiele portów nasłuchiwania i przydzielić do nich

różne przestrzenie nazw, patrz “Demon WebSphere MQ Telemetry dla portów nasłuchiwania urządzeń” na stronie 165. Demon jest sam w sobie klientem MQTT v3 . Skonfiguruj połączenie z mostem demona w celu połączenia demona z portem nasłuchiwania innego demona lub do usługi WebSphere MQ Telemetry (MQXR).

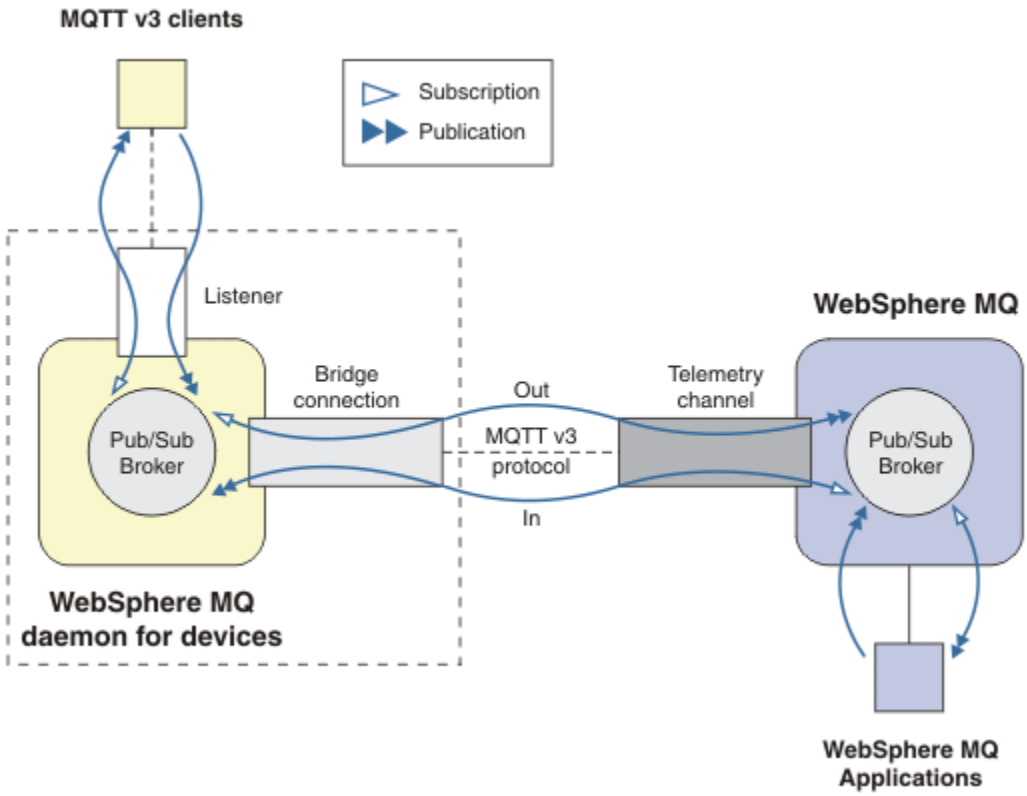
Istnieje możliwość skonfigurowania wielu mostów dla demona WebSphere MQ Telemetry dla urządzeń. Użyj mostów, aby połączyć sieć demonów, które mogą wymieniać się publikacjami.

Każdy most może publikować i subskrybować tematy w swoim lokalnym demonie. Może on także publikować i subskrybować tematy w innym demonie, brokerze publikowania/subskrypcji produktu WebSphere MQ lub dowolnym innym brokerze MQTT v3 , z którym jest połączony. Korzystając z filtru tematów, można wybrać publikacje, które mają być propagowane z jednego brokera do innego. Publikacje można propagować w dowolnym kierunku. Publikacje publiczne można propagować z demona lokalnego do każdego z przyłączonych brokerów zdalnych lub z dowolnego z przyłączonych brokerów do lokalnego demona; patrz “IBM WebSphere MQ Demon telemetryczny dla mostów urządzeń” na stronie 157.

IBM WebSphere MQ Demon telemetryczny dla mostów urządzeń

Demon IBM WebSphere MQ Telemetry dla urządzeń mostu łączy dwa brokery publikowania/subskrybowania przy użyciu protokołu MQTT v3 . Most propaguje publikacje z jednego brokera do drugiego w dowolnym kierunku. Na jednym końcu jest to demon WebSphere MQ Telemetry dla połączenia mostu urządzeń, a z drugiej strony może być menedżerem kolejek lub innym demonem. Menedżer kolejek jest połączony z połączeniem mostu za pomocą kanału telemetrycznego. Demon jest połączony z połączeniem mostu przy użyciu obiektu nasłuchiwania demona.

IBM WebSphere MQ Demon telemetryczny dla urządzeń obsługuje jedno lub większą liczbę jednoczesnych połączeń z innymi brokerami. Połączenia z demona są nazywane mostkami i są definiowane przez pozycje połączeń w pliku konfiguracyjnym demona. Połączenia z produktem IBM WebSphere MQ są nawiązywane przy użyciu kanałów pomiarowych IBM WebSphere MQ , jak pokazano na poniższym rysunku:



Rysunek 30. Łączenie IBM WebSphere MQ Telemetry daemon for devices z IBM WebSphere MQ

Most łączy demon z innym brokerem jako klient MQTT v3 . Parametry mostu odzwierciedlają atrybuty klienta MQTT v3 .

Most jest czymś więcej niż połączeniem. Działa on jako agent publikowania i subskrypcji znajdujący się między dwoma brokerami publikowania/subskrybowania. Lokalnym brokerem jest demon IBM WebSphere MQ Telemetry dla urządzeń, a broker zdalny jest dowolnym brokerem publikowania/subskrypcji, który obsługuje protokół MQTT v3 . Zwykle zdalnym brokerem jest inny demon lub IBM WebSphere MQ.

Zadaniem mostu jest propagacja publikacji pomiędzy dwoma brokerami. Most jest dwukierunkowy. Propaguje publikacje w dowolnym kierunku. Rysunek 30 na stronie 158 ilustruje sposób, w jaki most łączy demon IBM WebSphere MQ Telemetry dla urządzeń z serwerem IBM WebSphere MQ. W produkcie "Przykładowe ustawienia tematu dla mostu" na stronie 159 użyto przykładów ilustrujących sposób użycia parametru tematu w celu skonfigurowania mostu.

Strzałki In i Out w programie Rysunek 30 na stronie 158 wskazują dwukierunkowość mostu. Na jednym końcu strzałki jest tworzona subskrypcja. Publikacje, które są zgodne z subskrypcją, są publikowane w brokerze na przeciwległy koniec strzałki. Strzałka jest oznaczona zgodnie z przepływem publikacji. Przepływ publikacji W do demona i Wyjście z demona. Znaczenie etykiet jest używane w składni komendy. Należy pamiętać, że In i Out odnoszą się do miejsca, w którym przepływ publikacji jest wysyłany, a nie do miejsca, w którym wysyłana jest subskrypcja.

Inne klienty, aplikacje lub brokery mogą być połączone z programem IBM WebSphere MQ lub z demonem WebSphere MQ Telemetry dla urządzeń. Publikują one i subskrybują tematy w brokerze, z którym są połączone. Jeśli broker ma wartość IBM WebSphere MQ, tematy mogą być grupowane lub dystrybuowane, a nie jawnie zdefiniowane w lokalnym menedżerze kolejek.

Zastosowania mostów

Połącz demonów razem przy użyciu połączeń mostów i programów nasłuchujących. Łączenie demonów i menedżerów kolejek przy użyciu połączeń mostu i kanałów telemetrycznych. Po połączeniu wielu brokerów możliwe jest utworzenie pętli. Uważaj: Publikacje mogą krążyć w nieskończoność wokół pętli brokerów, niewykryte.

Niektóre z przyczyn używania demonów mostkowanych w produkcie IBM WebSphere MQ są następujące:

Zmniejsz liczbę połączeń klienta MQTT z produktem WebSphere MQ .
Korzystając z hierarchii demonów, można połączyć wiele klientów z produktem WebSphere MQ; więcej klientów niż jeden menedżer kolejek może nawiązać połączenie jednocześnie.

Przechowywanie i przekazywanie komunikatów między klientami MQTT a produktem WebSphere MQ
Jeśli klienty nie mają własnej pamięci masowej, można użyć sklepu i dalej, aby uniknąć utrzymywania ciągłości połączeń między klientami a produktem IBM WebSphere MQ. Istnieje możliwość użycia wielu typów połączeń między klientem MQTT a produktem WebSphere MQ. Patrz sekcja Pojęcia telemetryczne i scenariusze monitorowania i sterowania.

Filtrowanie publikacji wymienianych między klientami MQTT a produktem WebSphere MQ .
Często publikacje dzielą się na komunikaty przetwarzane lokalnie i komunikaty, które obejmują inne aplikacje. Publikacje lokalne mogą obejmować przepływy sterowania między czujnikami i aktywatorami, a publikacje zdalne obejmują żądania odczytów, statusu i komend konfiguracji.

Zmiana obszarów tematów publikacji
Unikaj łańcuchów tematów z klientów przyłączonych do różnych portów nasłuchiwania przed zderzeniem się ze sobą. Przykład używa demona do oznaczania odczytów liczników pochodzących z różnych budynków. Patrz sekcja Separacja obszarów tematów różnych grup klientów.

Przykładowe ustawienia tematu dla mostu

Opublikuj wszystko w zdalnym brokerze-przy użyciu wartości domyślnych

Domyślnym kierunkiem jest out(wyjście), a most publikuje tematy w zdalnym brokerze. Parametr topic określa, które tematy są propagowane przy użyciu filtrów tematów.

Most używa parametru topic w produkcie Rysunek 31 na stronie 159 do subskrybowania wszystkiego, co zostało opublikowane w lokalnym demonie przez klienty MQTT lub przez inne brokery. Most publikuje tematy w zdalnym brokerze połączonym przez most.

```
connection Daemon1
topic #
```

Rysunek 31. Opublikuj wszystko w zdalnym brokerze

Opublikuj wszystko w zdalnym brokerze-jawne

Ustawienie topic w poniższym fragmencie kodu daje taki sam wynik, jak przy użyciu wartości domyślnych. Jedyna różnica polega na tym, że parametr **direction** jest jawny. Użyj kierunku `out` , aby zasubskrybować lokalny broker, demon i opublikować go w zdalnym brokerze. Publikacje utworzone w ramach demona lokalnego, do którego został zasubskrybowany most, są publikowane w zdalnym brokerze.

```
connection Daemon1
topic # out
```

Rysunek 32. Opublikuj wszystko w zdalnym brokerze-jawne

Opublikuj wszystko w lokalnym brokerze

Zamiast kierunku `out`, można ustawić kierunek odwrotny, `in`. Poniższy fragment kodu konfiguruje most w taki sposób, aby zasubskrybował wszystko, co zostało opublikowane w zdalnym brokerze połączonym przez most. Most publikuje tematy w lokalnym brokerze, demon.

```
connection Daemon1
topic # in
```

Rysunek 33. Opublikuj wszystko w lokalnym brokerze

Opublikuj wszystko z tematu eksportu w lokalnym brokerze do tematu importowania w brokerze zdalnym

W celu zmodyfikowania filtru tematu `#` w poprzednich przykładach należy użyć dwóch dodatkowych parametrów tematu, **`local_prefix`** i **`remote_prefix`**. Jeden z parametrów jest używany do modyfikowania filtru tematu używanego w subskrypcji, a drugi parametr służy do modyfikowania tematu, w którym publikowana jest publikacja. Ten efekt polega na zastąpieniu początku łańcucha tematu używanego w jednym brokerze z innym łańcuchem tematu w innym brokerze.

W zależności od kierunku komendy tematu znaczenie ma znaczenie **`local_prefix`** i **`remote_prefix`** . Jeśli kierunek to `out`, wartość domyślna **`local_prefix`** jest używana jako część subskrypcji tematu, a produkt **`remote_prefix`** zastępuje część **`local_prefix`** łańcucha tematu w zdalnej publikacji. If the direction is `in`, **`remote_prefix`** becomes part of the remote subscription, and **`local_prefix`** replaces the **`remote_prefix`** part of the topic string.

Pierwsza część łańcucha tematu jest często uznaana za definiującą obszar tematu. Aby zmienić obszar tematu, do którego jest publikowany temat, należy użyć dodatkowych parametrów. Można to zrobić, aby uniknąć propagowanego tematu kolidującego z innym tematem w brokerze docelowym lub aby usunąć łańcuch tematu punktu podłączenia.

Na przykład w poniższym fragmencie kodu wszystkie publikacje dotyczące łańcucha tematu `export/#` w demonie są ponownie publikowane w produkcie `import/#` w zdalnym brokerze.

```
topic # out export/ import/
```

Rysunek 34. Opublikuj wszystko z tematu eksportu w lokalnym brokerze do tematu importowania w brokerze zdalnym

Opublikuj wszystko w temacie importu w lokalnym brokerze z tematu eksportowania w zdalnym brokerze

Poniższy fragment kodu przedstawia zmianę konfiguracji. Most subskrybuje wszystko, co zostało opublikowane za pomocą łańcucha tematu `export/#` w zdalnym brokerze i publikuje je w produkcie `import/#` w lokalnym brokerze.

```
connection Daemon1
topic # in import/ export/
```

Rysunek 35. Opublikuj wszystko w temacie importu w lokalnym brokerze z tematu eksportowania w zdalnym brokerze

Opublikuj wszystko z punktu podłączenia 1884/ do zdalnego brokera przy użyciu oryginalnych łańcuchów tematów

W poniższym fragmencie kodu most subskrybuje wszystkie opublikowane przez klienty połączenia z punktem podłączenia 1884/ w lokalnym demonie. Most publikuje wszystko, co zostało opublikowane w punkcie podłączenia do zdalnego brokera. Łańcuch punktu podłączenia 1884/ jest

usuwany z tematów opublikowanych w brokerze zdalnym. *przedrostek_lokalny* jest taki sam, jak łańcuch punktu podłączenia 1884/, a *przedrostek_zdalny* jest pustym łańcuchem.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Rysunek 36. Opublikuj wszystko z punktu podłączenia 1884/ w zdalnym brokerze przy użyciu oryginalnych łańcuchów tematów.

Oddzielanie obszarów tematów różnych klientów podłączonych do różnych demonów

Załóżmy, że aplikacja jest napisana dla liczników energii elektrycznej, aby publikować odczyty liczników dla budynku. Odczyty są publikowane za pomocą klientów MQTT do demona udostępnianego w tym samym budynku. Temat wybrany dla publikacji to `power`. Ta sama aplikacja jest wdrażana w wielu budynkach w kompleksie. W przypadku monitorowania ośrodka i przechowywania danych odczyty ze wszystkich budynków są agregowane przy użyciu połączeń mostu. Połączenia łączą demony budynku z produktem WebSphere MQ w centralnym miejscu.

Identyczna aplikacja kliencka jest używana we wszystkich budynkach. Ta aplikacja jest publikowana w temacie `power`. Jednak dane muszą być zróżnicowane w zależności od budynku. Jest to wykonywane przez demona dla każdego budynku, który dodaje numer budynku jako przedrostek do nazwy tematu. Most od pierwszego budynku w kompleksie korzysta z przedrostka `meters/building01/`, od budynku dwa przedrostek to `meters/building02/`. Odczyty z innych budynków są zgodne z tym samym wzorcem. Produkt WebSphere MQ odbiera więc odczyty z tematami, takimi jak `meters/building01/power`.

Plik konfiguracyjny dla każdego demona zawiera instrukcję tematu, która jest zgodna z wzorcem w następującym fragmencie kodu:

```
connection Daemon1
topic power out "" meters/building01/
```

Rysunek 37. Oddziel obszary tematyczne klientów podłączonych do różnych demonów

W poprzednim fragmencie kodu pusty łańcuch jest obiektem zastępczym dla nieużywanego parametru local_prefix.

Uwaga: Ten przykład jest nieco sztuczny i jest przeznaczony tylko jako ilustracja. W praktyce obszar tematu, w którym aplikacja jest publikowana, może być konfigurowalny.

Oddziel obszary tematyczne klientów połączonych z tym samym demonem.

Załóżmy, że pojedynczy demon jest używany do połączenia wszystkich liczników zasilania. Zakładając, że w aplikacji można skonfigurować połączenie z różnymi portami, można odróżnić budynki, podłączając liczniki z różnych budynków do różnych portów nasłuchiwania, tak jak w poniższym fragmencie kodu. W tym przykładzie pokazano, w jaki sposób można użyć punktów podłączenia.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Rysunek 38. Oddziel obszary tematyczne klientów połączonych z tym samym demonem.

Ponownie odwzoruj różne tematy dla publikacji płynących w obu kierunkach

W konfiguracji w następującym fragmencie kodu most subskrybuje pojedynczy temat b w zdalnym brokerze i przekazuje publikacje dotyczące produktu b do demona lokalnego, zmieniając temat na a. Most subskrybuje również pojedynczy temat x w lokalnym brokerze i przekazuje publikacje dotyczące produktu x do brokera zdalnego, zmieniając temat na y.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Rysunek 39. Ponownie odwzoruj różne tematy dla publikacji płynących w obu kierunkach

Ważnym punktem w tym przykładzie jest to, że różne tematy są subskrybowane i publikowane w obu brokerach. Obszary tematów w obu brokerach są rozłączne.

Ponownie odwzoruj te same tematy na publikacje, które przepływają w obu kierunkach (pętla)

W przeciwieństwie do poprzedniego przykładu, konfiguracja w produkcie Rysunek 40 na stronie 162, ogólnie, powoduje wystąpienie pętli. W instrukcji tematu `topic "" in a b`most subskrybuje produkt b zdalnie, a następnie publikuje do produktu a lokalnie. W drugiej instrukcji tematu most subskrybuje produkt a lokalnie, a następnie publikuje w produkcie b zdalnie. Tę samą konfigurację można zapisać w sposób przedstawiony w sekcji Rysunek 41 na stronie 162.

Ogólnym wynikiem jest to, że jeśli klient publikuje zdalnie produkt b , to publikacja jest przesyłana do demona lokalnego jako publikacja w temacie a. Jednak w przypadku publikowania przez most do lokalnego demona w temacie a, publikacja jest zgodna z subskrypcją udostępnianej przez most do tematu lokalnego a. Subskrypcja to `topic "" out a b`. W wyniku tego publikacja jest przesyłana z powrotem do brokera zdalnego jako publikacja w temacie b. Most jest teraz zasubskrybowany do tematu zdalnego b, a cykl rozpoczyna się ponownie.

Niektóre brokery implementują wykrywanie pętli, aby zapobiec sytuacji w pętli. Mechanizm wykrywania pętli musi działać, gdy różne typy brokery są połączone mostem. Wykrywanie pętli nie działa, jeśli produkt WebSphere MQ jest połączony z demonem produktu WebSphere MQ Telemetry dla urządzeń. Wykonuje on pracę, jeśli dwa demony produktu IBM WebSphere MQ Telemetry dla urządzeń są połączone mostem. Domyślnie funkcja wykrywania pętli jest włączona; patrz try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

*Rysunek 40. **!**Ponownie odwzoruj te same tematy na publikacje, które przepływają w obu kierunkach*

```
connection Daemon1
topic "" both a b
```

*Rysunek 41. **!**Ponownie odwzoruj te same tematy na publikacje, które przepływają w obu kierunkach, korzystając z produktu `both`.*

Konfiguracja w programie Rysunek 39 na stronie 162 jest taka sama, jak Rysunek 40 na stronie 162.

Dostępność połączeń mostu IBM WebSphere MQ Telemetry daemon for devices

Skonfiguruj wiele adresów połączeń mostu IBM WebSphere MQ Telemetry daemon for devices , aby połączyć się z pierwszym dostępnym zdalnym brokerem. Jeśli broker jest menedżerem kolejek z wieloma

instancjami, podaj oba jego adresy TCP/IP. Skonfiguruj połączenie podstawowe w celu nawiązania połączenia lub ponownego nawiązania połączenia z serwerem głównym, gdy jest on dostępny.

Parametr mostu połączenia, adresy, jest listą adresów gniazd TCP/IP. Most próbuje połączyć się z każdym adresem z kolei, dopóki nie powiedzie się połączenie. Parametry połączenia round_robin i start_type sterują sposobem korzystania z adresów po pomyślnym nawiązaniu połączenia.

Jeśli parametr start_type ma wartość auto, manuallub lazy, to jeśli połączenie nie powiedzie się, most próbuje ponownie nawiązać połączenie. Wykorzystuje on każdy adres z kolei, przy czym około 20 sekund opóźnienia między każdą próbą połączenia. Jeśli parametr `start_type` ma wartość once, to jeśli połączenie nie powiedzie się, most nie podejmie próby automatycznego ponownego nawiązania połączenia.

Jeśli parametr round_robin ma wartość true, próby nawiązania połączenia z mostem rozpoczynają się od pierwszego adresu na liście, a każdy adres na liście z kolei próbuje się z nim znaleźć. Zaczyna się na pierwszym adresie ponownie, gdy lista jest wyczerpana. Jeśli na liście znajduje się tylko jeden adres, będzie on próbował to powtórzyć co 20 sekund.

Jeśli parametr round_robin ma wartość false, pierwszeństwo ma pierwszy adres na liście, który jest nazywany serwerem głównym. Jeśli pierwsza próba nawiązania połączenia z serwerem głównym nie powiedzie się, most będzie kontynuował próby ponownego nawiązania połączenia z serwerem głównym w tle. W tym samym czasie most próbuje połączyć się z innymi adresami na liście. Gdy tło próbuje nawiązać połączenie z serwerem głównym, most rozłącza się z bieżącym połączeniem i przełącza się na połączenie z serwerem podstawowym.

Jeśli połączenie zostanie odłączone dobrowolnie, na przykład za pomocą komendy **`connection_stop`** , po zrestartowaniu połączenia próbuje użyć tego samego adresu ponownie. Jeśli połączenie zostanie odłączone z powodu awarii połączenia lub do zdalnego brokera, który usuwa połączenie, most czeka 20 sekund. Następnie próbuje połączyć się z następnym adresem na liście lub pod tym samym adresem, jeśli na liście znajduje się tylko jeden adres.

Nawiązanie połączenia z menedżerem kolejek z wieloma instancjami

W konfiguracji menedżera kolejek z wieloma instancjami menedżer kolejek jest uruchamiany na dwóch różnych serwerach o różnych adresach IP. Zazwyczaj kanały telemetryczne są konfigurowane bez określonego adresu IP. Są one konfigurowane tylko z numerem portu. Gdy kanał pomiarowy jest uruchomiony, domyślnie wybiera pierwszy dostępny adres sieciowy na serwerze lokalnym.

Skonfiguruj parametr adresy połączenia mostu przy użyciu dwóch adresów IP używanych przez menedżer kolejek. Ustaw wartość round_robin na wartość true.

Jeśli aktywna instancja menedżera kolejek nie powiedzie się, menedżer kolejek przełącza się do instancji rezerwowej. Demon wykryje, że połączenie z aktywną instancją zostało zerwane i próbuje ponownie nawiązać połączenie z instancją rezerwową. Używa on innego adresu IP na liście adresów skonfigurowanych dla połączenia mostu.

Menedżer kolejek, z którym łączy się most, nadal jest tym samym menedżerem kolejek. Menedżer kolejek odzyskuje swój własny stan. Jeśli opcja oczyszczanie jest ustawiona na wartość false, sesja połączenia mostu jest odtwarzana do stanu sprzed przełączenia awaryjnego. Połączenie jest wznawiane po opóźnieniu. Komunikaty z "co najmniej raz" lub "co najwyżej raz" jakością usługi nie są tracone, a subskrypcje działają dalej.

Czas ponownego nawiązania połączenia zależy od liczby kanałów i klientów, które są restartowe w momencie uruchomienia instancji rezerwowej, a także od liczby komunikatów w stanie zapalnych. Połączenie mostu może próbować ponownie połączyć się z obydwoma adresami IP, ile razy przed ponownym nawiązaniem połączenia.

Nie należy konfigurować kanału telemetrycznego menedżera kolejek z wieloma instancjami przy użyciu konkretnego adresu IP. Adres IP jest poprawny tylko na jednym serwerze.

Jeśli używane jest alternatywne rozwiązanie wysokiej dostępności, które zarządza adresem IP, może być poprawne skonfigurowanie kanału telemetrycznego o określonym adresie IP.

czyszczenie (cleansession)

Połączenie mostu jest sesją klienta MQTT v3 . Można określić, czy połączenie rozpoczyna nową sesję, czy też odtwarza istniejącą sesję. Jeśli odtwarza ona istniejącą sesję, połączenie mostu zachowuje subskrypcje i zachowane publikacje z poprzedniej sesji.

Nie należy ustawiać wartości oczyszczanie na wartość false, jeśli adresy zawierają wiele adresów IP, a adresy IP łączą się z kanałami telemetrycznych udostępnianych przez różne menedżery kolejek lub do różnych demonów telemetrycznych. Stan sesji nie jest przesyłany między menedżerami kolejek lub demonami. Próba zrestartowania istniejącej sesji w innym menedżerze kolejek lub w wyniku uruchomienia nowej sesji spowoduje uruchomienie nowej sesji. Komunikaty wątpliwe są tracone, a subskrypcje mogą nie działać zgodnie z oczekiwaniami.

powiadomienia

Aplikacja może śledzić, czy połączenie mostu jest uruchomione za pomocą powiadomień. Powiadomienie jest publikacją o wartości 1, połączonej lub 0, która została rozłączona. Jest on publikowany w parametrze *topicString* zdefiniowanym przez parametr `notification_topic` . Wartością domyślną parametru *topicString* jest $SYS/`broker`/`connection`/ *clientIdentifier*/`state`. Wartość domyślna *topicString* zawiera przedrostek $SYS. Subskrybuj tematy rozpoczynające się od $SYS , definiując filtr tematów zaczynający się od $SYS. Filtr tematów #, zasubskrybuj wszystko, nie subskrybuje tematy rozpoczynające się od $SYS w demonie. Należy pomyśleć o $SYS , ponieważ definiując specjalny obszar tematu systemu, który różni się od obszaru tematu aplikacji.

Powiadomienia umożliwiają IBM WebSphere MQ Telemetry daemon for devices powiadamianie klientów MQTT, gdy most jest podłączony lub odłączony.

keepalive_interval

Parametr połączenia mostu keepalive_interval ustawia przedział czasu między mostem wysyłającym komendę ping TCP/IP na serwer zdalny. Domyślny okres sprawdzania wynosi 60 sekund. Komenda ping zapobiega zamknięciu sesji TCP/IP przez serwer zdalny lub przez firewall, który wykrywa okres nieaktywności w połączeniu.

CLIENTID

Połączenie mostu jest sesją klienta MQTT v3 i ma `clientIdentifier` ustawioną za pomocą parametru połączenia mostu clientid. Jeśli planowane jest wznawianie poprzedniej sesji przez ustawienie parametru cleansession na wartość false, wartość parametru `clientIdentifier` używana w każdej sesji musi być taka sama. Wartością domyślną parametru clientid jest *hostname.connectionName*, która pozostaje taka sama.

Instalowanie, weryfikowanie, konfigurowanie i sterowanie demonem produktu WebSphere MQ Telemetry dla urządzeń

Instalacja, konfiguracja i sterowanie demonem jest oparte na plikach.

Zainstaluj demona, kopiując pakiet Software Development Kit do urządzenia, na którym ma być uruchomiony demon.

Jako przykład uruchom program narzędziowy klienta MQTT i nawiąże połączenie z demonem WebSphere MQ Telemetry dla urządzeń jako broker publikowania/subskrypcji. Patrz sekcja Korzystanie z demona WebSphere MQ Telemetry dla urządzeń jako broker publikowania/subskrypcji.

Skonfiguruj demon, tworząc plik konfiguracyjny. Patrz sekcja WebSphere MQ Telemetry daemon for devices configuration file(Demon produktu WebSphere MQ Telemetry dla urządzeń).

Sterowanie uruchomionym demonem poprzez tworzenie komend w pliku `amqtdd.upd`. Co 5 sekund demon odczytuje ten plik, uruchamia komendy i usuwa plik. Patrz sekcja WebSphere MQ Telemetry daemon for devices command file(Demon WebSphere MQ Telemetry dla urządzeń-plik komend).

Demon WebSphere MQ Telemetry dla portów nasłuchiwania urządzeń

Połącz klienty MQTT V3 z demonem WebSphere MQ Telemetry dla urządzeń korzystających z portów nasłuchiwania. Port nasłuchiwania można kwalifikować za pomocą punktu podłączenia i maksymalnej liczby połączeń.

Port nasłuchiwania musi odpowiadać numerowi portu określonym w metodzie `connect(serverURI)` klienta MQTT klienta łączącego się z tym portem. Wartością domyślną jest zarówno klient, jak i demon 1883.

Domyślny port dla demona można zmienić, ustawiając globalną definicję `port` w pliku konfiguracyjnym demona. Konkretne porty można ustawić, dodając definicję `nasłuchiwania` do pliku konfiguracyjnego demona.

Dla każdego portu nasłuchiwania, innego niż port domyślny, można określić punkt podłączenia w celu odizolowania klientów. Klienty połączone z portem z punktem podłączenia są odizolowane od innych klientów. Patrz sekcja “Demon WebSphere MQ Telemetry dla punktów podłączenia urządzeń” na stronie 165.

Można ograniczyć liczbę klientów, które mogą łączyć się z dowolnym portem. Ustaw globalną definicję `max_connections` , aby ograniczyć połączenia do domyślnego portu, lub kwalifikuj każdy port nasłuchiwania przy użyciu parametru `max_connections`.

Przykład

Przykład pliku konfiguracyjnego, który zmienia domyślny port z 1883 na 1880i ogranicza połączenia z portem 1880 do 10000. Połączenia z portem 1884 są ograniczone do 1000. Klienty przyłączone do portu 1884 są odizolowane od klientów przyłączonych do innych portów.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

Demon WebSphere MQ Telemetry dla punktów podłączenia urządzeń

Punkt podłączenia można powiązać z portem nasłuchiwania używanym przez klienty MQTT w celu nawiązania połączenia z demonem produktu WebSphere MQ Telemetry dla urządzeń. Punkt podłączenia izoluje publikacje i subskrypcje wymieniane przez klienty MQTT przy użyciu jednego portu nasłuchiwania z klientów MQTT połączonych z innym portem nasłuchiwania.

Klienty przyłączone do portu nasłuchiwania z punktem podłączenia nie mogą nigdy bezpośrednio wymieniać tematów z klientami przyłączonymi do żadnego innego portu nasłuchiwania. Klienty przyłączone do portu nasłuchiwania bez punktu podłączenia mogą publikować lub subskrybować tematy dowolnego klienta. Klienci nie są świadomi tego, czy są one przyłączone za pośrednictwem punktu podłączenia, czy też nie. Nie ma to znaczenia dla łańcuchów tematów utworzonych przez klienty.

Punkt podłączenia jest łańcuchem tekstowym, który jest poprzedzony łańcuchem tematu publikacji i subskrypcji. Jest ona wstępna do wszystkich łańcuchów tematów utworzonych przez klienty przyłączone do portu nasłuchiwania z punktem podłączenia. Łańcuch tekstu jest usuwany ze wszystkich łańcuchów tematów wysyłanych do klientów przyłączonych do portu nasłuchiwania.

Jeśli port nasłuchiwania nie ma punktu podłączenia, łańcuchy tematów publikacji i subskrypcji utworzonych i odebranych przez klienty przyłączone do portu nie są zmieniane.

Utwórz łańcuchy punktów podłączenia na końcu /. W ten sposób punkt podłączenia jest tematem nadrzędnym drzewa tematów dla punktu podłączenia.

Przykład

Plik konfiguracyjny zawiera następujące porty nasłuchiwania:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

Klient, przyłączony do portu 1883, tworzy subskrypcję programu `MyTopic`. Demon rejestruje subskrypcję jako `1883/MyTopic`. Inny klient przyłączony do portu 1883 publikuje komunikat w temacie `MyTopic`. Demon zmienia łańcuch tematu na `1883/MyTopic` i wyszukuje zgodne subskrypcje. Subskrybent na porcie 1883 odbiera publikację z oryginalnym łańcuchem tematu `MyTopic`. Demon usunł przedrostek punktu podłączenia z łańcucha tematu.

Inny klient, przyłączony do portu 1884, publikuje również w temacie `MyTopic`. Tym razem demon rejestruje temat jako `1884/MyTopic`. Subskrybent na porcie 1883 nie otrzymuje publikacji, ponieważ inny punkt podłączenia jest wynikiem subskrypcji z innym łańcuchem tematu.

Klient przyłączony do portu 1885publikuje w temacie `1883/MyTopic`informacje na temat tego tematu. Demon nie zmienia łańcucha tematu. Subskrybent na porcie 1883 otrzymuje publikację do produktu `MyTopic`.

Demon WebSphere MQ Telemetry dla urządzeń jakości usług, trwałych subskrypcji i zachowanych publikacji

Ustawienia jakości usług mają zastosowanie tylko do działającego demona. Jeśli demon zatrzyma się, niezależnie od tego, czy w sposób kontrolowany, czy z powodu awarii, stan komunikatów w stanie inflight jest tracony. Dostarczenie komunikatu co najmniej raz lub co najwyżej raz nie może być gwarantowane, jeśli demon zostanie zatrzymany. Demon WebSphere MQ Telemetry dla urządzeń obsługuje ograniczoną trwałość. Ustaw parametr konfiguracyjny **`retained_persistence`** , aby zachować zachowane publikacje i subskrypcje, gdy demon jest wyłączony.

W przeciwieństwie do produktu WebSphere MQ, demon WebSphere MQ Telemetry dla urządzeń nie kroniuje trwałych danych. Stan sesji, stan komunikatu i zachowane publikacje nie są zapisywane transakcyjnie. Domyślnie demon usuwa wszystkie dane po zatrzymaniu. Istnieje możliwość ustawienia opcji okresowo powtarzających się subskrypcji i zachowanych publikacji. Status komunikatu jest zawsze tracony, gdy demon zatrzyma się. Wszystkie publikacje niezachowane są tracone.

Ustaw opcję konfiguracyjna demona, `Retained_persistence` na `true`, aby okresowo składować przechowywane publikacje do pliku. Gdy demon zostanie zrestartowany, zachowane publikacje, które zostały ostatnio automatycznie wprowadzone, są ponownie włączone. Domyślnie zachowane komunikaty utworzone przez klienty nie są przywrócone po restarcie demona.

Ustaw opcję konfigurującą demona `Retained_persistence` na wartość `true`, aby zapisywać subskrypcje utworzone w trwałej sesji okresowo do pliku. Jeśli parametr `Retained_persistence` ma wartość `true`, subskrypcje tworzone przez klienty w sesji z ustawioną na wartość `CleanSession` ustawioną na wartość `false`("trwała sesja") są odtwarzane. Demon odtwarza subskrypcje po restarcie, które rozpoczynają odbieranie publikacji. Klient otrzymuje publikacje po jego zrestartowaniu z produktem `CleanSession` na wartość `false`. Domyślnie stan sesji klienta nie jest zapisywany, gdy demon zatrzyma się, a więc subskrypcje nie są odtwarzane, nawet jeśli klient ustawia wartość `CleanSession` na `false`.

`Retained_persistence` -mechanizm autosave. Może nie zapisywać najnowszych zachowanych publikacji ani subskrypcji. Użytkownik może zmienić częstotliwość zapisywania zachowanych publikacji i subskrypcji. Ustaw odstęp czasu między składowaniem lub liczbę zmian między składowaniem, korzystając z opcji konfiguracyjnych `autosave_on_changes` i `autosave_interval`.

Przykładowa konfiguracja do ustawiania trwałości

```
# 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
```

Demon WebSphere MQ Telemetry dla bezpieczeństwa urządzeń

Demon WebSphere MQ Telemetry dla urządzeń może uwierzytelniać klientów, które łączą się z nim, używać referencji w celu łączenia się z innymi brokerami oraz kontrolować dostęp do tematów. Zabezpieczenia demona są ograniczone, ponieważ są one budowane przy użyciu klienta WebSphere MQ Telemetry C, który nie zapewnia obsługi SSL. W związku z tym połączenia z i z demona nie są szyfrowane i nie mogą być uwierzytelniane za pomocą certyfikatów.

Domyślnie zabezpieczenia nie są włączone.

Uwierzytelnianie klientów

Klienty MQTT mogą ustawiać nazwę użytkownika i hasło przy użyciu metod `MqttConnectOptions.setUserName` i `MqttConnectOptions.setPassword`.

Uwierzytelnij klienta, który łączy się z demonem, sprawdzając nazwę użytkownika i hasło podane przez klienta w odniesieniu do wpisów w pliku haseł. Aby włączyć uwierzytelnianie, należy utworzyć plik haseł i ustawić parametr `password_file` w pliku konfiguracyjnym demona; patrz password_file.

Ustaw parametr `allow_anonymous` w pliku konfiguracyjnym demona, aby umożliwić klientom łączenie się bez nazw użytkowników lub haseł w celu nawiązania połączenia z demonem, który sprawdza uwierzytelnianie. Patrz allow_anonymous. Jeśli klient udostępni nazwę użytkownika lub hasło, zawsze jest on sprawdzany przy użyciu pliku haseł, jeśli ustawiono parametr `password_file` .

Ustaw parametr `clientid_prefixes` w pliku konfiguracyjnym demona, aby ograniczyć połączenia do konkretnych klientów. Klienty muszą mieć wartość `clientIdentifiers` , która rozpoczyna się od jednej z przedrostków wymienionych w parametrze `clientid_prefixes` . Patrz clientid_prefixes.

Zabezpieczenia połączenia mostu

Każdy demon WebSphere MQ Telemetry dla połączenia mostu urządzeń jest klientem MQTT V3 . Nazwę użytkownika i hasło dla każdego połączenia mostu można ustawić jako parametr połączenia mostu w pliku konfiguracyjnym demona. Patrz: nazwa_użytkownika i hasło. Most może następnie uwierzytelnić się w brokerze.

Kontrola dostępu do tematów

Jeśli klienci są uwierzytelniani, demon może również zapewnić kontrolę dostępu do tematów dla każdego użytkownika. Demon nadaje kontrolę dostępu na podstawie zgodnego tematu, którego klient publikuje lub subskrybuje łańcuch tematu dostępu w pliku kontroli dostępu. Patrz acl_file.

Lista kontroli dostępu ma dwie części. Pierwsza część steruje dostępem dla wszystkich klientów, w tym klientów anonimowych. Druga część zawiera sekcję dla dowolnego użytkownika w pliku haseł. Zawiera on konkretne elementy kontroli dostępu dla każdego użytkownika.

Przykład

Parametry zabezpieczeń są przedstawione w poniższym przykładzie.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

Rysunek 42. Plik konfiguracyjny demona

```
Fred:Fredpassword
Barney:Barneypassword
```

Rysunek 43. Plik haseł 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/#
```

Rysunek 44. Plik kontroli dostępu, acl.txt

Rozwiązywanie problemów z klientami MQTT

Poszukaj zadania rozwiązywania problemów, które pomoże rozwiązać problem z uruchomionym klientem MQTT .

Zadania pokrewne

“Śledzenie i debugowanie klienta Java MQTT (Paho)” na stronie 177
Domyślny program rejestrujący korzysta ze standardowego mechanizmu rejestrowania Java, który jest znany jako `java.util.logging` (JSR47). Można go skonfigurować przy użyciu pliku konfiguracyjnego lub programowo.

“Śledzenie klienta MQTT JavaScript” na stronie 180
Klienta JavaScript można użyć do gromadzenia danych śledzenia przez zmianę aplikacji WWW klienta w celu wywołania metod na podłączonym obiekcie klienta.

“Śledzenie usługi telemetrycznej (MQXR)” na stronie 173
Wykonaj poniższe instrukcje, aby uruchomić śledzenie usługi telemetrycznej, ustaw parametry sterujące śledzeniem i odszukaj dane wyjściowe śledzenia.

“Śledzenie klienta Java MQTT v3” na stronie 174
Wykonaj poniższe instrukcje, aby utworzyć śledzenie klienta Java MQTT i sterować jego danymi wyjściowymi.

“Śledzenie klienta MQTT dla języka C” na stronie 176
Ustaw zmienną środowiskową `MQTT_C_CLIENT_TRACE` , aby śledzić aplikację C klienta produktu MQTT .

“Rozwiązywanie problemu: klient MQTT nie łączy się z serwerem” na stronie 187
Rozwiąż problem z programem klienckim MQTT, który nie może nawiązać połączenia z usługą telemetryczną (MQXR).

“Rozwiązywanie problemu: połączenie klienta MQTT zostało usunięte” na stronie 189
Dowiedz się, co powoduje, że klient zgłasza nieoczekiwane wyjątki `ConnectionLost` po pomyślnym nawiązaniu połączenia i uruchomieniu go przez krótki lub długi czas.

“Rozwiązywanie problemów: utrata komunikatów w aplikacji MQTT” na stronie 189

Rozwiąż problem utraty komunikatu. Czy wiadomość jest nietrwała, wysłana do niewłaściwego miejsca, czy nigdy nie została wysłana? Błędnie zakodowany program kliencki może utracić komunikaty.

“Rozwiązywanie problemu: usługa telemetryczna (MQXR) nie jest uruchomiona” na stronie 191
Rozwiąż problem z usługą telemetryczną (MQXR), której uruchomienie nie powiodło się. Sprawdź, czy w instalacji produktu WebSphere MQ Telemetry nie brakuje plików, przeniesiono je lub czy mają one niepoprawne uprawnienia. Sprawdź ścieżki używane przez usługę telemetryczną (MQXR), która znajduje się w programach usługowych telemetrycznych (MQXR).

“Rozwiązywanie problemu: moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną” na stronie 193
Sprawdź, czy moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną (MQXR), a następnie skonfiguruj usługę JAAS w celu usunięcia problemu.

“Rozwiązywanie problemu: uruchamianie lub uruchamianie demona” na stronie 196
Zapoznaj się z dziennikiem konsoli produktu IBM WebSphere MQ Telemetry dla urządzeń, włącz śledzenie lub skorzystaj z tabeli objawów w tym temacie, aby rozwiązać problemy z demonem.

“Rozwiązywanie problemu: klienty MQTT nie nawiązują połączenia z demonem” na stronie 197
Klienty nie łączą się z demonem, a demon nie łączy się z innymi demonami ani z kanałem telemetryczny WebSphere MQ .

Odsyłacze pokrewne

“Położenie dzienników telemetrycznych, dzienników błędów i plików konfiguracyjnych” na stronie 169
W tym miejscu znajdują się dzienniki, dzienniki błędów i pliki konfiguracyjne używane przez produkt IBM WebSphere MQ Telemetry.

“Kody przyczyn klienta Java MQTT v3” na stronie 172
Sprawdź przyczyny kodów przyczyny w wyjątku klienta Java MQTT v3 lub w funkcji throwable.

“Wymagania systemowe dotyczące korzystania z zestawów algorytmów szyfrowania SHA-2 z klientami MQTT” na stronie 180
W przypadku środowiska Java 6 z wersji IBM, SR13 , można użyć zestawów algorytmów szyfrowania SHA-2 w celu zabezpieczenia kanałów i aplikacji klienckich produktu MQTT . Jednak zestawy algorytmów szyfrowania SHA-2 nie są domyślnie włączone, dopóki środowisko Java 7 nie będzie od wersji IBM, SR4 , tak więc we wcześniejszych wersjach należy określić wymagany pakiet. Jeśli używany jest klient MQTT z własnym środowiskiem JRE, należy upewnić się, że obsługuje on zestawy algorytmów szyfrowania SHA-2 . Aby aplikacje klienckie korzystały z zestawów algorytmów szyfrowania SHA-2 , klient musi również ustawić kontekst SSL na wartość obsługując protokół TLS (Transport Layer Security) w wersji 1.2.

“Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL” na stronie 181
Istnieją różnice między różnymi przeglądarkami, na różnych platformach. Zapoznanie się z tymi różnicami ułatwia konfigurowanie aplikacji, ośrodków certyfikacji (CA) i certyfikatów klienta w celu nawiązania połączenia przy użyciu protokołu Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i produktu WebSockets.

Położenie dzienników telemetrycznych, dzienników błędów i plików konfiguracyjnych

W tym miejscu znajdują się dzienniki, dzienniki błędów i pliki konfiguracyjne używane przez produkt IBM WebSphere MQ Telemetry.

Uwaga: Przykłady są kodowane dla systemu Windows. Zmień składnię tak, aby uruchamiała przykłady w systemie Linux .

Dzienniki po stronie serwera

Kreator instalacji produktu IBM WebSphere MQ Telemetry zapisuje komunikaty do swojego dziennika instalacji:

```
WMQ program directory\mqxr
```

Usługa telemetryczna (MQXR) zapisuje komunikaty w dzienniku błędów menedżera kolejek produktu WebSphere MQ i w plikach FDC do katalogu błędów produktu IBM WebSphere MQ :

```
WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQnnn.n.FDC
```

Zapisuje także dziennik dla usługi telemetrycznej (MQXR). W dzienniku są wyświetlane właściwości, z którymi usługa została uruchomiona, a także wystąpiły błędy, które zostały znalezione jako proxy dla klienta MQTT. Na przykład, anulowanie subskrypcji subskrypcji, której klient nie utworzył. Ścieżka do dziennika jest następująca:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

Przykładowa konfiguracja telemetryczna IBM WebSphere MQ utworzona przez program IBM WebSphere MQ Explorer uruchamia usługę telemetryczną przy użyciu komendy **runMQXRService**. **runMQXRService** znajduje się w *WMQ Telemetry install directory*\bin. Zapisuje do:

```
WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Zmodyfikuj **runMQXRService** , aby wyświetlić ścieżki skonfigurowane dla usługi telemetrycznej (MQXR), lub aby echo zainicjowano przed uruchomieniem usługi telemetrycznej (MQXR).

Pliki konfiguracyjne po stronie serwera

Kanały telemetryczne i telemetria (MQXR)

Ograniczenie: Format, położenie, treść i interpretacja pliku konfiguracyjnego kanału telemetrycznego mogą być zmieniane w przyszłych wersjach. Aby skonfigurować kanały telemetryczne, należy użyć programu IBM WebSphere MQ Explorer.

Program IBM WebSphere MQ Explorer zapisuje konfiguracje telemetryczne w pliku `mqxr_win.properties` w systemie Windowsoraz w pliku `mqxr_unix.properties` w systemie Linux. Pliki właściwości są zapisywane w katalogu konfiguracji telemetrycznej:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Rysunek 45. Katalog konfiguracji telemetrycznej w systemie Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Rysunek 46. Katalog konfiguracji telemetrycznej w systemie Linux

Maszyna JVM

Ustaw właściwości Java, które są przekazywane jako argumenty do usługi telemetrycznej (MQXR) w pliku `java.properties`. Właściwości znajdujące się w pliku są przekazywane bezpośrednio do maszyny JVM, na której działa usługa telemetryczna (MQXR). Są one przekazywane jako dodatkowe właściwości maszyny JVM w wierszu komend Java. Właściwości ustawione w wierszu komend mają pierwszeństwo przed właściwościami dodanym do wiersza komend z pliku `java.properties` .

Znajdź plik `java.properties` w tym samym folderze, w którym znajdują się konfiguracje telemetryczne, patrz Rysunek 45 na stronie 170 i Rysunek 46 na stronie 170.

Zmodyfikuj `java.properties` , określając każdą właściwość jako osobną linię. Każdą właściwość należy sformatować dokładnie tak, jak należy przekazać tę właściwość do maszyny JVM jako argument, na przykład:

```
-Xmx1024m
-Xms1024m
```

JAAS

Plik konfiguracyjny JAAS jest opisany w sekcji Konfiguracja JAAS kanału telemetrycznego , która zawiera przykładowy plik konfiguracyjny JAAS , JAAS.config, dostarczany wraz z produktem IBM WebSphere MQ Telemetry.

W przypadku skonfigurowania usługi JAASprawie na pewno będzie można napisać klasę w celu uwierzytelnienia użytkowników w celu zastąpienia standardowych procedur uwierzytelniania JAAS .

Aby uwzględnić klasę `Login` w ścieżce klasy używanej przez ścieżkę klasy usługi telemetrycznej (MQXR), należy podać plik konfiguracyjny produktu WebSphere MQ `service.env` .

Ustaw ścieżkę klasy dla JAAS `LoginModule` w produkcie `service.env`. Nie można użyć zmiennej `%classpath%` w produkcie `service.env`. Ścieżka klasy w produkcie `service.env` jest dodawana do ścieżki klasy już ustawionej w definicji usługi telemetrycznej (MQXR).

Wyświetl ścieżki klas, które są używane przez usługę telemetryczną (MQXR), przez dodanie `echo set classpath` do `runMQXRService.bat`. Dane wyjściowe są wysyłane do produktu `mqxr.stdout`.

Domyślnym położeniem pliku `service.env` jest:

```
WMQ data directory\service.env
```

Przesłoń te ustawienia za pomocą pliku `service.env` dla każdego menedżera kolejek w:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

Rysunek 47 na stronie 171 przedstawia przykładowy plik `service.env` , w którym można użyć przykładowego pliku `LoginModule.class`.

```
CLASSPATH=WMQ Install Directory\mqxr\samples
```

Uwaga: `service.env` nie może zawierać żadnych zmiennych. Zastąp rzeczywistą wartość *WMQ Install Directory*.

Rysunek 47. Przykład `service.env` dla systemu Windows

Śledzenie

Inżynier serwisu IBM może poprosić użytkownika o skonfigurowanie śledzenia; patrz “Śledzenie usługi telemetrycznej (MQXR)” na stronie 173. Parametry do skonfigurowanego śledzenia są przechowywane w dwóch plikach:

```
WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Pliki dziennika po stronie klienta

Domyślna klasa trwałości plików w kliencie Java SE MQTT dostarczonym razem z produktem IBM WebSphere MQ Telemetry tworzy folder o nazwie: *clientIdentifier*`-tcp`*hostNamePort* lub *clientIdentifier*`-ssl`*hostNamePort* w katalogu roboczym klienta. Nazwa folderu informuje o wartościach (`nazwaHosta` i `port`) użytych podczas próby nawiązania połączenia.. Folder zawiera komunikaty, które zostały zapisane przez klasę trwałości. Komunikaty są usuwane po pomyślnym dostarczyniu.

Folder jest usuwany, gdy klient, z czystą sesją, kończy działanie.

Jeśli śledzenie klienta jest włączone, niesformatowany dziennik jest domyślnie przechowywany w katalogu roboczym klienta. Plik śledzenia ma nazwę `mqtt-`*n*`.trc`

Pliki konfiguracyjne po stronie klienta

Ustaw właściwości śledzenia i SSL dla klienta Java MQTT przy użyciu plików właściwości Java lub ustaw właściwości programowo. Przekaz właściwości do klienta Java MQTT za pomocą przełącznika JVM `-D` : na przykład:

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
     -Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

Patrz “Śledzenie klienta Java MQTT v3” na stronie 174.W celu uzyskania odsyłaczy do dokumentacji interfejsu API klienta dla bibliotek klienta MQTT należy zapoznać się z informacjami w sekcji Skorowidz programistyczny klienta MQTT.

Kody przyczyn klienta Java MQTT v3

Sprawdź przyczyny kodów przyczyny w wyjątku klienta Java MQTT v3 lub w funkcji throwable.

Tabela 5. Kody przyczyn klienta Java MQTT v3

Kod przyczyny	Wartość	Przyczyna
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	Klient jest już połączony.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	Klient jest już rozłączony.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Zgłaszane, gdy próba wywołania `MqttClient.disconnect` została wykonana z poziomu metody w systemie `MqttCallback`.
REASON_CODE_CLIENT_DISCONNECTING	32102	Klient obecnie rozłącza się i nie może zaakceptować żadnej nowej pracy.
REASON_CODE_CLIENT_EXCEPTION	0	Klient napotkał wyjątek.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	Klient nie jest połączony z serwerem.
REASON_CODE_CLIENT_TIMEOUT	32000	Przekroczono limit czasu klienta podczas oczekiwania na odpowiedź z serwera.
REASON_CODE_FAILED_AUTHENTICATION	4	Uwierzytelnianie na serwerze nie powiodło się z powodu błędnej nazwy użytkownika lub hasła.
REASON_CODE_INVALID_CLIENT_ID	2	Serwer odrzucił podany identyfikator klienta.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	Żądana wersja protokołu nie jest obsługiwana przez serwer.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Błąd wewnętrzny, spowodowany przez brak dostępnych nowych identyfikatorów komunikatów.
REASON_CODE_NOT_AUTHORIZED	5	Brak uprawnień do wykonania żądanej operacji.
REASON_CODE_SERVER_CONNECT_ERROR	32103	Nie można nawiązać połączenia z serwerem.

Tabela 5. Kody przyczyn klienta Java MQTT v3 (kontynuacja)		
Kod przyczyny	**Wartość**	**Przyczyna**
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	Identyfikator URI serwera i dostarczona fabryka SocketFactory nie są zgodne.
REASON_CODE_SSL_CONFIG_ERROR	32106	Błąd konfiguracji SSL.
REASON_CODE_UNEXPECTED_ERROR	6	Wystąpił nieoczekiwany błąd.

Śledzenie usługi telemetrycznej (MQXR)

Wykonaj poniższe instrukcje, aby uruchomić śledzenie usługi telemetrycznej, ustaw parametry sterujące śledzeniem i odszukaj dane wyjściowe śledzenia.

Zanim rozpoczniesz

Śledzenie jest funkcją obsługi. Postępuj zgodnie z tymi instrukcjami, jeśli inżynier serwisu IBM poprosi o śledzenie usługi telemetrycznej (MQXR). Dokumentacja produktu nie dokumentuje formatu pliku śledzenia ani sposobu korzystania z niego w celu debugowania klienta.

O tym zadaniu

Do uruchamiania i zatrzymywania śledzenia produktu IBM WebSphere MQ można używać komend IBM WebSphere MQ **strmqtrc** i **endmqtrc** . Program **strmqtrc** przechwytuje dane śledzenia dla usługi telemetrycznej (MQXR). Jeśli używany jest produkt **strmqtrc** , przed uruchomieniem śledzenia usługi telemetrycznej istnieje opóźnienie do kilku sekund. Więcej informacji na temat śledzenia produktu IBM WebSphere MQ można znaleźć w sekcji Korzystanie ze śledzenia. Alternatywnie można śledzić usługę telemetryczną (MQXR), wykonując następującą procedurę:

Procedura

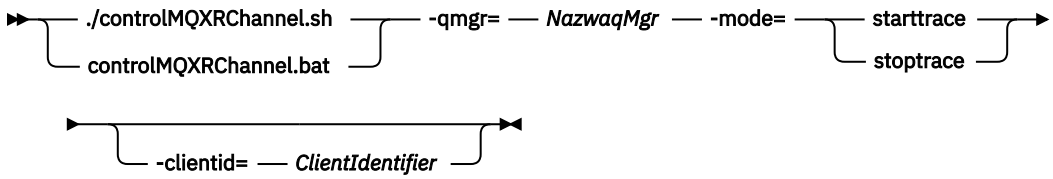
1. Ustaw opcje śledzenia, aby kontrolować ilość szczegółów i wielkość śledzenia. Opcje mają zastosowanie do śledzenia, które zostało uruchomione za pomocą komendy **strmqtrc** lub **controlMQXRChannel** .

   Ustaw opcje śledzenia w następujących plikach:

   ```
   mqxrtrace.properties
   trace.config
   ```

   Pliki znajdują się w katalogu:

   - On Windows, *WebSphere MQ data directory*\qmgrs*qMgrName* \mqxr.
   - On Linux, var/mqm/qmgrs/ *qMgrName*/mqxr.
2. Otwórz okno komend w następującym katalogu:
   - W systemach Windows *WebSphere MQ installation directory*\mqxr\bin .
   - W systemach Linux /opt/mqm/mqxr/bin.
3. Uruchom następującą komendę, aby uruchomić śledzenie produktu SYSTEM.MQXR.SERVICE :



Parametry obowiązkowe

qmgr=*qmgrName*
Ustaw parametr *qmgrName* na nazwę menedżera kolejek.

mode=starttrace| stoptrace
Ustaw parametr starttrace , aby rozpocząć śledzenie, lub stoptrace , aby zakończyć śledzenie

Parametry opcjonalne

clientid=*ClientIdentifier*
Ustaw parametr *ClientIdentifier* na wartość `ClientIdentifier` klienta. clientid filtruje dane śledzenia do pojedynczego klienta. Uruchom komendę śledzenia wiele razy, aby śledzić wiele klientów.

Na przykład:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid=
problemclient
```

Wyniki

Aby wyświetlić dane wyjściowe śledzenia, przejdź do następującego katalogu:

- On Windows , *WebSphere MQ data directory*\trace.
- On Linux, /var/mqm/trace .

Pliki śledzenia mają nazwę mqxr_PPPPP.trc, gdzie PPPPP jest identyfikatorem procesu.

Odsyłacze pokrewne

strmqtrc

Śledzenie klienta Java MQTT v3

Wykonaj poniższe instrukcje, aby utworzyć śledzenie klienta Java MQTT i sterować jego danymi wyjściowymi.

Zanim rozpoczniesz

Ten temat ma zastosowanie wyłącznie do produktu IBM WebSphere MQ w wersji 7.5.0.0. Informacje na temat śledzenia wersji klienta Java dla nowszych wersji zawiera sekcja “Śledzenie i debugowanie klienta Java MQTT (Paho)” na stronie 177.

Śledzenie jest funkcją obsługi. Postępuj zgodnie z tymi instrukcjami, jeśli inżynier serwisu IBM poprosi o śledzenie klienta Java MQTT. Dokumentacja produktu nie dokumentuje formatu pliku śledzenia ani sposobu korzystania z niego w celu debugowania klienta.

Śledzenie działa tylko w przypadku klienta Java produktu WebSphere MQ Telemetry .

O tym zadaniu

Uwaga: Przykłady są kodowane dla systemu Windows. Zmień składnię tak, aby uruchamiała przykłady w systemie Linux .[2].

Procedura

1. Utwórz plik właściwości Java zawierający konfigurację śledzenia.

   W pliku właściwości określ następujące właściwości opcjonalne. Jeśli klucz właściwości jest określony więcej niż jeden raz, to ostatnie wystąpienie ustawia właściwość.

[2] W języku Java jest używany poprawny separator ścieżki. Ogranicznik można zakodować w pliku właściwości jako '/' lub '\\'; '\' jest znakiem zmiany znaczenia.

a) `com.ibm.micro.client.mqttv3.trace.outputName`

Katalog, w którym ma zostać zapisane plik śledzenia. Wartością domyślną jest katalog roboczy klienta. Plik śledzenia nosi nazwę `mqtt-n.trc`.

```
java com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
```

b) `com.ibm.micro.client.mqttv3.trace.count`

Liczba plików śledzenia do zapisu. Wartością domyślną jest jeden plik o nieograniczonej wielkości.

```
java com.ibm.micro.client.mqttv3.trace.count=5
```

c) `com.ibm.micro.client.mqttv3.trace.limit`

Maksymalna wielkość pliku do zapisu, wartością domyślną jest 500000. Limit ma zastosowanie tylko wtedy, gdy zażądano więcej niż jednego pliku śledzenia.

```
java com.ibm.micro.client.mqttv3.trace.limit=100000
```

d) `com.ibm.micro.client.mqttv3.trace.client.`*`clientIdentifier`*`.status`

Włącz lub wyłącz śledzenie na kliencie. Jeśli *`clientIdentifier`*`=*`, śledzenie jest włączone lub wyłączone dla wszystkich klientów. Domyślnie śledzenie jest wyłączone dla wszystkich klientów.

```
java com.ibm.micro.client.mqttv3.trace.client.*.status=on
```

```
java com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
```

2. Przekaz plik właściwości śledzenia do maszyny JVM, korzystając z właściwości systemowej.

```
java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
```

3. Uruchom klienta.
4. Przekształć plik śledzenia z kodowania binarnego na tekst lub `.html`. Użyj następującej komendy:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o
outputFile] [-h] [-d
time]
```

gdzie argumenty są następujące:

-?
wyświetla pomoc

-i traceFile
Wymagane. Przechodzi do pliku wejściowego (na przykład `mqtt-0.trc`).

-o outputFile
Wymagane. Definiuje plik wyjściowy (na przykład `mqtt-0.trc.html` lub `mqtt-0.trc.txt`).

-h
Dane wyjściowe jako HTML. Rozszerzenie plików wyjściowych musi mieć wartość `.html`. Jeśli wartość nie zostanie określona, dane wyjściowe są zwykłym tekstem.

-d time
Umożliwia wcięcie linii z * , jeśli różnica czasu (w milisekundach) jest większa lub równa (> =). Nie dotyczy danych wyjściowych HTML.

Poniższy przykład spowoduje wyjście z pliku śledzenia w formacie HTML.

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.html -h
```

Drugi przykład spowoduje wyjście z pliku śledzenia jako zwykłego tekstu, z dowolnymi kolejnymi znacznikami czasu, które zawierają milisekundy z różnicą 50 lub większą, z gwiazdką (*).

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt -d 50
```

Ostatni przykład spowoduje wyjście z pliku śledzenia jako zwykły tekst:

```
java com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o
mqtt-0.trc.txt
```

Śledzenie klienta MQTT dla języka C

Ustaw zmienną środowiskową `MQTT_C_CLIENT_TRACE` , aby śledzić aplikację C klienta produktu MQTT .

Zanim rozpoczniesz

MQTT client for C trace is available for both the pre-built Windows and Linux MQTT client for C libraries, and for the iOS libraries you build yourself.

O tym zadaniu

Ustaw zmienną środowiskową `MQTT_C_CLIENT_TRACE` na ścieżkę do pliku, który ma zawierać dane wyjściowe śledzenia. Dane wyjściowe śledzenia są zapisywane w pliku.

Procedura

Ustaw `MQTT_C_CLIENT_TRACE=mqttcclient.log` przed uruchomieniem aplikacji MQTT klienta C.

a) Na przykład zmodyfikuj przykładowy skrypt w produkcie “Pierwsze kroki z klientem MQTT dla języka C” na stronie 27:

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

b) Uruchom skrypt z katalogu `%sdkroot%/sdk/client/c/samples` .

Wyniki

Pliki wyjściowe śledzenia zaczynają się od następujących wierszy:

```
============================================================
                    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
```

Zadania pokrewne

“Pierwsze kroki z klientem MQTT dla języka C” na stronie 27
Na dowolnej platformie, na której można skompilować źródło C, należy pobrać i uruchomić przykładowy klient MQTT dla języka C. Sprawdź, czy można uruchomić przykładowy klient MQTT dla języka C z produktem IBM MessageSight lub IBM WebSphere MQ jako serwer MQTT.

Śledzenie i debugowanie klienta Java MQTT (Paho)

Domyślny program rejestrujący korzysta ze standardowego mechanizmu rejestrowania Java, który jest znany jako `java.util.logging` (JSR47). Można go skonfigurować przy użyciu pliku konfiguracyjnego lub programowo.

O tym zadaniu

Uwaga: Klient Paho Java ma zastosowanie tylko do wersji produktu IBM WebSphere MQ w wersji 7.5.0.1 i nowszych. Informacje na temat śledzenia klienta Java w produkcie IBM WebSphere MQ w wersji 7.5.0.0znajdują się w sekcji “Śledzenie klienta Java MQTT v3” na stronie 174.

Uwaga: Śledzenie jest funkcją obsługi. Postępuj zgodnie z tymi instrukcjami, jeśli inżynier serwisu IBM poprosi o śledzenie klienta Java MQTT . Dokumentacja produktu nie dokumentuje formatu pliku śledzenia ani sposobu korzystania z niego w celu debugowania klienta. Śledzenie działa tylko w przypadku klienta Java produktu IBM WebSphere MQ Telemetry.

Najprostszą metodą użycia pliku konfiguracyjnego jest określenie jego nazwy we właściwości `java.util.logging.config.file`.

Roboczy plik właściwości `jsr47min.properties` jest udostępniany w pakiecie `org.eclipse.paho.client.mqttv3.logging`

Narzędzie do rejestrowania JSR47 może być używane na wiele sposobów:

- Aby gromadzić komunikaty z wybranego zestawu pakietów
- Aby gromadzić komunikaty z poziomu dziennika i poniżej niego
- Aby wybrać wiele miejsc docelowych dla komunikatów dziennika
- Poprzez udostępnienie wbudowanego programu rejestrującego, który zapisuje do pliku i kontroluje rozmiar i liczbę używanych plików
- Poprzez udostępnienie wbudowanego programu rejestrującego, który zapisuje do pamięci i umożliwia zapisywanie komunikatów w pamięci w oparciu o wyzwalacz.
- Jeśli aplikacja używała biblioteki klienta MQTT również jest instrumentowana przy użyciu komendy JSR47, komunikaty z aplikacji i biblioteki klienta są połączone ze sobą.

Dostępna jest klasa narzędziowa, która ułatwia gromadzenie informacji debugowania. Ta klasa zawiera komunikaty dziennika i śledzenia, które zostały opisane wcześniej, ale mogą gromadzić informacje, takie jak właściwości systemu Java i wartość zmiennych z poziomu klienta Paho.

Narzędzie do debugowania jest udostępniane w klasie publicznej Debug, która jest częścią pakietu `org.eclipse.paho.client.mqttv3.util`. Instancję debugowania można uzyskać za pomocą metody getDebug() zarówno w przypadku asynchronicznych, jak i synchronicznych obiektów klienta MQTT.

Na przykład:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Metoda dumpClientDebuguj () zrzuca maksymalną ilość informacji debugowania. Funkcja dziennika musi być włączona, aby przechwytywać pełne informacje debugowania, które są zapisywane w tym obiekcie. Aby przechwycić pełne informacje debugowania, należy wywołać metodę zrzutu, gdy problem jest znany, na przykład po wystąpieniu określonego wyjątku.

Procedura

1. Utwórz plik konfiguracyjny lub użyj jsr47min.properties , który jest dostarczany.

   Jeśli używany jest udostępniony plik właściwości, należy sprawdzić, czy wyzwalacz jest ustawiony na poprawny poziom błędu. Domyślnie jest to błąd na poziomie Poważny, ale może być konieczne ciągłe zapisywanie danych śledzenia do pliku, a nie przechowywanie go w pamięci, dopóki nie wystąpi błąd. W tym celu należy zmienić:

   ```
   java.util.logging.MemoryHandler.push=SEVERE
   ```

   do

   ```
   java.util.logging.MemoryHandler.push=ALL
   ```

2. Przekaz plik konfiguracyjny śledzenia do maszyny JVM, korzystając z właściwości systemowej.

   Jeśli używany jest plik jsr4min.properties , to jest to:

   ```
   java -Djava.util.logging.config.file=C:\temp\jsr47min.properties
   ```

3. Uruchom klienta.

Wyniki

Jeśli wystąpi wyjątek lub problem, klasa debugowania Paho zapisuje dane śledzenia w pamięci do skonfigurowanego celu pliku.

Śledzenie nie jest automatycznie zapisywane w pliku, ponieważ jest generowane. Jest to wykonywane tylko wtedy, gdy zostanie uruchomiony wyzwalacz lub klasa debugowania powoduje zapisanie danych śledzenia. Te ostatnie mogą wymagać wprowadzenia zmian w kodzie aplikacji.

Każdy wiersz jest zapisywany do procedury obsługi pliku w postaci, w której jest on tworzony. Istnieje możliwość sterowania formatem, w którym komunikaty są zapisywane, konfigurując program FileHandler. Niestandardowa procedura obsługi plików jest udostępniana razem z programem Paho, który zapisuje więcej niż SimpleHandler i mniej niż program XMLHandler udostępniony ze środowiskiem JRE. Rekordy śledzenia, które korzystają z formatera dziennika Paho, mają następującą postać:

```
Level   Data and Time   Class   Method   Thread   clientID   Message
```

Przykład

Udostępniany jest roboczy plik właściwości `jsr47min.properties` . Ten plik zawiera sugerowaną konfigurację gromadzenia danych śledzenia, która ułatwia rozwiązywanie problemów związanych

z klientem programu Paho MQTT. Konfiguruje ona śledzenie, które ma być stale gromadzone w pamięci, przy minimalnym wpływie na wydajność. Po wystąpieniu wyzwalacza lub gdy zostanie wykonane konkretne żądanie, śledzenie w pamięci jest przekazywane do skonfigurowanej procedury obsługi. Domyślnym wyzwalaczem jest komunikat o poważnym poziomie, który jest zerowym połączeniem. Domyślnie śledzenie, które jest gromadzone w pamięci, jest zapisywane w określonym pliku w tym momencie. Domyślnie ten plik jest standardowym `java.util.logging.FileHandler`. Można użyć klasy Debuguj Paho, aby przesunąć śledzenie pamięci do jego celu.

Szczegółowe informacje na temat JSR47 można znaleźć w dokumentacji Javadoc dla pakietu 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
```

Co dalej

W celu programowego gromadzenia danych śledzenia udostępniono klasę narzędziową, która ułatwia gromadzenie informacji debugowania. Ta klasa zawiera komunikaty dziennika i śledzenia, które zostały opisane wcześniej, ale mogą gromadzić informacje, takie jak właściwości systemu Java i wartość zmiennych z poziomu klienta Paho.

Narzędzie do debugowania jest udostępniane w klasie publicznej Debug, która jest częścią pakietu `org.eclipse.paho.client.mqttv3.util`. Instancję debugowania można uzyskać za pomocą metody getDebug() zarówno w przypadku asynchronicznych, jak i synchronicznych obiektów klienta MQTT.

Na przykład:

```
MqttClient cl = new MqttClient();
Debug d = cl.getDebug();
```

Metoda dumpClientDebuguj () zrzuca maksymalną ilość informacji debugowania. Funkcja dziennika musi być włączona, aby przechwytywać pełne informacje debugowania, które są zapisywane w tym obiekcie. Aby przechwycić pełne informacje debugowania, należy wywołać metodę zrzutu, gdy problem jest znany, na przykład po wystąpieniu określonego wyjątku.

Śledzenie klienta MQTT JavaScript

Klienta JavaScript można użyć do gromadzenia danych śledzenia przez zmianę aplikacji WWW klienta w celu wywołania metod na podłączonym obiekcie klienta.

O tym zadaniu

Aby zgromadzić dane śledzenia, można użyć następujących metod:

- Program `client.startTrace()` uruchamia śledzenie dla klienta.
- Program `client.stopTrace()` zatrzymuje śledzenie dla klienta.
- `client.getTraceLog()` zwraca bieżący bufor śledzenia.

Można wysłać bufor śledzenia, aby wysłać go do działu wsparcia IBM dla oprogramowania. Istnieje wiele sposobów, aby to zrobić. W przykładzie pokazano śledzenie, które jest uruchamiane, a następnie dane wyjściowe wysyłane zarówno do konsoli, jak i na podany adres e-mail, a następnie dane śledzenia są zatrzymane.

```
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();
 };
```

Przykładowy wynik:

```
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,
```

V7.5.0.2 Wymagania systemowe dotyczące korzystania z zestawów algorytmów szyfrowania SHA-2 z klientami MQTT

W przypadku środowiska Java 6 z wersji IBM, SR13 , można użyć zestawów algorytmów szyfrowania SHA-2 w celu zabezpieczenia kanałów i aplikacji klienckich produktu MQTT . Jednak zestawy algorytmów szyfrowania SHA-2 nie są domyślnie włączone, dopóki środowisko Java 7 nie będzie od wersji IBM, SR4 , tak więc we wcześniejszych wersjach należy określić wymagany pakiet. Jeśli używany jest klient MQTT z własnym środowiskiem JRE, należy upewnić się, że obsługuje on zestawy algorytmów szyfrowania SHA-2 . Aby aplikacje klienckie korzystały z zestawów algorytmów szyfrowania SHA-2 , klient musi

również ustawić kontekst SSL na wartość obsługując protokół TLS (Transport Layer Security) w wersji 1.2.

W przypadku środowiska Java 7 z wersji IBM, SR4 i SHA-2 zestawy algorytmów szyfrowania SHA-2 są domyślnie włączone. W przypadku środowiska Java 6 z wersji IBM, SR13 i późniejszych wersji usług, jeśli zdefiniowany jest kanał MQTT bez określania zestawu algorytmów szyfrowania, kanał nie będzie akceptować połączeń od klienta przy użyciu zestawu algorytmów szyfrowania SHA-2 . Aby korzystać z zestawów algorytmów szyfrowania SHA-2 , należy określić wymagany pakiet w definicji kanału. Oznacza to, że serwer MQTT włącza ten pakiet przed nawiązaniem połączeń. Oznacza to również, że tylko aplikacje klienckie korzystające z określonego pakietu mogą nawiązać połączenie z tym kanałem.

Istnieje podobne ograniczenie dla produktu Klient MQTT dla produktu Java. Jeśli kod klienta jest uruchomiony na środowisku Java 1.6 JRE z programu IBM, wymagane zestawy algorytmów szyfrowania SHA-2 muszą być jawnie włączone. Aby można było korzystać z tych zestawów, klient musi także ustawiać kontekst SSL na wartość obsługującą protokół TLS (Transport Layer Security) w wersji 1.2. Na przykład:

```
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);
```

W czerwcu 2013 r. przeglądarka Internet Explorer 10 jest jedyną przeglądarką, która współpracuje z serwerem Klient przesyłania komunikatów MQTT dla produktu JavaScript , a także obsługuje protokół TLS 1.2 , więc jest to jedyna przeglądarka, której można użyć, jeśli chcesz utworzyć połączenia SHA-2 z klientem JavaScript .

Listę obsługiwanych obecnie zestawów algorytmów szyfrowania można znaleźć w sekcji odsyłaczy do stron pokrewnych.

Pojęcia pokrewne

“Konfiguracja klienta MQTT dla uwierzytelniania klienta przy użyciu protokołu SSL” na stronie 110
Aby uwierzytelnić klienta MQTT za pomocą protokołu SSL, klient łączy się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port TCP odpowiadający kanałowi telemetrycznego, który jest skonfigurowany do uwierzytelniania klientów SSL.

“Konfiguracja klienta MQTT dla uwierzytelniania kanału przy użyciu protokołu SSL” na stronie 112
Aby uwierzytelnić kanał pomiarowy przy użyciu protokołu SSL, klient musi połączyć się z kanałem pomiarowym przy użyciu protokołu SSL. Musi on określać port odpowiadający kanałowi telemetrycznego, który jest skonfigurowany dla protokołu SSL. Konfiguracja musi zawierać zabezpieczony magazyn kluczy chroniony hasłem, który zawiera certyfikat cyfrowy podpisany przez prywatnie serwer.

V 7.5.0.1 Ograniczenia obsługi przeglądarek WWW dla aplikacji WWW dla urządzeń mobilnych przy użyciu protokołu SSL

Istnieją różnice między różnymi przeglądarkami, na różnych platformach. Zapoznanie się z tymi różnicami ułatwia konfigurowanie aplikacji, ośrodków certyfikacji (CA) i certyfikatów klienta w celu nawiązania połączenia przy użyciu protokołu Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i produktu WebSockets.

Przesyłanie wiadomości mobilnych przy użyciu protokołu JavaScript przez SSL jest całkiem nowe, więc nie jest zaskakujące, że różne kombinacje przeglądarek i platform implementowały możliwość w nieco inny sposób, a także do różnych przydziałów. W poniższej tabeli przedstawiono przegląd aktualnie prac i nie działa on dla każdej kombinacji przeglądarki (Firefox, Chrome, Internet Exploreri Safari) oraz platformy (Windows, Linux, Mac, iOSi Android).

Tabela 6. Obsługa SSL wg platformy i przeglądarki. Dla każdej kombinacji przeglądarki i platformy tabela określa, czy obsługiwane są połączenia anonimowe i nieanonimowe SSL, a także zakres, w jakim przeglądarka współpracuje ze wszystkimi certyfikatami (CA) i certyfikatami klienta.

Przeglądarka	**Obsługa SSL (T/N)**	**SSL działa z dowolnym ośrodkiem CA (T/N)**	**Więcej informacji**
Pulpit programu Firefox .	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Dodaj certyfikat CA i klienta do przeglądarki. Produkt Firefox korzysta ze swojej własnej bazy certyfikatów. Aby zaimportować certyfikat ośrodka CA, należy kliknąć opcję **Narzędzia** > **Opcje** > **Zaawansowane** > **Szyfrowanie** > **Wyświetl certyfikaty** > **Uprawnienia** > **Importuj** . Aby zaimportować certyfikat klienta, należy kliknąć opcję **Narzędzia** > **Opcje** > **Zaawansowane** > **Szyfrowanie** > **Wyświetl certyfikaty** > **Twoje certyfikaty** > **Importuj** . Aby włączyć bezpieczne połączenie, w adresie URL należy podać wartość `https://` . Firefox daje możliwość automatycznego wyboru certyfikatu lub poprzez zadawalanie się o to za każdym razem. Program Firefox udostępnia również opcję korzystania z protokołu SSL 3.0 lub TLS 1.0. Upewnij się, że wybrano obie opcje.

Tabela 6. Obsługa SSL wg platformy i przeglądarki. Dla każdej kombinacji przeglądarki i platformy tabela określa, czy obsługiwane są połączenia anonimowe i nieanonimowe SSL, a także zakres, w jakim przeglądarka współpracuje ze wszystkimi certyfikatami (CA) i certyfikatami klienta. *(kontynuacja)*

Przeglądarka	**Obsługa SSL (T/N)**	**SSL działa z dowolnym ośrodkiem CA (T/N)**	**Więcej informacji**
Pulpit programu Chrome .	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Użyj przeglądarki, aby dodać certyfikat CA i klienta do bazy certyfikatów systemu operacyjnego, która jest współużytkowana z innym oprogramowaniem. Aby zaimportować certyfikat ośrodka CA, kliknij opcję **Ustawienia** > **Pokaż ustawienia zaawansowane** > **Zarządzaj certyfikatami** > **Zaufane główne urzędy certyfikacji** > **Importuj** . Aby zaimportować certyfikat klienta, kliknij opcję **Ustawienia** > **Pokaż ustawienia zaawansowane** > **Zarządzaj certyfikatami** > **Osobiste** > **Importuj** . Aby włączyć bezpieczne połączenie, w adresie URL należy podać wartość `https://` . Chrom monit z kilkoma wyborami. Wybierz poprawną nazwę, w zależności od tego, czy konfigurowana jest połączenie anonimowe czy nieanonimowe.

Tabela 6. Obsługa SSL wg platformy i przeglądarki. Dla każdej kombinacji przeglądarki i platformy tabela określa, czy obsługiwane są połączenia anonimowe i nieanonimowe SSL, a także zakres, w jakim przeglądarka współpracuje ze wszystkimi certyfikatami (CA) i certyfikatami klienta. *(kontynuacja)*

Przeglądarka	**Obsługa SSL (T/N)**	**SSL działa z dowolnym ośrodkiem CA (T/N)**	**Więcej informacji**
Internet Explorer.	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Po nawiązaniu połączenia SSL nieanonimowego użytkownik zostanie poproszony o wybranie poprawnego certyfikatu klienta. Produkt Internet Explorer korzysta ze składnicy certyfikatów Windows , która jest współużytkowana z innym oprogramowaniem. Aby zaimportować certyfikat ośrodka CA, należy kliknąć opcję **Narzędzia** > **Opcje internetowe** > **Treść** > **Certyfikaty** > **Zaufane główne ośrodki certyfikacji** > **Importuj** . Aby zaimportować certyfikat klienta, należy kliknąć opcję **Narzędzia** > **Opcje internetowe** > **Treść** > **Certyfikaty** > **Osobiste** > **Importuj** .
Pulpit programu Safari .	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Użyj przeglądarki, aby dodać certyfikat CA i klienta do bazy certyfikatów systemu operacyjnego, która jest współużytkowana z innym oprogramowaniem.

Tabela 6. Obsługa SSL wg platformy i przeglądarki. Dla każdej kombinacji przeglądarki i platformy tabela określa, czy obsługiwane są połączenia anonimowe i nieanonimowe SSL, a także zakres, w jakim przeglądarka współpracuje ze wszystkimi certyfikatami (CA) i certyfikatami klienta. *(kontynuacja)*

Przeglądarka	**Obsługa SSL (T/N)**	**SSL działa z dowolnym ośrodkiem CA (T/N)**	**Więcej informacji**
Firefox w systemie Android	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Nie	Nieanonimowy: Certyfikaty klienta nie działają, ponieważ nie można spełnić wymagania dodania ośrodka CA do listy w produkcie Firefox. Aby zaimportować certyfikat klienta, należy kliknąć opcję **Ustawienia** > **Zabezpieczenia** > **Pamięć informacji autoryzacyjnych**. Jeśli certyfikat jest podpisany przez zaufany ośrodek CA na liście, można nawiązać bezpieczne połączenie.
Chrome w systemie Android	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Nie	Nieanonimowy: Certyfikaty klienta nie działają, ponieważ nie można spełnić wymagania dodania ośrodka CA do listy w produkcie Chrome. **Uwaga:** Google , aby obsłużyć tę opcję w wersji 27 produktu Chrome. Jest to defekt otwarty od wersji 18. Aby zaimportować certyfikat klienta, należy kliknąć opcję **Ustawienia** > **Zabezpieczenia** > **Pamięć informacji autoryzacyjnych**. Jeśli certyfikat jest podpisany przez zaufany ośrodek CA na liście, można nawiązać bezpieczne połączenie.

Tabela 6. Obsługa SSL wg platformy i przeglądarki. Dla każdej kombinacji przeglądarki i platformy tabela określa, czy obsługiwane są połączenia anonimowe i nieanonimowe SSL, a także zakres, w jakim przeglądarka współpracuje ze wszystkimi certyfikatami (CA) i certyfikatami klienta. *(kontynuacja)*

Przeglądarka	**Obsługa SSL (T/N)**	**SSL działa z dowolnym ośrodkiem CA (T/N)**	**Więcej informacji**
Safari w systemie iOS	Anonimowy SSL-Tak Nieanonimowy SSL-Tak	Anonimowy SSL-Tak Nieanonimowy SSL-Nie	Nieanonimowy: Urządzenie nie ufa certyfikatowi klienta, nawet wtedy, gdy certyfikat CA jest zainstalowany w tym samym czasie. Produkt Safari korzysta ze składnicy certyfikatów urządzenia. Aby zaimportować do tego sklepu, kliknij opcję **Ustawienia** > **Ogólne** > **Profil**, a następnie podj certyfikat CA lub certyfikat klienta na stronie WWW lub wyślij go do siebie.
Chrome w systemie iOS	Anonimowy SSL-Tak Nieanonimowy SSL-Nie	Anonimowy SSL-Nie Nieanonimowy SSL-Nie	Anonimowy: Tylko aplikacje Apple mogą uzyskiwać dostęp do głównego sklepu systemu iOS . Dlatego produkt Chrome musi korzystać z własnej listy ośrodków CA, do której nie można dodać. Nieanonimowy: Certyfikaty klienta nie działają, ponieważ nie można spełnić wymagania dodania ośrodka CA do listy.

Zadania pokrewne

“Łączenie partycji Klient przesyłania komunikatów MQTT dla produktu JavaScript przez SSL i WebSockets” na stronie 81

Połączenie aplikacji WWW należy bezpiecznie połączyć z produktem IBM WebSphere MQ , korzystając z przykładowych stron HTML produktu Klient przesyłania komunikatów MQTT dla produktu JavaScript z protokołem SSL i WebSocket protocol.

Informacje pokrewne

Mozilla: (SSL) Czy Firefox korzysta z pamięci masowej w systemie Android CA lub jej własnej?

Chrom: Wydanie 134418-Wdrożenie obsługi certyfikatu klienta

Nie można otworzyć serwisu https z niezaufanym certyfikatem na serwerze ie10

Rozwiązywanie problemu: klient MQTT nie łączy się z serwerem

Rozwiąż problem z programem klienckim MQTT, który nie może nawiązać połączenia z usługą telemetryczną (MQXR).

Zanim rozpoczniesz

Czy problem jest dostępny na serwerze, na kliencie, czy w połączeniu? Czy został zapisany własny klient obsługi protokołu MQTT v3 lub aplikacja kliencka MQTT przy użyciu klientów MQTT w języku C lub Java WebSphere ?

Uruchom aplikację weryfikacyjną dostarczonej z produktem WebSphere MQ Telemetry na serwerze, a następnie sprawdź, czy usługa telemetryczna i usługi telemetryczne (MQXR) działają poprawnie. Następnie należy przesłać aplikację weryfikacyjną do klienta, a następnie uruchomić tam aplikację weryfikacyjną.

O tym zadaniu

Istnieje wiele powodów, dla których klient MQTT może nie nawiązać połączenia, lub może to zakończyć nie nawiązywający połączenia z serwerem telemetryczny.

Procedura

1. Należy wziąć pod uwagę, jakie wnioski można wyciągnąć z kodu przyczyny, który usługa telemetryczna (MQXR) zwróciła do produktu `MqttClient.Connect`. Jaki jest typ błędu połączenia?

Opcja	Opis
REASON_CODE_INVALID_PROTOCOL_VERSION	Upewnij się, że adres gniazda odpowiada kanałowi telemetrycznego, a nie użyto tego samego adresu gniazda dla innego brokera.
REASON_CODE_INVALID_CLIENT_ID	Sprawdź, czy identyfikator klienta nie jest dłuższy niż 23 bajty i czy zawiera tylko znaki z zakresu: `A-Z, a-z, 0-9, './_%`
REASON_CODE_INVALID_DESTINATION	Sprawdź, czy identyfikator klienta nie jest taki sam, jak nazwa menedżera kolejek.
REASON_CODE_SERVER_CONNECT_ERROR	Sprawdź, czy usługa telemetryczna (MQXR) i menedżer kolejek działają normalnie. Użyj komendy **netstat** , aby sprawdzić, czy adres gniazda nie jest przydzielony do innej aplikacji.

   Jeśli w bibliotece klienta MQTT została napisana biblioteka klienta MQTT, a nie jest używana jedna z bibliotek udostępnionych w produkcie IBM WebSphere MQ Telemetry, należy sprawdzić kod powrotu `CONNACK` .

   Z tych trzech błędów można wnioskować, że klient połączył się z usługą telemetryczną (MQXR), ale usługa napotkała błąd.

2. Zastanów się, jakie wnioski mogą być wyciągane z kodów przyczyny, które klient generuje, gdy usługa telemetryczna (MQXR) nie odpowiada:

Opcja	Opis
REASON_CODE_CLIENT_EXCEPTION **REASON_CODE_CLIENT_TIMEOUT**	Poszukaj pliku FDC na serwerze. Patrz sekcja “Dzienniki po stronie serwera” na stronie 170. Gdy usługa telemetryczna (MQXR) wykryje, że klient przekroczył limit czasu, zapisuje plik FDC (first-failure data capture). Zapisuje plik FDC

Opcja	Opis
	za każdym razem, gdy połączenie zostanie nieoczekiwanie przerwane.

Usługa telemetryczna (MQXR) mogła nie odpowiedzieć na klienta, a limit czasu na kliencie utraci ważność. Klient produktu WebSphere MQ Telemetry Java zawiesza się tylko wtedy, gdy aplikacja ustała nieokreślony limit czasu. Klient zgłasza jeden z tych wyjątków po przekroczeniu limitu czasu ustawionego dla `MqttClient.Connect` , co oznacza, że problem nie został zdiagnozowany.

Jeśli nie znajdziesz pliku FDC, który koreluje z awarią połączenia, nie możesz wnioskować, że klient próbował połączyć się z serwerem:

a) Upewnij się, że klient wysłał żądanie połączenia.

Sprawdź żądanie TCPIP przy użyciu narzędzia, takiego jak **tcpmon**, dostępnego pod adresem https://java.net/projects/tcpmon .

b) Czy adres zdalnego gniazda używany przez klienta jest zgodny z adresem gniazda zdefiniowanym dla kanału telemetrycznego?

Domyślna klasa trwałości plików w kliencie Java SE MQTT dostarczonym razem z produktem IBM WebSphere MQ Telemetry tworzy folder o nazwie: *clientIdentifier*`-tcp`*hostNamePort* lub *clientIdentifier*`-ssl`*hostNamePort* w katalogu roboczym klienta. Nazwa folderu informuje o wartościach (`nazwaHosta` i `port`) użytych podczas próby nawiązania połączenia.; patrz “Pliki dziennika po stronie klienta” na stronie 171.

c) Czy można wysłać komendę ping do adresu serwera zdalnego?

d) Czy **netstat** na serwerze pokazuje, że kanał telemetryczny działa na porcie, z którym klient się łączy?

3. Sprawdź, czy usługa telemetryczna (MQXR) znalazła problem w żądaniu klienta.

Usługa telemetryczna (MQXR) zapisuje błędy wykryte w programie `mqxr.log`, a menedżer kolejek zapisuje błędy w programie `AMQERR01.LOG`; patrz

4. Spróbuj zlokalizować problem, uruchamiając innego klienta.

- Uruchom przykładową aplikację MQTT przy użyciu tego samego kanału telemetrycznego.
- Uruchom klienta interfejsu GUI produktu **wmqttSample** , aby sprawdzić połączenie. Pobierz produkt **wmqttSample** , pobierając pakiet SupportPac IA92.

Uwaga: Starsze wersje IA92 nie zawierają biblioteki klienta Java MQTT v3 .

Uruchom przykładowe programy na platformie serwera, aby wyeliminować wątpliwości dotyczące połączenia sieciowego, a następnie uruchomić przykłady na platformie klienckiej.

5. Inne rzeczy do sprawdzenia:

a) Czy dziesiątki tysięcy klientów MQTT próbuje połączyć się jednocześnie?

Kanały telemetryczne mają kolejkę, która buforuje zaleganie połączeń przychodzących. Połączenia są przetwarzane powyżej 10 000 sekund. Wielkość buforu dziennika backlog można skonfigurować przy użyciu kreatora kanału pomiarowego w programie IBM WebSphere MQ Explorer. Jego domyślna wielkość to 4096. Sprawdź, czy dziennik zaległy nie został skonfigurowany do niskiej wartości.

b) Czy usługa telemetryczna (MQXR) i menedżer kolejek są nadal uruchomione?

c) Czy klient jest połączony z menedżerem kolejek o wysokiej dostępności, który zmienił swój adres TCPIP?

d) Czy firewall selektywnie filtruje pakiety danych wychodzących lub zwracających dane?

Rozwiązywanie problemu: połączenie klienta MQTT zostało usunięte

Dowiedz się, co powoduje, że klient zgłasza nieoczekiwane wyjątki `ConnectionLost` po pomyślnym nawiązaniu połączenia i uruchomieniu go przez krótki lub długi czas.

Zanim rozpoczniesz

Klient MQTT pomyślnie nawiąże połączenie. Klient może być przez długi czas. Jeśli klienty zaczynają się od krótkiego odstępu czasu między nimi, czas między połączeniem i odrzucanym połączeniem może być krótki.

Odróżnienie usuniętego połączenia od połączenia, które zostało pomyślnie wykonane, a następnie porzucone, nie jest trudne. Usunięte połączenie jest definiowane przez klienta MQTT wywołując metodę `MqttCallback.ConnectionLost` . Metoda jest wywoływana tylko po pomyślnym nawiązaniu połączenia. Objaw różni się od `MqttClient.Connect` zgłaszającego wyjątek po odebraniu ujemnego potwierdzenia lub przekroczenia limitu czasu.

Jeśli aplikacja kliencka MQTT nie korzysta z bibliotek klienta MQTT dostarczanych przez produkt IBM WebSphere MQ, objaw jest zależny od klienta. W protokole MQTT v3 objawem jest brak terminowej odpowiedzi na żądanie do serwera lub niepowodzenie połączenia TCP/IP.

O tym zadaniu

Klient MQTT wywołuje produkt `MqttCallback.ConnectionLost` z wyjątek throwable w odpowiedzi na wszystkie problemy z serwerem, które wystąpiły po odebraniu potwierdzenia połączenia. Gdy klient MQTT zwróci dane z `MqttTopic.publish` i `MqttClient.subscribe` , żądanie jest przesyłane do wątku klienta MQTT, który jest odpowiedzialny za wysyłanie i odbieranie komunikatów. Błędy po stronie serwera są raportowane asynchronicznie, przekazując wyjątek throwable do metody wywołania zwrotnego `ConnectionLost` .

Usługa telemetryczna (MQXR) zawsze zapisuje plik przechwytywania danych pierwszego niepowodzenia, jeśli spowoduje to usunięcie połączenia.

Procedura

1. Czy uruchomiono inny klient, który użył tego samego `ClientIdentifier`?

   Jeśli drugi klient zostanie uruchomiony lub ten sam klient zostanie zrestartowany, za pomocą tego samego `ClientIdentifier`, pierwsze połączenie z pierwszym klientem zostanie usunięte.
2. Czy klient uzyskał dostęp do tematu, do którego nie ma uprawnień do publikowania lub subskrybowania?

   Wszelkie działania wykonywane przez usługę telemetryczną w imieniu klienta, który zwraca wynik `MQCC_FAIL` , powodują, że usługa upuszcza połączenie klienta.

   Kod przyczyny nie jest zwracany do klienta.

   - Poszukaj komunikatów dziennika w plikach `mqxr.log` i `AMQERR01.LOG` dla menedżera kolejek, z którym klient jest połączony. Patrz sekcja “Dzienniki po stronie serwera” na stronie 170.
3. Czy połączenie TCP/IP zostało zerowane?

   Zapora firewall może mieć niski limit czasu na potrzeby oznaczania połączenia TCPIP jako nieaktywnego, a następnie upuściło połączenie.

   - Skróć nieaktywny czas połączenia TCPIP za pomocą komendy `MqttConnectOptions.setKeepAliveInterval`.

Rozwiązywanie problemów: utrata komunikatów w aplikacji MQTT

Rozwiąż problem utraty komunikatu. Czy wiadomość jest nietrwała, wysłana do niewłaściwego miejsca, czy nigdy nie została wysłana? Błędnie zakodowany program kliencki może utracić komunikaty.

Zanim rozpoczniesz

Jak pewne jest to, że wiadomość, którą wysłałeś, została utracona? Czy można wnioskować, że komunikat został utracony, ponieważ komunikat nie został odebrany? Jeśli komunikat jest publikacją, która komunikat jest utracony: komunikat wysłany przez publikatora lub komunikat wysłany do subskrybenta? Czy subskrypcja została utracona, a broker nie wysyła publikacji dla tej subskrypcji do subskrybenta?

Jeśli rozwiązanie obejmuje rozproszoną publikowanie/subskrybowanie, korzystanie z klastrów lub hierarchie publikowania/subskrypcji, istnieje wiele problemów z konfiguracją, które mogą spowodować pojawienie się utraconego komunikatu.

Jeśli wysłałeś wiadomość z "Co najmniej raz" lub "Co najwyżej raz" jakość usługi, to jest prawdopodobne, że wiadomość, którą uważasz, że jest zagubiona nie została dostarczona w sposób, w jaki się spodziewasz. Jest mało prawdopodobne, aby komunikat został błędnie usunięty z systemu. Nie powiodło się utworzenie publikacji lub subskrypcji, której oczekiwał użytkownik.

Najważniejszy krok, jaki należy wykonać w celu określenia problemu utraty wiadomości, to potwierdzenie utraty komunikatu. Utwórz ponownie scenariusz i utracisz więcej komunikatów. Aby wyeliminować wszystkie przypadki systemu odrzucających komunikaty, należy użyć jakości usługi "Co najmniej raz" lub "Co najwyżej raz".

O tym zadaniu

Są cztery nogi do diagnozowania zaginionego komunikatu.

1. Komunikaty "Fire and forget" pracujące jako-zaprojektowane. Komunikaty "Fire and forget" są czasami usuwane przez system.
2. Konfiguracja: konfigurowanie publikowania/subskrybowania z poprawnymi uprawnieniami w środowisku rozproszonym nie jest proste.
3. Błędy programowania klienta: odpowiedzialność za dostarczenie wiadomości nie jest wyłącznie obowiązkiem kodu napisanego przez IBM.
4. Jeśli wyczerpałeś wszystkie te możliwości, możesz podjąć decyzję o zaangażowaniu serwisu IBM .

Procedura

1. Jeśli zgubiony komunikat miał "Ogień i zapomnieć" jakość usługi, ustawić "Co najmniej raz" lub "Co najwyżej raz" jakość usługi. Spróbuj ponownie utracić wiadomość.
   - Komunikaty wysyłane z jakością usługi "Fire and forget" są odrzucane przez produkt IBM WebSphere MQ w wielu okolicznościach:
     - Zatrzymano utratę komunikacji i kanał.
     - Menedżer kolejek został zamknięty.
     - Nadmierna liczba komunikatów.
   - Dostarczenie wiadomości typu "Fire and forget" zależy od niezawodności TCP/IP. Protokół TCP/IP nadal wysyła pakiety danych do momentu potwierdzenia ich dostarczenia. Jeśli sesja TCP/IP zostanie zerwana, komunikaty z jakością usługi "Ogień i zapomnij" zostaną utracone. Sesja może zostać zerwana przez zamknięcie klienta lub serwera, problem z komunikacją lub zaporę firewall rozłączenia sesji.
2. Sprawdź, czy klient restartuje poprzednią sesję, aby ponownie wysłać niedostarczone wiadomości z "Przynajmniej raz" lub "Co najwyżej raz" jakości usługi.
   a) Jeśli aplikacja kliencka korzysta z klienta Java SE MQTT, sprawdź, czy jest ona ustawiana `MqttClient.CleanSession` na `false` .
   b) Jeśli używane są różne biblioteki klienta, sprawdź, czy sesja jest poprawnie restartowana.
3. Sprawdź, czy aplikacja kliencka restartuje tę samą sesję i przez pomyłkę nie uruchamiała innej sesji.

Aby ponownie uruchomić tę samą sesję, serwer `cleanSession = false`i `Mqttclient.clientIdentifier` i `MqttClient.serverURI` muszą być takie same, jak w poprzedniej sesji.

4. Jeśli sesja zostanie zamknięta przedwcześnie, sprawdź, czy komunikat jest dostępny w składnicy trwałości w kliencie, aby wysłać je ponownie.
   a) Jeśli aplikacja kliencka korzysta z klienta MQTT Java SE, należy sprawdzić, czy komunikat jest zapisywany w folderze trwałości. Patrz sekcja “Pliki dziennika po stronie klienta” na stronie 171 .
   b) Jeśli używane są różne biblioteki klienta lub zaimplementowano własny mechanizm utrwalania, należy sprawdzić, czy działa on poprawnie.
5. Sprawdź, czy nikt nie usunował komunikatu przed jego dostarczyniem.

   Niedostarczone komunikaty oczekujące na dostarczenie do klientów MQTT są przechowywane w produkcie `SYSTEM.MQTT.TRANSMIT.QUEUE`. Komunikaty oczekujące na dostarczenie do serwera telemetrycznego są zapisywane przez mechanizm trwałości klienta. Patrz sekcja Trwałość komunikatów w klientach MQTT.
6. Sprawdź, czy klient ma subskrypcję dla tej publikacji, która oczekuje na odebranie.

   Lista subskrypcji za pomocą programu WebSphere MQ Explorer lub za pomocą komend **runmqsc** lub PCF. Zostaną nazwane wszystkie subskrypcje klienta MQTT. Nadano im nazwę w postaci: *`ClientIdentifier`*:*Topic name*
7. Sprawdź, czy publikator ma uprawnienia do publikowania oraz subskrybent, który ma zostać subskrybowany w temacie publikowania.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   W klastrowym systemie publikowania/subskrypcji subskrybent musi być autoryzowany do tematu w menedżerze kolejek, z którym powiązany jest subskrybent. Subskrybent nie musi być autoryzowany do subskrybowania tematu w menedżerze kolejek, w którym publikowana jest publikacja. Kanały między menedżerami kolejek muszą być poprawnie autoryzowane do przekazywania subskrypcji proxy i przekazywania publikacji.

   Utwórz tę samą subskrypcję i opublikuj ją w programie IBM WebSphere MQ Explorer. Symulowanie publikowania i subskrybowania klienta aplikacji przy użyciu programu narzędziowego klienta. Uruchom program narzędziowy z programu IBM WebSphere MQ Explorer, a następnie zmień jego identyfikator, aby był zgodny z adoptą przez aplikację kliencką.
8. Sprawdź, czy subskrybent ma uprawnienia do umieszczania publikacji na serwerze `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```
9. Sprawdź, czy aplikacja IBM WebSphere MQ punkt z punktem ma uprawnienia do umieszczania swojego komunikatu na serwerze `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

   Więcej informacji na ten temat zawiera sekcja "Wysyłanie komunikatu do klienta bezpośrednio" w sekcji Konfigurowanie rozproszonego kolejkowania w celu wysyłania komunikatów do klientów MQTT.

Rozwiązywanie problemu: usługa telemetryczna (MQXR) nie jest uruchomiona

Rozwiąż problem z usługą telemetryczną (MQXR), której uruchomienie nie powiodło się. Sprawdź, czy w instalacji produktu WebSphere MQ Telemetry nie brakuje plików, przeniesiono je lub czy mają one niepoprawne uprawnienia. Sprawdź ścieżki używane przez usługę telemetryczną (MQXR), która znajduje się w programach usługowych telemetrycznych (MQXR).

Zanim rozpoczniesz

Składnik WebSphere MQ Telemetry jest zainstalowany. Eksplorator produktu IBM WebSphere MQ ma folder Telemetry w programie **IBM WebSphere MQ > Menedżery kolejek > *qMgrNazwa* > Telemetry**. Jeśli folder nie istnieje, instalacja nie powiodła się.

Usługa telemetryczna (MQXR) musi zostać utworzona, aby można ją było uruchomić. Jeśli usługa telemetryczna (MQXR) nie została utworzona, należy uruchomić opcję **Definiowanie przykładowej konfiguracji ...** w folderze `Telemetry` .

Jeśli usługa telemetryczna (MQXR) została uruchomiona przed, wówczas dodatkowe foldery **Kanały** i **Status kanału** są tworzone w folderze `Telemetry` . Usługa telemetryczna (`SYSTEM.MQXR.SERVICE`) znajduje się w folderze **Usługi** . Jest on widoczny po kliknięciu przełącznika eksploratora w celu wyświetlenia obiektów systemowych.

Kliknij prawym przyciskiem myszy opcję `SYSTEM.MQXR.SERVICE` , aby uruchomić i zatrzymać usługę, wyświetlić jej status i wyświetlić, czy identyfikator użytkownika ma uprawnienia do uruchamiania usługi.

O tym zadaniu

Uruchomienie usługi telemetrycznej `SYSTEM.MQXR.SERVICE` (MQXR) nie powiodło się. Nierozpoczęcie się objawia się na dwa różne sposoby:

1. Komenda start nie powiodła się natychmiast.
2. Komenda start zakończy się powodzeniem i natychmiast po niej następuje zatrzymanie usługi.

Procedura

1. Uruchom usługę

   Wynik

   Usługa zostanie natychmiast zatrzymana. W oknie zostanie wyświetlony komunikat o błędzie, na przykład:

   ```
   WebSphere MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Przyczyna

   Pliki nie są dostępne w instalacji lub uprawnienia do zainstalowanych plików są ustawione nieprawidłowo.

   Składnik IBM WebSphere MQ Telemetry jest instalowany tylko na jednej z pary wysoce dostępnych menedżerów kolejek. Jeśli instancja menedżera kolejek przełącza się do trybu gotowości, podejmuje próbę uruchomienia produktu `SYSTEM.MQXR.SERVICE`. Wykonanie komendy uruchamianej w celu uruchomienia usługi nie powiedzie się, ponieważ usługa telemetryczna (MQXR) nie jest zainstalowana w stanie gotowości.

   Dochodzenie

   Zapoznaj się z dziennikami błędów; patrz sekcja “Dzienniki po stronie serwera” na stronie 170.

   Działania

   Zainstaluj lub zdeinstaluj i ponownie zainstaluj składnik WebSphere MQ Telemetry.

2. Uruchom usługę, poczekaj 30 sekund, odśwież eksplorator i sprawdź status usługi.

   Wynik

   Usługa zostanie uruchomiona, a następnie zostanie zatrzymana.

   Przyczyna

   Program `SYSTEM.MQXR.SERVICE` uruchomił komendę **`runMQXRService`** , ale wykonanie komendy nie powiodło się.

   Dochodzenie

   Zapoznaj się z dziennikami błędów; patrz sekcja “Dzienniki po stronie serwera” na stronie 170.

Sprawdź, czy problem występuje tylko z określonym kanałem przykładowym. Utwórz kopię zapasową i usuń zawartość katalogu *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ . Uruchom przykładowy kreator konfiguracji i spróbuj uruchomić usługę.

Działania

Poszukaj problemów z uprawnieniami i ścieżkami.

Rozwiązywanie problemu: moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną

Sprawdź, czy moduł logowania JAAS nie jest wywoływany przez usługę telemetryczną (MQXR), a następnie skonfiguruj usługę JAAS w celu usunięcia problemu.

Zanim rozpoczniesz

Zmodyfikowano produkt *WMQ installation directory*\mqxr\samples\LoginModule.java w celu utworzenia własnej klasy uwierzytelniania *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. Alternatywnie, użytkownik zapisał własne klasy uwierzytelniania JAAS i umieł je w wybranym katalogu. Po przeprowadzeniu pewnych początkowych testów z usługą telemetryczną (MQXR) istnieje podejrzenie, że klasa uwierzytelniania nie jest wywoływana przez usługę telemetryczną (MQXR).

Uwaga: Ochrona przed możliwością nadpisania klas uwierzytelniania przez konserwację używaną w produkcie WebSphere MQ. Użyj własnej ścieżki do klas uwierzytelniania, a nie ścieżki w drzewie katalogów produktu WebSphere MQ .

O tym zadaniu

W ramach zadania przedstawiony jest scenariusz ilustrujący sposób rozwiązania problemu. W scenariuszu pakiet o nazwie security.jaas zawiera klasę uwierzytelniania JAAS o nazwie JAASLogin.class. Jest on przechowywany w ścieżce C:\WMQTelemetryApps\security\jaas. Informacje na temat konfigurowania produktu JAAS for IBM WebSphere MQ Telemetry można znaleźć w sekcji Konfiguracja kanału telemetrycznego JAAS . Przykład: “Przykładowa konfiguracja JAAS” na stronie 194 jest przykładową konfiguracją.

Procedura

1. Informacje o wyjątku zgłoszonym przez produkt javax.security.auth.login.LoginExceptionmożna znaleźć w sekcji mqxr.log .

   Zapoznaj się z “Dzienniki po stronie serwera” na stronie 170 , aby uzyskać informacje na temat ścieżki do mqxr.log, a Rysunek 54 na stronie 195 -na przykład wyjątku wymienionego w dzienniku.
2. Popraw konfigurację JAAS , porównując ją z obrobionym przykładem w produkcie “Przykładowa konfiguracja JAAS” na stronie 194.
3. Zastąp klasę logowania przykładową JAASLoginModule, po refaktoryzacji jej w pakiecie uwierzytelniania i wdróż ją przy użyciu tej samej ścieżki. Zmień wartość loggedIn między true i false.

   Jeśli problem zostanie rozwiązany, gdy loggedIn to true, i pojawi się taka sama, gdy loggedIn to false, problem leży w klasie logowania.
4. Sprawdź, czy problem dotyczy autoryzacji, a nie uwierzytelniania.
   a) Zmień definicję kanału telemetrycznego, aby wykonać sprawdzanie autoryzacji przy użyciu stałego identyfikatora użytkownika. Wybierz identyfikator użytkownika, który należy do grupy mqm .
   b) Ponownie uruchom aplikację kliencką.

      Jeśli problem zniknie, rozwiązanie polega na tym, że ID użytkownika jest przekazywany do autoryzacji. Jaka jest przekazywana nazwa użytkownika? Wydrukuj go w pliku z modułu logowania. Sprawdź uprawnienia dostępu za pomocą programu IBM WebSphere MQ Explorer lub **dspmqauth**.

Przykładowa konfiguracja JAAS

Aby skonfigurować kanał pomiarowy, należy użyć kreatora **Nowy kanał pomiarowy** w programie WebSphere MQ Explorer. Klient łączy się z portem 1884 i łączy się z kanałem telemetryczny `JAASMCAUser` . Rysunek 48 na stronie 194 przedstawia przykład pliku właściwości telemetrycznych utworzonego przez kreator telemetryczny. Nie należy edytować tego pliku bezpośrednio. Kanał uwierzytelnia za pomocą usługi JAASprzy użyciu konfiguracji o nazwie `JAASConfig`. Po uwierzytelnieniu klienta korzysta on z identyfikatora użytkownika `Admin` w celu autoryzowania dostępu do obiektów produktu 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
```

Rysunek 48. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties

Plik konfiguracyjny JAAS zawiera sekcję o nazwie `JAASConfig` , która określa nazwę klasy Java `security.jaas.JAASLogin`, która jest używana przez produkt JAAS do uwierzytelniania klientów.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Rysunek 49. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

Po uruchomieniu programu `SYSTEM.MQTT.SERVICE` do ścieżki klas zostanie dodana ścieżka w programie Rysunek 50 na stronie 194 .

```
CLASSPATH=C:\WMQTelemtryApps;
```

Rysunek 50. WMQ Installation directory\data\qmgrs\qMgrName\service.env

Rysunek 51 na stronie 194 przedstawia dodatkową ścieżkę w produkcie Rysunek 50 na stronie 194 dodaną do ścieżki klasy, która jest ustawiona dla usługi telemetrycznej (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;
```

Rysunek 51. Dane wyjściowe ścieżki klasy z pliku runMQXRService.bat

Dane wyjściowe w programie Rysunek 52 na stronie 195 pokazują, że usługa telemetryczna (MQXR) została uruchomiona z definicją kanału wyświetlenią w produkcie Rysunek 48 na stronie 194.

```
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
```

Rysunek 52. WMQ Installation directory\data\qmgrs\qMgrName\errors\mqxr.log

Gdy aplikacja kliencka łączy się z kanałem JAAS , jeśli produkt `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` nie jest zgodny z nazwą sekcji JAAS w pliku `jaas.config` , nawiązanie połączenia nie powiedzie się, a klient zgłosi wyjątek z kodem powrotu `0`; patrz Rysunek 53 na stronie 195. Drugi wyjątek, `Client is not connected (32104)`, został zgłoszony, ponieważ klient próbował rozłączyć się, gdy nie był połączony.

```
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)
```

Rysunek 53. Zgłoszono wyjątek podczas nawiązywania połączenia com.ibm.mq.id.PubAsyncRestartable .

`mqxr.log` zawiera dodatkowe dane wyjściowe przedstawione w sekcji Rysunek 53 na stronie 195.

Błąd jest wykrywany przez usługę JAAS , która zgłasza produkt `javax.security.auth.login.LoginException` z przyczyną `No LoginModules configured for JAAS`. Może to być spowodowane, podobnie jak w produkcie Rysunek 54 na stronie 195, błędną nazwą konfiguracji. Może to być również wynikiem innych problemów JAAS , które napotkały ładowanie konfiguracji JAAS .

Jeśli usługa JAASnie zgłosiła wyjątku, usługa JAAS pomyślnie załadowała klasę `security.jaas.JAASLogin` o nazwie podanej w sekcji `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
```

Rysunek 54. mqxr.log -błąd podczas ładowania konfiguracji JAAS

Rozwiązywanie problemu: uruchamianie lub uruchamianie demona

Zapoznaj się z dziennikiem konsoli produktu IBM WebSphere MQ Telemetry dla urządzeń, włącz śledzenie lub skorzystaj z tabeli objawów w tym temacie, aby rozwiązać problemy z demonem.

Procedura

1. Sprawdź dziennik konsoli.

   Jeśli demon jest uruchomiony na pierwszym planie, komunikaty konsoli są zapisywane w oknie terminalu. Jeśli demon został uruchomiony w tle, konsola jest w miejscu, w którym przekierowano `stdout` do.
2. Zrestartuj demona.

   Zmiany w pliku konfiguracyjnym nie są aktywowane, dopóki demon nie zostanie zrestartowany.
3. Zapoznaj się z Tabela 7 na stronie 196:

Tabela 7. Tabela objawów

Problem	**Sugerowane rozwiązanie**

Po uruchomieniu demona w systemie Windowszostanie wyświetlony następujący komunikat:
`System nie może wykonać określonego programu`
lub wersji
`Nie powiodło się uruchomienie aplikacji`
`ponieważ jego konfiguracja z boku jest niepoprawna.`
	Zainstaluj pakiet Microsoft Visual C++ 2008 Redistributable Package.

Dwa lub więcej demonów lub serwery z obsługą MQTT są połączone z mostem przez most lub mosty, a procesor wykazuje nadmierne obciążenie.	Prawdopodobnie istnieje pętla komunikatów z jednym lub większą liczbę komunikatów przekazywanych wielokrotnie z jednego serwera do drugiego. Sprawdź parametry tematu w plikach konfiguracyjnych. W miarę możliwości należy używać bardziej szczegółowych tematów. Najczęstszym powodem pętli połączeń są szerokie znaki wieloznaczne w obu kierunkach.
Most nie może połączyć się ze zdalnym serwerem obsługujący MQTT, z którym mogą się łączyć inne klienty MQTT.	Serwer zdalny może być niekompatybilny z próbami określenia, czy zdalny serwer jest również WebSphere MQ Telemetry dla urządzeń. Spróbuj ustawić **`try_private`** na `wyłączone` , aby wyłączyć przetwarzanie specjalne w celu wyeliminowania pętli komunikatów.
Ten komunikat jest wyświetlany, gdy skonfigurowano most: `Ostrzeżenie: program Connect nie był pierwszym pakietem w gnieździe 1888, został przechwyty CONNACK.`	Prawdopodobnie został skonfigurowany most do powrotu do lokalnego demona. Pętla zwrotna nie jest obsługiwana.

Rozwiązywanie problemu: klienty MQTT nie nawiązują połączenia z demonem

Klienty nie łączą się z demonem, a demon nie łączy się z innymi demonami ani z kanałem telemetryczny WebSphere MQ .

O tym zadaniu

Śledzenie każdego pakietu MQTT wysłanego i odebranego przez demona.

Procedura

Ustaw parametr **trace_output** na wartość `protocol` w pliku konfiguracyjnym demona lub wyślij komendę do demona przy użyciu pliku `amqtdd.upd` .

Przykład korzystania z pliku `amqtdd.upd` zawiera sekcja Przesyłanie komunikatów między demonem produktu IBM WebSphere MQ Telemetry dla urządzeń i IBM WebSphere MQ .

Korzystając z ustawienia protokołu, demon drukuje komunikat do konsoli opisujący każdy pakiet MQTT, który wysyła i odbiera.

Uwagi

Niniejsza publikacja została opracowana z myślą o produktach i usługach oferowanych w Stanach Zjednoczonych.

IBM może nie oferować w innych krajach produktów, usług lub opcji omawianych w tej publikacji. Informacje o produktach i usługach dostępnych w danym kraju można uzyskać od lokalnego przedstawiciela IBM. Odwołanie do produktu, programu lub usługi IBM nie oznacza, że można użyć wyłącznie tego produktu, programu lub usługi IBM. Zamiast nich można zastosować ich odpowiednik funkcjonalny pod warunkiem, że nie narusza to praw własności intelektualnej firmy IBM. Jednakże cała odpowiedzialność za ocenę przydatności i sprawdzenie działania produktu, programu lub usługi pochodzących od producenta innego niż IBM spoczywa na użytkowniku.

IBM może posiadać patenty lub złożone wnioski patentowe na towary i usługi, o których mowa w niniejszej publikacji. Używanie tego dokumentu nie daje żadnych praw do tych patentów. Pisemne zapytania w sprawie licencji można przesyłać na adres:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Zapytania w sprawie licencji dotyczących informacji kodowanych przy użyciu dwubajtowych zestawów znaków (DBCS) należy kierować do lokalnych działów IBM Intellectual Property Department lub zgłaszać na piśmie pod adresem:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Poniższy akapit nie obowiązuje w Wielkiej Brytanii, a także w innych krajach, w których jego treść pozostaje w sprzeczności z przepisami prawa miejscowego: INTERNATIONAL BUSINESS MACHINES CORPORATION DOSTARCZA TĘ PUBLIKACJĘ W STANIE, W JAKIM SIĘ ZNAJDUJE ("AS IS"), BEZ JAKICHKOLWIEK GWARANCJI (RĘKOJMIĘ RÓWNIEŻ WYŁĄCZA SIĘ), WYRAŹNYCH LUB DOMNIEMANYCH, A W SZCZEGÓLNOŚCI DOMNIEMANYCH GWARANCJI PRZYDATNOŚCI HANDLOWEJ, PRZYDATNOŚCI DO OKREŚLONEGO CELU ORAZ GWARANCJI, ŻE PUBLIKACJA TA NIE NARUSZA PRAW OSÓB TRZECICH. Ustawodawstwa niektórych krajów nie dopuszczają zastrzeżeń dotyczących gwarancji wyraźnych lub domniemanych w odniesieniu do pewnych transakcji; w takiej sytuacji powyższe zdanie nie ma zastosowania.

Informacje zawarte w niniejszej publikacji mogą zawierać nieścisłości techniczne lub błędy typograficzne. Informacje te są okresowo aktualizowane, a zmiany te zostaną uwzględnione w kolejnych wydaniach tej publikacji. IBM zastrzega sobie prawo do wprowadzania ulepszeń i/lub zmian w produktach i/lub programach opisanych w tej publikacji w dowolnym czasie, bez wcześniejszego powiadomienia.

Wszelkie wzmianki w tej publikacji na temat stron internetowych innych podmiotów zostały wprowadzone wyłącznie dla wygody użytkowników i w żadnym wypadku nie stanowią zachęty do ich odwiedzania. Materiały dostępne na tych stronach nie są częścią materiałów opracowanych dla tego produktu IBM, a użytkownik korzysta z nich na własną odpowiedzialność.

IBM ma prawo do używania i rozpowszechniania informacji przysłanych przez użytkownika w dowolny sposób, jaki uzna za właściwy, bez żadnych zobowiązań wobec ich autora.

Licencjobiorcy tego programu, którzy chcieliby uzyskać informacje na temat programu w celu: (i) wdrożenia wymiany informacji między niezależnie utworzonymi programami i innymi programami (łącznie

© Copyright IBM Corp. 2007, 2024

z tym opisywanym) oraz (ii) wspólnego wykorzystywania wymienianych informacji, powinni skontaktować się z:

IBM Corporation
Koordynator współdziałania z oprogramowaniem, Dział 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Informacje takie mogą być udostępnione, o ile spełnione zostaną odpowiednie warunki, w tym, w niektórych przypadkach, zostanie uiszczona stosowna opłata.

Licencjonowany program opisany w niniejszej publikacji oraz wszystkie inne licencjonowane materiały dostępne dla tego programu są dostarczane przez IBM na warunkach określonych w Umowie IBM z Klientem, Międzynarodowej Umowie Licencyjnej IBM na Program lub w innych podobnych umowach zawartych między IBM i użytkownikami.

Wszelkie dane dotyczące wydajności zostały zebrane w kontrolowanym środowisku. W związku z tym rezultaty uzyskane w innych środowiskach operacyjnych mogą się znacząco różnić. Niektóre pomiary mogły być dokonywane na systemach będących w fazie rozwoju i nie ma gwarancji, że pomiary wykonane na ogólnie dostępnych systemach dadzą takie same wyniki. Niektóre z pomiarów mogły być estymowane przez ekstrapolację. Rzeczywiste wyniki mogą być inne. Użytkownicy powinni we własnym zakresie sprawdzić odpowiednie dane dla ich środowiska.

Informacje dotyczące produktów innych niż produkty IBM pochodzą od dostawców tych produktów, z opublikowanych przez nich zapowiedzi lub innych powszechnie dostępnych źródeł. Firma IBM nie testowała tych produktów i nie może potwierdzić dokładności pomiarów wydajności, kompatybilności ani żadnych innych danych związanych z tymi produktami. Pytania dotyczące możliwości produktów innych podmiotów należy kierować do dostawców tych produktów.

Wszelkie stwierdzenia dotyczące przyszłych kierunków rozwoju i zamierzeń IBM mogą zostać zmienione lub wycofane bez powiadomienia.

Publikacja ta zawiera przykładowe dane i raporty używane w codziennych operacjach działalności gospodarczej. W celu kompleksowego ich zilustrowania podane przykłady zawierają nazwiska osób prywatnych, nazwy przedsiębiorstw oraz nazwy produktów. Wszystkie te nazwy/nazwiska są fikcyjne i jakiekolwiek podobieństwo do istniejących nazw/nazwisk i adresów jest całkowicie przypadkowe.

LICENCJA W ZAKRESIE PRAW AUTORSKICH:

Niniejsza publikacja zawiera przykładowe aplikacje w kodzie źródłowym, ilustrujące techniki programowania w różnych systemach operacyjnych. Użytkownik może kopiować, modyfikować i dystrybuować te programy przykładowe w dowolnej formie bez uiszczania opłat na rzecz IBM, w celu projektowania, używania, sprzedaży lub dystrybucji aplikacji zgodnych z aplikacyjnym interfejsem programistycznym dla tego systemu operacyjnego, dla którego napisane zostały programy przykładowe. Programy przykładowe nie zostały gruntownie przetestowane. IBM nie może zatem gwarantować ani sugerować niezawodności, użyteczności i funkcjonalności tych programów.

W przypadku przeglądania niniejszych informacji w formie elektronicznej, zdjęcia i kolorowe ilustracje mogą nie być wyświetlane.

Informacje dotyczące interfejsu programistycznego

Informacje dotyczące interfejsu programistycznego, o ile są udostępniane, mają być pomocne podczas tworzenia oprogramowania aplikacji do użytku z tym programem.

Podręcznik ten zawiera informacje na temat planowanych interfejsów programistycznych, które umożliwiają klientom pisanie programów w celu uzyskania dostępu do usług IBM WebSphere MQ.

Informacje te mogą również zawierać informacje na temat diagnostyki, modyfikacji i strojenia. Tego typu informacje są udostępniane jako pomoc przy debugowaniu aplikacji.

Ważne: Informacji na temat diagnostyki, modyfikacji i strojenia nie należy używać jako interfejsu programistycznego, ponieważ może on ulec zmianie.

Znaki towarowe

IBM, logo IBM , ibm.com, są znakami towarowymi IBM Corporation, zarejestrowanymi w wielu systemach prawnych na całym świecie. Aktualna lista znaków towarowych IBM jest dostępna w serwisie WWW, w sekcji "Copyright and trademark information"(Informacje o prawach autorskich i znakach towarowych), pod adresemwww.ibm.com/legal/copytrade.shtml. Nazwy innych produktów lub usług mogą być znakami towarowymi IBM lub innych podmiotów.

Microsoft oraz Windows są znakami towarowymi Microsoft Corporation w Stanach Zjednoczonych i/lub w innych krajach.

UNIX jest zastrzeżonym znakiem towarowym The Open Group w Stanach Zjednoczonych i/lub w innych krajach.

Linux jest zastrzeżonym znakiem towarowym Linusa Torvaldsa w Stanach Zjednoczonych i/lub w innych krajach.

Ten produkt zawiera oprogramowanie opracowane przez Eclipse Project (http://www.eclipse.org/).

Java oraz wszystkie znaki towarowe i logo dotyczące języka Java są znakami towarowymi lub zastrzeżonymi znakami towarowymi Oracle i/lub przedsiębiorstw afiliowanych Oracle.

Numer pozycji:

(1P) P/N: