開發人員指南

AWS IoT Core



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS IoT Core: 開發人員指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 AWS IoT？ 1
您的裝置和應用程式如何存取 AWS IoT 2
AWS IoT 可以做什麼 2
產業中的 IoT 2
家庭自動化中的 IoT 3
AWS IoT 運作方式 4
IoT 的宇宙 4
AWS IoT 服務概觀 6
AWS IoT Core 服務 10
進一步了解 AWS IoT 13
的訓練資源 AWS IoT 13
AWS IoT 資源和指南 13
AWS IoT 在社交媒體中 14
AWS AWS IoT Core 規則引擎使用的 服務 15
支援的通訊協定 AWS IoT Core 16
AWS IoT 主控台中的新功能 16
圖例 19
使用 AWS SDKs 19
入門教學課程 21
將您的第一個裝置連接至 AWS IoT Core 21
設定 AWS 帳戶 23
註冊 AWS 帳戶 23
開啟 AWS IoT 主控台 23
互動式教學課程 23
連線 IoT 裝置 24
儲存裝置離線狀態 25
將裝置資料路由到服務 26
快速連線教學課程 27
步驟 1. 開始教學課程 28
步驟 2. 建立物件 28
步驟 3。將檔案下載到您的裝置 31
步驟 4. 執行範例 33
步驟 5. 深入探索 37
測試連線能力 37

進階連線教學課程 43
哪種裝置選項最適合您？ 44
建立 AWS IoT 資源 45
設定您的裝置 48
使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息 84
在 MQTT 用戶端中檢視 MQTT 訊息 85
從 MQTT 用戶端發佈 MQTT 訊息 87
在 MQTT 用戶端中測試共享訂閱 89
AWS IoT 教學課程 91
使用 AWS IoT 裝置用戶端建置示範 91
使用 AWS IoT 裝置用戶端建置示範的先決條件 92
準備使用 IoT 裝置用戶端 94
安裝和設定 IoT 裝置用戶端 106
使用 MQTT 與裝置用戶端通訊 118
使用 裝置用戶端執行 IoT 任務 136
清除 150
使用 AWS IoT 裝置SDKs建置解決方案 159
使用 AWS IoT 裝置SDKs開始建置解決方案 159
AWS IoT Core 使用 裝置 SDK 將 AWS IoT 裝置連線至 159
建立 AWS IoT 規則將裝置資料路由到其他 服務 181
在裝置離線時保留裝置狀態 220
為 建立自訂授權方 AWS IoT Core 247
使用 AWS IoT 和 Raspberry Pi 監控土壤濕度 263
連線至 AWS IoT Core 276
AWS IoT Core：控制平面端點 276
AWS IoT 裝置端點 278
AWS IoT Core for LoRaWAN 閘道和裝置 279
連線至 AWS IoT Core 服務端點 280
AWS CLI 適用於 的 AWS IoT Core 280
AWS SDKs 281
AWS 行動 SDKs 286
AWS IoT Core 服務的 REST APIs 287
將裝置連接至 AWS IoT 288
AWS IoT 裝置資料和服務端點 288
AWS IoT 裝置 SDKs 290
裝置通訊協定 292

MQTT 主題 349
網域組態 369
連線至 AWS IoT FIPS 端點 394
AWS IoT Core：控制平面端點 394
AWS IoT Core：資料平面端點 395
AWS IoT Core - 登入資料 提供者端點 395
AWS IoT Device Management：任務資料端點 395
AWS IoT Device Management：Fleet Hub 端點 396
AWS IoT Device Management：安全通道端點 396
AWS IoT Device Management - 受管整合端點 396
管理裝置 398
登錄檔 398
建立物件 399
列出物件 400
說明物件 402
更新物件 403
刪除物件 403
將主體連接至物件 403
列出與委託人相關聯的物件 404
列出與物件相關聯的主體 405
列出與委託人 V2 相關聯的物件 405
列出與物件 V2 相關聯的主體 406
將主體與物件分離 407
物件類型 407
建立物件類型 408
列出物件類型 408
描述物件類型 409
將物件類型與物件建立關聯 410
更新物件類型 410
棄用物件類型 410
刪除物件類型 412
靜態物件群組 412
建立靜態物件群組 414
描述物件群組 415
新增一個物件到一個靜態的物件群組 416
從靜態物件群組中移除一個物件 416

列出物件群組中的物件 416
列出物件群組 417
列出物件的群組 419
更新靜態物件群組 420
刪除物件群組 420
將政策附加到靜態物件群組 421
將政策與靜態物件群組分離 421
列出連線至靜態物件群組的政策 422
列出政策所在的群組 422
為物件取得有效政策 423
MQTT 動作的測試授權 423
動態物件群組 425
動態物件群組的使用案例 426
建立動態物件群組 427
描述動態物件群組 427
更新動態物件群組 429
刪除動態物件群組 429
動態和靜態物件群組限制 429
動態物件群組限制 430
將物件與連線建立關聯 432
使用案例 432
如何將物件與連線建立關聯 433
新增傳播屬性 435
AWS 管理主控台 436
AWS CLI 437
標籤資源 439
標籤基本概念 439
標籤的限制與上限 440
使用 IAM 政策標記 441
帳單群組 443
檢視成本分配和用量資料 444
安全 446
中的安全性 AWS IoT 447
身分驗證 448
X.509 憑證概觀 448
伺服器驗證 448

用戶端身分驗證 451
自訂身分驗證和授權 487
Authorization 512
AWS 訓練和認證 515
AWS IoT Core 政策 515
使用 AWS IoT Core 登入資料提供者授權直接呼叫 AWS 服務 594
利用 IAM 跨帳戶存取 600
資料保護 602
中的資料加密 AWS IoT 603
中的傳輸安全性 AWS IoT Core 603
資料加密 609
身分與存取管理 624
目標對象 624
使用 IAM 身分來驗證 625
使用政策管理存取權 626
AWS IoT 如何使用 IAM 627
身分型政策範例 658
AWS 受管政策 661
疑難排解 675
記錄和監控 677
監控工具 677
法規遵循驗證 679
恢復能力 679
AWS IoT Core 搭配 VPC 端點使用 680
建立 AWS IoT Core 控制平面的 VPC 端點 680
為 AWS IoT Core 資料平面建立 VPC 端點 681
為 AWS IoT Core 登入資料提供者建立 VPC 端點 681
建立 Amazon VPC 介面端點 682
設定私有託管區域 684
透過 VPC AWS IoT Core 端點控制對 的存取 686
限制 687
使用 擴展 VPC 端點 AWS IoT Core 688
搭配使用自訂網域與 VPC 端點 688
的 VPC 端點可用性 AWS IoT Core 688
AWS IoT Device Management 搭配 VPC 端點使用 689
基礎設施安全性 694

安全監控 694
安全最佳實務 694
在 中保護 MQTT 連線 AWS IoT 695
讓裝置的時鐘保持同步 697
驗證伺服器憑證 697
使用每個裝置單一身分 698
使用秒 AWS 區域 作為備份 698
使用及時佈建 698
執行 AWS IoT Device Advisor 測試的許可 698
Device Advisor 跨服務預防混淆代理人 700
AWS 訓練和認證 701
監控 AWS IoT 702
設定 AWS IoT 記錄 703
設定 V2 登入 AWS IoT 703
設定記錄角色和政策 704
在 AWS IoT （主控台） 中設定記錄 706
在 AWS IoT (CLI) 中設定帳戶和事件層級登入 708
在 AWS IoT (CLI) 中設定資源特定覆寫 710
日誌層級 712
使用 Amazon CloudWatch 監控 AWS IoT 警示和指標 713
使用 AWS IoT 指標 713
建立 CloudWatch 警示 714
指標和維度 718
AWS IoT 使用 CloudWatch Logs 進行監控 736
在 CloudWatch 主控台中檢視 AWS IoT 日誌 736
CloudWatch Logs AWS IoT 日誌項目 737
將裝置端日誌上傳到 Amazon CloudWatch 793
運作方式 794
使用 AWS IoT 規則上傳裝置端日誌 794
記錄 AWS IoT API 呼叫 804
AWS IoT CloudTrail 中的資訊 804
了解 AWS IoT 日誌檔案項目 805
規則 807
存取控制 808
授予 AWS IoT 規則所需的存取權 808
撤銷規則引擎存取權 810

限制 HTTP 動作流量至規則引擎 IPv6 範圍 810
傳送角色許可 813
建立規則 814
建立規則 （主控台） 815
建立規則 (CLI) 816
管理規則 819
標記規則 819
檢視規則 821
刪除規則 821
AWS IoT 規則動作 821
Apache Kafka 824
CloudWatch 警示 835
CloudWatch Logs 837
CloudWatch 指標 839
DynamoDB 841
DynamoDBv2 844
Elasticsearch 846
HTTP 849
InfluxDB 865
AWS IoT SiteWise 880
Firehose 885
Kinesis Data Streams 887
Lambda 889
Location 892
OpenSearch 896
重新發佈 898
S3 901
SNS 903
SQS 905
步驟函數 908
Timestream 909
規則疑難排解 916
存取跨帳戶資源 916
先決條件 917
Amazon SQS 的跨帳戶設定 917
Amazon SNS 的跨帳戶設定 919

Amazon S3 的跨帳戶設定 920
的跨帳戶設定 AWS Lambda 922
錯誤處理 (錯誤動作) 924
錯誤動作訊息格式 925
錯誤動作範例 926
基本擷取 927
使用基本擷取 928
AWS IoT SQL 參考 928
SELECT 子句 930
FROM 子句 932
SET 子句 933
WHERE 子句 936
資料類型 937
運算子 941
函數 950
文字 1016
案例陳述式 1017
JSON Extensions 1018
替代範本 1020
巢狀物件查詢 1025
二進位承載 1027
SQL 版本 1033
AWS IoT 規則動作中的預留字元序列 1034
Shadows 1035
使用影子 1035
選擇使用已命名或未命名的影子 1035
存取影子 1036
在裝置、應用程式和其他雲端服務中使用影子 1037
訊息順序 1037
裁剪影子訊息 1039
在裝置中使用影子 1040
在第一次連線至 時初始化裝置 AWS IoT 1041
在裝置連線到 時處理訊息 AWS IoT 1042
當裝置重新連線至 時處理訊息 AWS IoT 1043
在應用程式和服務中使用影子 1043
在連線至 時初始化應用程式或服務 AWS IoT 1044

應用程式或服務連線到 時，處理狀態會變更 AWS IoT 1045
偵測裝置是否已連線 1045
模擬 Device Shadow 服務通訊 1047
準備模擬 1047
初始化裝置 1047
從應用程式傳送更新 1051
回應裝置中的更新 1054
觀察應用程式中的更新 1058
超越模擬 1060
與影子互動 1060
通訊協定支援 1061
請求和報告狀態 1061
更新影子 1061
擷取影子文件 1065
刪除影子資料 1066
Device Shadow REST API 1069
GetThingShadow 1070
UpdateThingShadow 1071
DeleteThingShadow 1072
ListNamedShadowsForThing 1073
Device Shadow MQTT 主題 1075
/get 1076
/get/accepted 1076
/get/rejected 1077
/update 1078
/update/delta 1080
/update/accepted 1081
/update/documents 1082
/update/rejected 1083
/delete 1084
/delete/accepted 1084
/delete/rejected 1085
Device Shadow 服務文件 1086
影子文件範例 1087
文件屬性 1093
差量狀態 1094

版本控制影子文件 1096
影子文件中的用戶端字符 1096
空白影子文件屬性 1096
影子文件中的陣列值 1097
Device Shadow 錯誤訊息 1098
軟體套件目錄 1100
準備使用軟體套件目錄 1100
套件版本生命週期 1100
套件版本命名慣例 1102
預設版本 1102
版本屬性 1103
軟體物料清單 1103
啟用 AWS IoT 機群索引 1106
預留已命名影子 1106
刪除軟體套件 1108
準備安全性 1108
以資源為基礎的身分驗證 1108
AWS IoT 部署套件版本的任務權限 1110
AWS IoT 更新預留具名影子的任務權限 1111
AWS IoT 從 Amazon S3 下載的任務許可 1113
更新套件版本之軟體物料清單的許可 1113
準備機群索引 1116
將 $package 影子設定為資料來源 1116
控制台顯示的指標 1117
查詢模式 1118
透過 getBucketsAggregation 收集套件版本分佈 1120
準備 AWS IoT 任務 1121
AWS IoT 任務的替代參數 1121
準備任務文件和套件版本以進行部署 1125
部署時命名套件和版本 1128
透過 AWS IoT 動態物件群組將任務設為目標 1128
預留已命名影子和套件版本 1128
解除安裝軟體套件 1129
開始使用 1130
建立套件和版本 1130
部署套件版本 1132

將套件版本建立關聯 1134
任務 1136
存取 AWS IoT 任務 1136
AWS IoT 任務區域和端點 1136
什麼是遠端操作？ 1136
使用 AWS IoT Device Management 任務進行遠端操作的優點 1137
什麼是 AWS IoT 任務？ 1138
任務的重要概念 1139
任務和任務執行狀態 1142
管理任務 1147
任務的程式碼簽署 1147
任務文件 1147
預先簽章的 URL 1147
檔案上傳的預先簽章 URL 1150
使用 Amazon S3 版本控制的預先簽章 URL 1150
使用主控台建立和管理任務 1152
使用 CLI 建立和管理任務 1155
任務範本 1166
自訂和 AWS 受管範本 1166
使用 AWS 受管範本 1166
建立自訂任務範本 1184
任務 組態 1192
任務組態的運作方式 1192
指定其他組態 1204
裝置和任務 1213
設定裝置與任務合作 1215
裝置工作流程 1216
任務工作流程 1217
任務通知 1221
AWS IoT 任務 API 操作 1229
任務管理、控制 API 和控制資料類型 1231
任務裝置 MQTT、HTTPS API 操作和資料類型 1250
為任務加密使用者和裝置 1264
AWS IoT 任務的必要政策類型 1264
授權任務使用者和雲端服務 1265
授權裝置使用任務 1277

AWS IoT 任務限制 1280
任務執行限制 1281
作用中和並行任務限制 1282
命令 1285
快速入門 1286
命令概念和狀態 1286
命令關鍵概念 1287
命令狀態 1290
命令執行狀態 1291
命令工作流程 1295
建立和管理命令 1295
選擇目標並訂閱主題 1296
開始和監控命令執行 1297
（選用） 啟用命令事件的通知 1298
建立和管理命令 1299
建立命令資源 1300
擷取命令的相關資訊 1305
在 中列出命令 AWS 帳戶 1306
更新命令資源 1308
棄用或還原命令資源 1310
刪除命令資源 1310
開始和監控命令執行 1312
啟動命令執行 1312
更新命令執行的結果 1318
擷取命令執行 1323
使用 MQTT 測試用戶端檢視命令更新 1327
在 中列出命令執行 AWS 帳戶 1328
刪除命令執行 1330
取代指令資源 1332
關鍵考量 1332
棄用命令資源 （主控台） 1332
棄用命令資源 (CLI) 1332
檢查棄用時間和狀態 1333
還原命令資源 1333
安全通道 1335
什麼是安全通道？ 1335

安全通道概念 1335
安全通道運作方式 1336
安全通道生命週期 1337
安全通道教學課程 1338
本節中的教學課程 1338
開啟通道並啟動遠端裝置的 SSH 工作階段 1339
開啟遠端裝置的通道並使用瀏覽器型 SSH 1355
本機代理 1358
如何使用本機代理 1359
為使用 Web 代理的裝置配置本機代理 1365
多工處理和同步 TCP 連線 1372
多工處理多個資料串流 1372
使用同時 TCP 連線 1375
設定遠端裝置並使用 IoT 代理程式 1378
IoT Agent Snippet 1378
控制 Kibana 的存取 1380
通道存取先決條件 1380
通道存取政策 1380
解決安全通道連線問題 1387
無效的用戶端存取字符錯誤 1388
用戶端字符不相符錯誤 1388
遠端裝置連線問題 1389
裝置佈建 1392
在 中佈建裝置 AWS IoT 1393
機群佈建 API 1394
使用機群佈建來佈建沒有裝置憑證的裝置 1394
透過要求佈建 1395
由信任的使用者佈建 1397
搭配 CLI AWS 使用預先佈建掛鉤 1399
佈建具有裝置憑證的裝置 1402
單一物件佈建 1403
即時佈建 1404
大量註冊 1409
佈建範本 1410
參數部分 1411
資源部分 1411

大量註冊的範本範例 1417
即時佈建 (JITP) 的範本範例 1418
機群佈建 1420
預先佈建掛接 1423
預先佈建掛接輸入 1424
預先佈建掛接傳回值 1424
預先佈建掛接 Lambda 範例 1425
使用憑證提供者進行自我管理 AWS IoT Core 的憑證簽署 1427
自我管理憑證簽署如何在機群佈建中運作 1428
憑證提供者 Lambda 函數輸入 1430
憑證提供者 Lambda 函數傳回值 1430
Lambda 函數範例 1430
機群佈建的自我管理憑證簽署 1432
AWS CLI 憑證提供者的 命令 1433
為安裝裝置的使用者建立 IAM 政策和角色 1436
為安裝裝置的使用者建立 IAM 政策 1436
為安裝裝置的使用者建立 IAM 角色 1437
更新現有政策以授權新範本 1438
裝置佈建 MQTT API 1439
CreateCertificateFromCsr 1440
CreateKeysAndCertificate 1442
RegisterThing 1444
機群索引 1447
管理索引更新 1447
查詢特定裝置的連線狀態 1447
跨資料來源搜尋 1447
查詢彙總資料 1447
使用機群指標監控彙總資料和建立警示 1448
管理機群索引 1448
物件索引 1448
物件群組索引 1449
受管欄位 1449
自訂欄位 1451
管理物件索引 1452
管理物件群組索引 1469
裝置連線狀態 1471

運作方式 1471
功能 1471
優勢 1472
先決條件 1472
範例 1473
查詢彙總資料 1475
GetStatistics 1475
GetCardinality 1478
GetPercentiles 1479
GetBucketsAggregation 1481
Authorization 1482
查詢語法 1482
支援的功能 1483
不支援的功能 1483
備註 1484
範例物件查詢 1484
範例物件群組查詢 1488
索引位置資料 1490
支援的資料格式 1490
如何索引位置資料 1491
更新物件索引組態 1491
範例地理查詢 1494
入門教學課程 1495
機群指標 1499
入門教學課程 1500
管理機群指標 1506
MQTT 型檔案交付 1513
什麼是串流？ 1513
管理串流 1514
授與您的裝置許可 1515
將您的裝置連線至 AWS IoT 1516
TagResource 用量 1516
在裝置中使用 AWS IoT MQTT 型檔案交付 1517
使用 DescribeStream 以取得串流資料 1517
從串流檔案取得資料區塊 1519
處理 AWS IoT MQTT 型檔案交付的錯誤 1524

FreeRTOS OTA 中的範例使用案例 1526
Device Advisor 1527
設定 1528
建立 IoT 物件 1528
建立要用作裝置角色的 IAM 角色 1529
為要使用 Device Advisor 的 IAM 使用者建立自訂受管政策 1532
建立要使用 Device Advisor 的 IAM 使用者 1532
設定您的裝置 1535
在主控台中開始使用 Device Advisor 1536
Device Advisor 工作流程 1544
先決條件 1544
建立測試套件定義 1544
取得測試套件定義 1547
取得測試端點 1547
啟動測試套件執行 1548
取得測試套件執行 1548
停止測試套件執行 1549
取得成功資格測試套件執行的資格報告 1549
Device Advisor 詳細主控台工作流程 1550
先決條件 1550
建立測試套件定義 1550
啟動測試套件執行 1558
停止測試套件執行 (選用) 1560
檢視測試套件執行詳細資訊和記錄 1561
下載 AWS IoT 資格報告 1562
長時間測試主控台工作流程 1563
Device Advisor VPC 端點 (AWS PrivateLink) 1571
AWS IoT Core Device Advisor VPC 端點的考量事項 1571
建立 的介面 VPC 端點 AWS IoT Core Device Advisor 1572
透過 VPC AWS IoT Core Device Advisor 端點控制對 的存取 1572
Device Advisor 測試案例 1574
Device Advisor 測試案例以符合 AWS Device Qualification Program 的資格。 1574
TLS 1575
MQTT 1581
影子 1593
任務執行 1596

許可和政策 1598
長時間測試 1598
裝置位置 1617
測量類型和求解器 1617
AWS IoT Core 裝置位置的運作方式 1619
如何使用 AWS IoT Core 裝置位置 1620
解析 IoT 裝置的位置 1621
解析裝置位置 (主控台) 1621
解析裝置位置 (API) 1624
對解析位置時發生的錯誤進行疑難排解 1626
使用 MQTT 主題解析裝置位置 1627
裝置位置 MQTT 主題的格式 1627
裝置位置 MQTT 主題的政策 1628
裝置位置主題和承載 1629
位置求解器和裝置承載 1634
基於 Wi-Fi 的求解器 1635
行動網路求解器 1636
IP 反向查詢求解器 1640
求解器 1641
多影格 GNSS 求解器 1642
Amazon Sidewalk 啟用裝置的 BLE 求解器 1644
AWS IoT Core 裝置位置的進階組態 1644
WiFiCellular 組態 1645
使用 AdvancedConfiguration 的 Wi-Fi 承載範例 1645
使用 AdvancedConfiguration 的行動承載範例 1646
事件訊息 1649
如何產生事件訊息 1649
接收事件訊息的政策 1649
啟用 的事件 AWS IoT 1650
登錄檔事件 1655
物件事件 1655
物件類型事件 1656
物件群組事件 1660
任務事件 1665
命令執行事件 1670
生命週期事件 1671

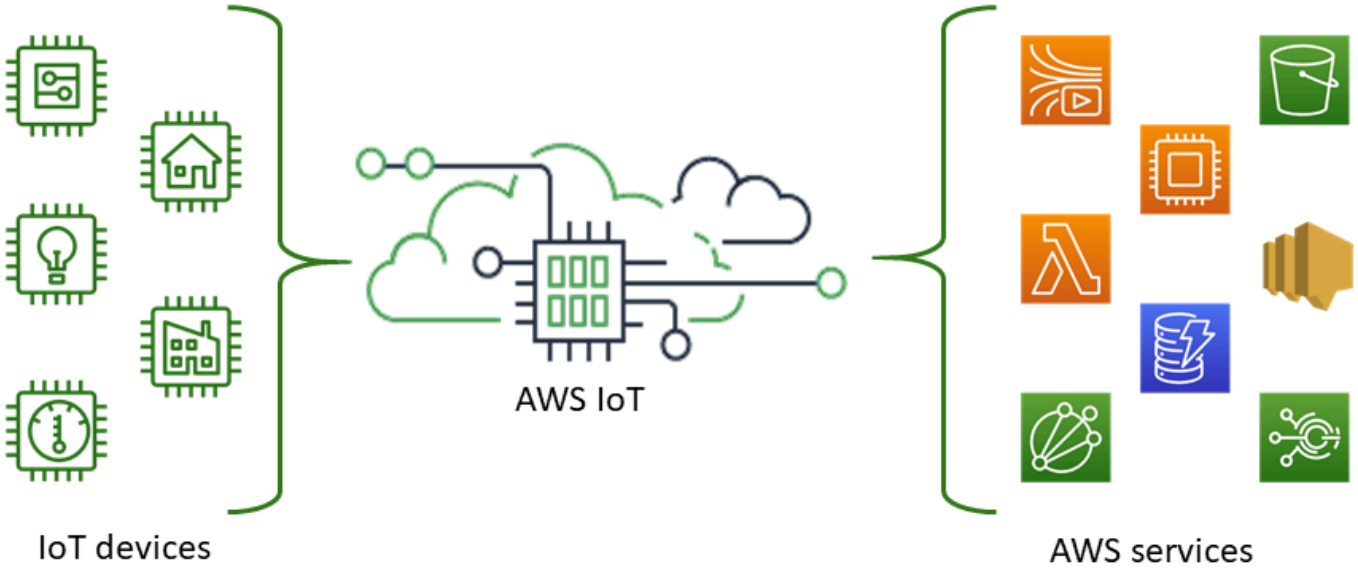
連線/中斷連線事件 1672
連線嘗試失敗事件 1676
訂閱/取消訂閱事件 1677
疑難排解 1680
AWS IoT Core 故障診斷指南 1680
診斷連線問題 1681
診斷規則問題 1684
診斷影子的問題 1686
診斷串流限制 1687
對裝置機群中斷連線進行疑難排解 1688
AWS IoT Device Management 疑難排解指南 1688
AWS IoT 任務故障診斷 1689
AWS IoT 命令故障診斷 1692
機群索引故障診斷 1695
AWS IoT 裝置管理軟體套件目錄故障診斷 1697
AWS IoT Device Advisor 疑難排解指南 1704
AWS IoT 錯誤 1706
AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端 1708
AWS IoT 裝置 SDKs 1708
AWS IoT 適用於 Embedded C 的 裝置 SDK 1710
AWS 行動SDKs 1711
AWS IoT 裝置用戶端 1712
舊型 AWS IoT 裝置 SDKs版本 1712
程式碼範例 1713
基本概念 1714
您好 AWS IoT 1714
了解基本概念 1723
動作 1836
案例 1943
開始使用物聯網裝置保護 1943
開始使用物聯網簡訊 1963
AWS IoT 配額 1974
AWS IoT Core 定價 1975
支援通訊協定的FAQs 1976
什麼是 AWS JSON 和 Smithy RPC v2 CBOR 通訊協定，它們有何不同？ 1976
如何開始使用 Smithy RPC v2 CBOR 通訊協定？ 1976

啟用 Smithy RPC v2 CBOR 通訊協定有哪些風險？ 1977
如果我的解決方案不支援 Smithy RPC v2 CBOR，該怎麼辦？ 1977
哪些 AWS 區域支援 Smithy RPC v2 CBOR 通訊協定？ 1977
升級至支援 Smithy RPC v2 CBOR 的 SDK 版本時，可以預期哪些延遲改善？ 1977
AWS JSON 通訊協定是否會棄用？ 1977
哪裡可以找到有關 Smithy RPC v2 CBOR 的詳細資訊？ 1977
..... mcmlxxviii

什麼是 AWS IoT？

AWS IoT 提供將 IoT 裝置連接到其他裝置和 AWS 雲端服務的雲端服務。 AWS IoT 提供裝置軟體，可協助您將 IoT 裝置整合到 AWS IoT解決方案。如果您的裝置可以連接到 AWS IoT， AWS IoT 可以將其連接到 AWS 提供的雲端服務。

如需實作簡介 AWS IoT，請造訪 入門教學課程。



使用 AWS IoT，您可以為您的解決方案選取最適當且up-to-date技術。為了協助您管理和支援 欄位中的 IoT 裝置， AWS IoT Core 支援這些通訊協定：

- MQTT (訊息佇列和遙測傳輸)
- 透過 WSS 的 MQTT (Websocket 安全)
- HTTPS (超文字傳輸通訊協定：安全)
- LoRaWAN (長距離廣域網路)

AWS IoT Core 訊息中介裝置支援透過 WSS 通訊協定使用 MQTT 和 MQTT 來發佈和訂閱訊息的裝置和用戶端。其也支援使用 HTTPS 通訊協定發佈訊息的裝置和用戶端。

AWS IoT Core for LoRaWAN 可協助您連接和管理無線 LoRaWAN （低功耗長距離廣域網路） 裝置。AWS IoT Core for LoRaWAN 會取代您開發和操作 LoRaWAN 網路伺服器 (LNS) 的需求。

如果您不需要裝置通訊、規則或任務等 AWS IoT 功能，請參閱AWS 簡訊以取得可能更符合您需求的其他 AWS IoT 簡訊服務相關資訊。

您的裝置和應用程式如何存取 AWS IoT

AWS IoT 為 提供下列界面AWS IoT 教學課程：

- AWS IoT 裝置 SDKs - 在裝置上建置應用程式，以傳送訊息至 並從中接收訊息 AWS IoT。如需詳細資訊，請參閱AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端。
- AWS IoT Core for LoRaWAN — 使用 for LoRaWAN 連接和管理長距離 WAN (LoRaWAN) 裝置和閘道。 AWS IoT Core LoRaWAN
- AWS Command Line Interface (AWS CLI)—在 Windows、macOS 和 Linux AWS IoT 上執行 的命令。這些命令可讓您建立並管理物件物件、憑證、規則、任務和政策。若要開始使用，請參閱《AWS Command Line Interface 使用者指南》。如需 命令的詳細資訊 AWS IoT，請參閱《 AWS CLI 命令參考》中的 iot。
- AWS IoT API - 使用 HTTP 或 HTTPS 請求建置 IoT 應用程式。這些 API 動作可讓您以程式設計方式建立並管理物件物件、憑證、規則和政策。如需 API 動作的詳細資訊 AWS IoT，請參閱 AWS IoT API 參考中的動作。
- AWS SDKs - 使用特定語言 APIs 建置 IoT 應用程式。這些 SDK 整合了 HTTP/HTTPS API，可讓您以任何支援的語言編寫程式。如需詳細資訊，請參閱AWS 開發套件與工具。

您也可以 AWS IoT 透過 AWS IoT 主控台存取 ，該主控台提供圖形化使用者介面 (GUI)，您可以透過該界面來設定和管理 IoT 解決方案的物件、憑證、規則、任務、政策和其他元素。

AWS IoT 可以做什麼

本主題說明您可能需要 AWS IoT 支援的一些解決方案。

產業中的 IoT



這些是一些工業使用案例 AWS IoT 解決方案的範例，這些解決方案會套用 IoT 技術來改善工業程序的效能和生產力。

適用於工業使用案例的解決方案

- 使用 AWS IoT 在工業操作中建置預測品質模型

  了解如何從工業操作 AWS IoT 收集和分析資料，以建置預測品質模型。進一步了解

- 用於 AWS IoT 支援工業操作的預測性維護

  了解 AWS IoT 如何協助規劃預防性維護，以減少意外停機時間。進一步了解

家庭自動化中的 IoT



這些是一些適用於家庭自動化使用案例 AWS IoT 的解決方案範例，這些解決方案會套用 IoT 技術來建置可擴展的 IoT 應用程式，以使用連線家庭裝置自動化家庭活動。

家庭自動化解決方案

- 在連線家庭 AWS IoT 中使用

  了解如何 AWS IoT 提供整合式家庭自動化解決方案。

- 使用 AWS IoT 提供家庭安全和監控

  了解如何 AWS IoT 將機器學習和邊緣運算套用至您的家庭自動化解決方案。

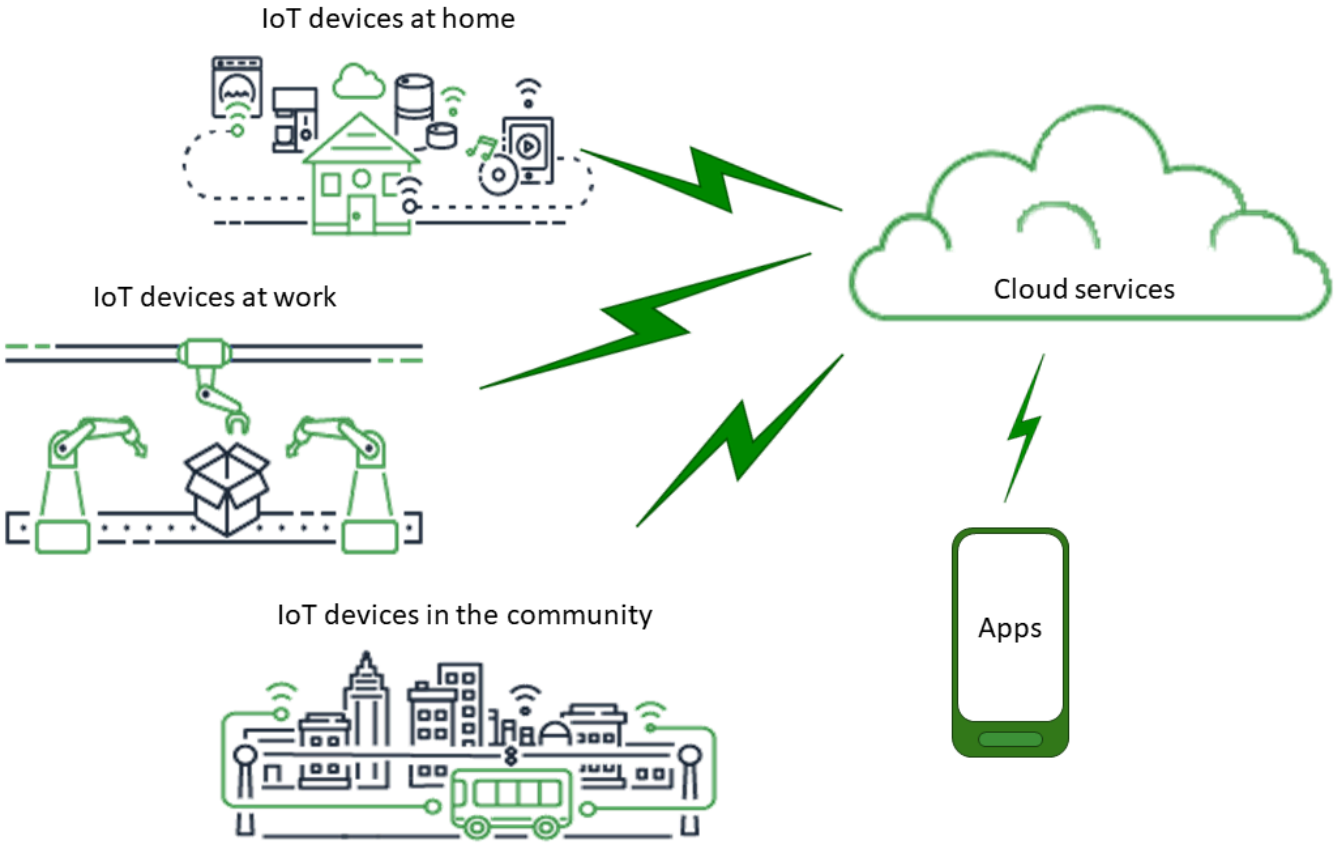
如需工業、消費者和商業使用案例的解決方案清單，請參閱 AWS IoT 解決方案儲存庫。

AWS IoT 運作方式

AWS IoT 提供雲端服務和裝置支援，您可以用來實作 IoT 解決方案。 AWS 提供許多雲端服務以支援以 IoT 為基礎的應用程式。因此，為了幫助您了解從何處開始，本節提供了基本概念的圖表和定義，以便向您介紹 IoT 世界。

IoT 的宇宙

一般而言，物聯網 (IoT) 是由此圖表所示的關鍵元件所組成。



應用程式

應用程式可讓最終使用者存取 IoT 裝置，以及這些裝置連接的雲端服務提供的功能。

雲端服務

雲端服務是連接到網際網路之分配式、大規模的資料儲存和處理服務。範例包括：

- IoT 連接和管理服務

AWS IoT 是 IoT 連線和管理服務的範例。

- 運算服務，例如 Amazon Elastic Compute Cloud 和 AWS Lambda
- 資料庫服務，例如 Amazon DynamoDB

通訊

裝置會使用各種技術和通訊協定與雲端服務進行通訊。範例包括：

- Wi-Fi/寬頻網際網路
- 寬頻行動數據
- 窄頻行動數據
- 長距離廣域網絡 (LoRaWAN)
- 專屬 RF 通訊

Devices

裝置是一種管理介面和通訊的硬體類型。裝置通常位於所監控和控制的真實世界介面附近。裝置可包含運算和儲存資源，例如微控制器、CPU、記憶體。範例包括：

- Raspberry Pi
- Arduino
- 語音介面助理
- LoRaWAN 和裝置
- Amazon Sidewalk 裝置
- 自訂 IoT 裝置

介面

介面是將裝置連接到實體世界的元件。

- 使用者介面

  允許裝置和使用者彼此通訊的元件。

  - 文字輸入介面

讓使用者可以和裝置通訊

範例：鍵盤、按鈕

- 輸出介面

  讓裝置可以和使用者通訊

  範例：字母數字顯示器、圖形顯示器、指示燈、鬧鐘

- 感測器

  測量或感測外部世界中的某些東西，並以裝置能理解的方式輸入的元件。範例包括：

  - 溫度感測器 (將溫度轉換為類比或數位訊號)
  - 濕度感測器 (將相對濕度轉換為類比或數位訊號)
  - 類比數位轉換器 (將類比電壓轉換為數值)
  - 超音波距離測量單元 (將距離轉換為數值)
  - 光學感測器 (將光電位轉換為數值)
  - 相機 (將影像資料轉換為數位資料)

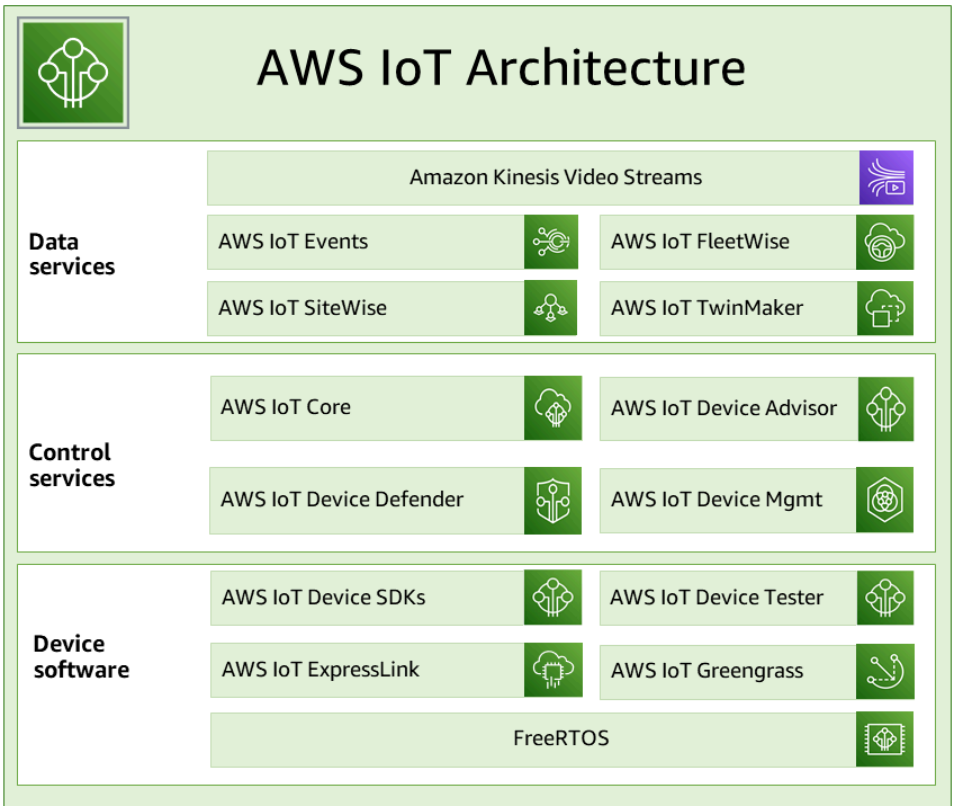
- 致動器

  輸出元件，裝置可使用該元件控制外部世界的東西。範例包括：

  - 步進馬達 (將電信號轉換為移動)
  - 繼電器 (控制高電壓和電流)

AWS IoT 服務概觀

在 IoT 世界中， AWS IoT 提供的服務支援與世界互動的裝置，以及它們與 之間傳遞的資料 AWS IoT。 AWS IoT 是由本圖所示的服務組成，以支援您的 IoT 解決方案。



AWS IoT 裝置軟體

AWS IoT 提供此軟體以支援您的 IoT 裝置。

AWS IoT 裝置 SDKs

AWS IoT 裝置和行動SDKs可協助您有效率地將裝置連線至 AWS IoT。 AWS IoT 裝置和行動SDKs包括開放原始碼程式庫、具有範例的開發人員指南，以及移植指南，讓您可以在選擇的硬體平台上建置創新的 IoT 產品或解決方案。

AWS IoT Device Tester

AWS IoT Device Tester for FreeRTOS 和 AWS IoT Greengrass 是 microcontrollers 的測試自動化工具。 會 AWS IoT Device Tester 測試您的裝置，以判斷裝置是否會執行 FreeRTOS 或 ， AWS IoT Greengrass 並與 AWS IoT 服務互通。

AWS IoT ExpressLink

AWS IoT ExpressLink 支援由 AWS 合作夥伴開發和提供的一系列硬體模組。連線模組包含 已 AWS驗證的軟體，可讓您更快更輕鬆地將裝置安全地連線至雲端，並與各種 AWS 服務無縫整合。如需詳細資訊，請造訪AWS IoT ExpressLink 概覽頁面，或查看AWS IoT ExpressLink 程式設計人員指南。

AWS IoT Greengrass

AWS IoT Greengrass 延伸 AWS IoT 到邊緣裝置，讓他們可以根據其產生的資料在本機採取行動、根據機器學習模型執行預測，以及篩選和彙總裝置資料。 AWS IoT Greengrass 可讓您的裝置收集和分析更接近產生資料的位置的資料、自動回應本機事件，以及安全地與本機網路上的其他裝置通訊。您可以使用 AWS IoT Greengrass ，使用稱為元件的預先建置軟體模組來建置邊緣應用程式，這些模組可將邊緣裝置連線至 AWS 服務或第三方服務。

FreeRTOS

FreeRTOS 是一個適用於微控制器的開放原始碼即時作業系統，可讓您將小型、低功耗邊緣裝置納入您的 IoT 解決方案。FreeRTOS 包含一個內核，以及一組不斷增長，並能支援許多應用程式的程式庫。FreeRTOS 系統可安全地將您的小型低功耗裝置連接至 AWS IoT，並支援執行 AWS IoT Greengrass 且功能更強大的邊緣裝置。

AWS IoT 控制服務

連線至下列 AWS IoT 服務，以管理 IoT 解決方案中的裝置。

AWS IoT Core

AWS IoT Core 是一種受管雲端服務，可讓連線裝置安全地與雲端應用程式和其他裝置互動。 AWS IoT Core 可支援許多裝置和訊息，並可處理這些訊息並將其路由至 AWS IoT 端點和其他裝置。透過 AWS IoT Core，您的應用程式即使未連線也能與所有裝置互動。

AWS IoT Core Device Advisor

AWS IoT Core Device Advisor 是一種以雲端為基礎的全受管測試功能，可在裝置軟體開發期間驗證 IoT 裝置。Device Advisor 提供預先建置的測試，可讓您在將裝置部署至生產環境之前 AWS IoT Core，用來驗證 IoT 裝置是否有可靠且安全的連線能力。

AWS IoT Device Defender

AWS IoT Device Defender 可協助您保護 IoT 裝置機群的安全。 AWS IoT Device Defender 會持續稽核 IoT 組態，以確保它們不會偏離安全最佳實務。 AWS IoT Device Defender 會在偵測到任何可能造成安全風險的 IoT 組態漏洞時傳送提醒，例如在多個裝置之間共用身分憑證，或是具有已撤銷身分憑證的裝置嘗試連線至 AWS IoT Core。

AWS IoT 裝置管理

AWS IoT Device Management 服務可協助您追蹤、監控和管理組成裝置機群的連線裝置數量。AWS IoT Device Management 服務可協助您確保 IoT 裝置在部署後正常運作且安全。它們還提供

安全通道功能，以存取您的裝置、監控其健全狀況、偵測和遠端進行問題疑難排解，以及管理裝置軟體和韌體更新的服務。

AWS IoT 資料服務

分析 IoT 解決方案中裝置的資料，並使用下列 AWS IoT 服務採取適當的動作。

Amazon Kinesis Video Streams

Amazon Kinesis Video Streams 可讓您將即時影片從裝置串流到 AWS 雲端，在雲端中長期存放、加密和編製索引，讓您透過easy-to-use存取資料。 APIs 您可以使用 Amazon Kinesis Video Streams 從數百萬個來源擷取大量即時影片資料，包括智慧型手機、監視攝影機、網路攝影機、汽車內建攝影機、空拍機及其他來源。Amazon Kinesis Video Streams 使您能夠播放影片，進行即時和隨選點播，並通過 Amazon Rekognition Video 和 ML 架構資料庫整合，快速利用視覺化運算和影片分析的方式建置應用程式。您也可以傳送非影片時間序列化資料，例如音訊資料、熱影像、深度資料、RADAR 資料等。

Amazon Kinesis Video Streams with WebRTC

Amazon Kinesis Video Streams with WebRTC 以全受管功能提供符合標準的 WebRTC 實作。您可以使用 Amazon Kinesis Video Streams with WebRTC 安全地即時串流媒體，或在任何攝影機 IoT 裝置與符合 WebRTC 標準的行動或 Web 播放器之間，執行雙向音訊或視訊互動。因為是全受管功能，您不需要建置、執行或擴展任何與 WebRTC 相關的雲端基礎設施，例如訊號或媒體轉送伺服器，即可在應用程式和裝置之間安全地串流媒體。您可以使用 Amazon Kinesis Video Streams with WebRTC，輕鬆建置應用程式進行即時點對點媒體串流，或在攝影機 IoT 裝置、Web 瀏覽器和行動裝置之間進行即時音訊或視訊互動，以滿足各種使用案例。

AWS IoT FleetWise

AWS IoT FleetWise 是一項受管服務，可用來近乎即時地收集車輛資料並將其傳輸至雲端。透過 AWS IoT FleetWise，您可以輕鬆地從使用不同通訊協定和資料格式的車輛收集和組織資料。 AWS IoT Fleetwise 有助於將低級別訊息轉換為人類可讀的值，並標準化雲端資料格式，以便進行資料分析。您還可以定義資料蒐集的方案，控制要蒐集的車輛資料，以及何時將資料傳輸至雲端。

AWS IoT SiteWise

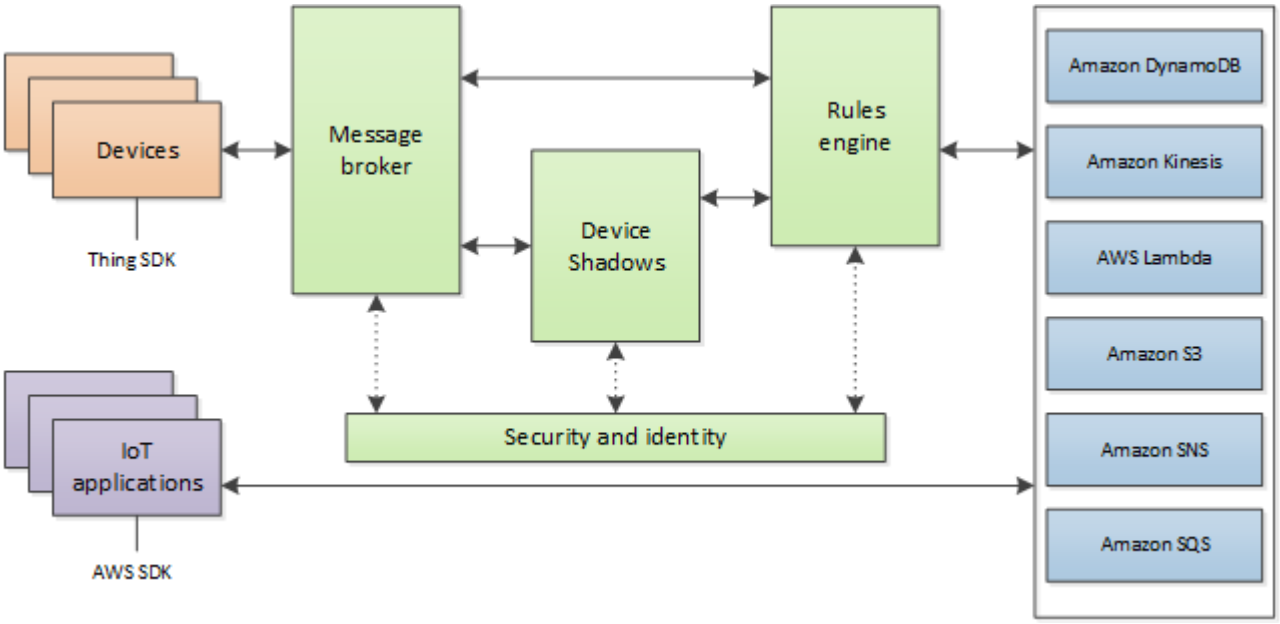
AWS IoT SiteWise 透過提供在您設施中的閘道器上執行的軟體大規模收集、存放、組織和監控 MQTT 訊息或 API 從產業設備傳遞的資料。閘道可安全地連線至您的內部部署資料伺服器，並自動化收集和組織資料並將其傳送至 AWS 雲端的程序。

AWS IoT TwinMaker

AWS IoT TwinMaker 會建置實體和數位系統的營運數位分身。 AWS IoT TwinMaker 會使用來自各種真實世界感應器、攝影機和企業應用程式的測量和分析來建立數位視覺化，協助您追蹤實體工廠、建築物或工業工廠。您可以使用實際資料監控作業、診斷和糾正錯誤以及優化作業。

AWS IoT Core 服務

AWS IoT Core 提供將 IoT 裝置連線至 AWS 雲端的服務，讓其他雲端服務和應用程式可以與您的網際網路連線裝置互動。



下一節說明圖例中顯示的每個 AWS IoT Core 服務。

AWS IoT Core 訊息服務

AWS IoT Core 連線服務提供與 IoT 裝置的安全通訊，並管理在 和 之間傳遞的訊息 AWS IoT。

裝置閘道

讓裝置以安全有效的方式與 AWS IoT通訊。裝置通訊由使用 X.509 憑證的安全通訊協定保護。

訊息代理程式

為裝置和 AWS IoT 應用程式提供安全機制，以互相發佈和接收訊息。您可以直接使用 MQTT 通訊協定或是經 WebSocket 的 MQTT 來發佈和訂閱。如需 AWS IoT 支援的通訊協定的詳細資訊，請參閱 the section called “裝置通訊協定”。裝置和用戶端也可使用 HTTP REST 介面，將資料發佈至訊息代理程式。

訊息中介裝置會將裝置資料分發至已訂閱的裝置，以及其他 AWS IoT Core 服務，例如 Device Shadow 服務和規則引擎。

AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN 可讓您將 LoRaWAN 裝置和閘道連接至 來設定私有 LoRaWAN 網路，而不需要開發和操作 LoRaWAN 網路伺服器 (LNS)。 LoRaWAN AWS 從 LoRaWAN 裝置收到的訊息會傳送至規則引擎，在此引擎中可格式化並傳送至其他 AWS IoT 服務。

規則引擎

規則引擎將訊息代理程式的資料連接至其他 AWS IoT 服務，以進行儲存和額外處理。例如，您可以插入、更新或查詢 DynamoDB 資料表，或根據您在規則引擎中定義的運算式呼叫 Lambda 函數。您可使用 SQL 型語言從訊息承載選取資料，然後處理資料並傳送至其他服務，例如 Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB 和 AWS Lambda。您還可建立將訊息重新發佈至訊息代理程式及其他訂閱者的規則。如需詳細資訊，請參閱的規則 AWS IoT。

AWS IoT Core 控制服務

AWS IoT Core 控制服務提供裝置安全性、管理和註冊功能。

自訂身分驗證服務

您可以定義自訂授權方，使用自訂身分驗證服務和 Lambda 函數，藉以管理自己的身分驗證和授權策略。自訂授權方允許 使用承載字符身分驗證和授權策略 AWS IoT 來驗證您的裝置和授權操作。

自訂授權方可以實作各種身分驗證策略；例如 JSON Web 權杖驗證或 OAuth 提供者標註。它們必須傳回裝置閘道用來授權 MQTT 操作的政策文件。如需詳細資訊，請參閱自訂身分驗證和授權。

裝置佈建服務

可讓您使用描述裝置所需資源 (物件物件、憑證、一或多項政策) 的範本，以佈建裝置。物件物件是登錄檔中的項目，內含描述裝置的屬性。裝置使用憑證進行身分驗證 AWS IoT。政策決定裝置在 AWS IoT可執行的操作。

範本包含了被字典 (對應) 之值取代的變數。只要在字典中為範本的變數傳入不同的值，您就可以使用相同的範本來佈建多個裝置。如需詳細資訊，請參閱裝置佈建。

群組登錄檔

您可利用群組，將多個裝置分類至群組來同時管理。群組也可包含群組：您可建立群組階層。您對父群組執行的任何動作都會套用至其子群組。同樣的動作也適用於父群組中的所有裝置，以及子

群組中的所有裝置。授予群組的許可也會套用至該群組及其所有子群組中的所有裝置。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。

任務服務

可讓您定義一組遠端操作，這組操作會傳送到連接至 AWS IoT的一個或多個裝置，並在其上執行。例如，您可以定義一個任務，指示一組裝置下載並安裝應用程式或韌體更新、重新啟動、輪換憑證，或者執行遠端故障排除操作。

欲建立任務，您必須指定要執行的遠端操作之描述，以及應執行操作的目標清單。目標可以是個別裝置、群組或兩者。如需詳細資訊，請參閱AWS IoT 任務。

登錄檔

整理 AWS 雲端中與每項裝置相關聯的資源。您可登錄您的裝置，且每項裝置最多可與三個自訂屬性相關聯。您也可以建立憑證和 MQTT 用戶端 ID 與每項裝置的關聯，提升您管理裝置及排解其問題的能力。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。

安全與身分服務

在 AWS 雲端中提供共同的安全責任。您的裝置必須保護其憑證，才可以將資料安全地傳送至訊息代理程式。訊息代理程式和規則引擎使用 AWS 安全功能將資料安全傳送至裝置或其他 AWS 服務。如需詳細資訊，請參閱身分驗證。

AWS IoT Core 資料服務

AWS IoT Core 資料服務可協助您的 IoT 解決方案提供可靠的應用程式體驗，即使裝置並非一律連線也一樣。

裝置影子

用於存放及擷取裝置目前狀態資訊的 JSON 文件。

Device Shadow 服務

Device Shadow 服務會維護裝置的狀態，無論裝置是否在線上，都能讓應用程式與裝置通訊。裝置離線時，Device Shadow 服務將為連線應用程式管理該裝置的資料。當裝置重新連線時，它會將自己的狀態與 Device Shadow 服務中的影子同步。您的裝置還可將其目前狀態發佈至影子，以供應用程式或其他可能並非隨時隨地連線的裝置使用。如需詳細資訊，請參閱AWS IoT Device Shadow 服務。

AWS IoT Core 支援服務

適用於 的 Amazon Sidewalk 整合 AWS IoT Core

Amazon Sidewalk 是個共享網路，可改善連線選項，協助裝置更好地共同運作。Amazon Sidewalk 支援範圍廣泛的客戶裝置，例如可找到寵物或貴重物品的裝置、提供智慧家庭安全和照明控制的裝置，及為設備和工具提供遠端診斷的裝置。適用於 的 Amazon Sidewalk 整合 AWS IoT Core 可讓裝置製造商將其 Sidewalk 裝置機群新增至 AWS IoT 雲端。

如需詳細資訊，請參閱AWS IoT Core for Amazon Sidewalk。

進一步了解 AWS IoT

本主題可協助您熟悉 的世界 AWS IoT。您可以取得 IoT解決方案如何套用到各種使用案例、訓練資源、 AWS IoT 社交媒體與所有其他 AWS 服務的連結，以及 AWS IoT 使用之服務和通訊協定清單的一般資訊。

的訓練資源 AWS IoT

我們提供這些培訓課程，協助您了解 AWS IoT 以及如何將這些課程套用至您的解決方案設計。

- 簡介 AWS IoT

  AWS IoT 及其核心服務的影片概觀。
- 深入探索 AWS IoT 身分驗證和授權

  探索 AWS IoT 身分驗證和授權概念的進階課程。您將了解如何驗證和授權用戶端存取 AWS IoT 控制平面和資料平面 APIs。
- 物聯網基礎系列

  IoT 線上學習模組的學習路徑，討論不同的 IoT 技術和功能。

AWS IoT 資源和指南

這些是 特定方面的深入技術資源 AWS IoT。

- IoT Lens – AWS IoT Well-Architected Framework

  描述建構 IoT 應用程式之最佳實務的文件 AWS。

- 設計 的 MQTT 主題 AWS IoT Core

  白皮書說明在 中設計 MQTT 主題 AWS IoT Core 和利用 MQTT AWS IoT Core 功能的最佳實務。

- 摘要和介紹

  PDF 文件，描述 AWS IoT 提供的不同方式來佈建大型裝置機群。

- AWS IoT Core Device Advisor

  AWS IoT Core Device Advisor 提供預先建置的測試，可讓您在將裝置部署到生產環境之前 AWS IoT Core，用來驗證 IoT 裝置是否有可靠且安全的連線最佳實務。

- AWS IoT 資源

  IoT 特定資源，例如技術指南、參考架構、電子書和彙整過的部落格文章，以可搜尋索引呈現。

- 物聯網圖集

  如何解決常見的 IoT 設計問題概述。IoT 圖集提供您在開發 IoT 解決方案時可能遇到的設計挑戰的深入探討。

- AWS 白皮書與指南

  我們目前針對 和其他 AWS 技術收集的白皮書 AWS IoT 和指南。

AWS IoT 在社交媒體中

這些社交媒體管道提供有關 AWS IoT 和 AWS相關主題的資訊。

- 物聯網 AWS IoT – 官方部落格
- AWS IoT YouTube 上 Amazon Web Services 頻道中的影片

這些社交媒體帳戶涵蓋所有 AWS 服務，包括 AWS IoT

- 在 YouTube 上的 Amazon Web Services 頻道
- 推特上的 Amazon Web Services
- 臉書上的 Amazon Web Services
- Instagram 上的 Amazon Web Services
- LinkedIn 上的 Amazon Web Services

AWS AWS IoT Core 規則引擎使用的 服務

AWS IoT Core 規則引擎可以連線至 AWS 這些服務。

- Amazon DynamoDB

  Amazon DynamoDB 是可擴展的 NoSQL 資料庫服務，提供快速且可預測的資料庫效能。

- Amazon Kinesis

  Amazon Kinesis 可輕鬆收集、處理和分析即時串流資料，讓您及時取得洞見並快速回應新資訊。Amazon Kinesis 可擷取即時資料，例如影片、音訊、應用程式記錄、網站點擊流和 IoT 遙測資料，用於機器學習、分析及其他應用。

- AWS Lambda

  AWS Lambda 可讓您執行程式碼，而無需佈建或管理伺服器。您可以設定程式碼以自動觸發 AWS IoT 資料和事件，或直接從 Web 或行動應用程式呼叫。

- Amazon Simple Storage Service

  Amazon Simple Storage Service (Amazon S3) 可以隨時從 Web 上的任意位置存放和擷取任意數量的資料。 AWS IoT 規則可以將資料傳送至 Amazon S3 進行儲存。

- Amazon Simple Notification Service

  Amazon Simple Notification Service (Amazon SNS) 是一種 Web 服務，可讓應用程式、最終使用者和裝置傳送和接收雲端的通知。

- Amazon Simple Queue Service

  Amazon Simple Queue Service (Amazon SQS) 是一種訊息佇列服務，可解偶和擴展微服務、分配式系統和無伺服器應用程式。

- Amazon OpenSearch Service

  Amazon OpenSearch Service 是一個受管服務，可輕鬆部署、操作和擴展 OpenSearch。OpenSearch 是一個熱門的開放原始碼搜尋和分析引擎。

- Amazon SageMaker AI

  Amazon SageMaker AI 可以透過尋找 IoT 資料中的模式來建立機器學習 (ML) 模型。這個服務使用這些模型處理新資料，並為應用程式產生預測結果。

- Amazon CloudWatch

Amazon CloudWatch 提供可靠、可擴展且彈性的監控解決方案，協助設定、管理及擴展您自己的監控系統和基礎設施。

支援的通訊協定 AWS IoT Core

這些主題提供 AWS IoT所使用通訊協定的詳細資訊。如需 所使用的通訊協定 AWS IoT ，以及將裝置和服務連接到其中的詳細資訊 AWS IoT，請參閱 連線至 AWS IoT Core。

- MQTT (訊息佇列遙測傳輸)

  在 MQTT.org 網站的首頁上，您可以找到 MQTT 通訊協定規格。如需 如何 AWS IoT 支援 MQTT 的詳細資訊，請參閱 MQTT。
- HTTPS (超文字傳輸通訊協定：安全)

  裝置和應用程式可以使用 HTTPS 存取 AWS IoT 服務。
- LoRaWAN (長距離廣域網路)

  LoRaWAN 裝置和閘道可以使用 AWS IoT Core for LoRaWAN AWS IoT Core 連線至 。
- TLS (Transport Layer Security) 1.3 版

  TLS v1.3 (RFC 5246) 的規格。 AWS IoT 使用 TLS v1.3 在裝置與 之間建立安全連線。 AWS IoT

AWS IoT 主控台中的新功能

我們正在將 AWS IoT 主控台的使用者介面更新為新的體驗。我們正在分階段更新使用者介面，因此主控台中的某些頁面將有新的體驗，有些可能同時具有原始和新的體驗，有些可能只有原始體驗。

此資料表顯示截至 2022 年 1 月 27 日的 AWS IoT 主控台使用者介面個別區域狀態。

AWS IoT 主控台使用者介面狀態

主控台頁面	原始體驗	全新體驗	說明
監控	不適用	Available	
活動	不適用	Available	
加入：開始使用	不適用	Available	不適用於 CN 區域

主控台頁面	原始體驗	全新體驗	說明
加入：機群佈建範本	Available	Available	
管理：物件	Available	Available	
管理 - 類型	Available	Available	
管理：物件群組	Available	Available	
管理：帳單群組	Available	Available	
管理：任務	Available	Available	
管理：任務範本	不適用	Available	
管理：通道	不適用	Available	
Fleet Hub：開始使用	不適用	Available	不適用於所有 AWS 區域
Fleet Hub：應用程式	不適用	Available	不適用於所有 AWS 區域
Greengrass：開始使用	不適用	Available	不適用於所有 AWS 區域
Greengrass：核心裝置	不適用	Available	不適用於所有 AWS 區域
Greengrass：元件	不適用	Available	不適用於所有 AWS 區域
Greengrass：部署	不適用	Available	不適用於所有 AWS 區域
Greengrass：經典 (V1)	Available	Available	
無線連線：簡介	不適用	Available	不適用於所有 AWS 區域

主控台頁面	原始體驗	全新體驗	說明
無線連線：閘道	不適用	Available	不適用於所有 AWS 區域
無線連線：裝置	不適用	Available	不適用於所有 AWS 區域
無線連線：設定檔	不適用	Available	不適用於所有 AWS 區域
無線連線：目的地	不適用	Available	不適用於所有 AWS 區域
安全：憑證	Available	Available	
安全：政策	Available	Available	
安全：CA	Available	Available	
安全：角色別名	Available	Available	
安全：授權者	Available	Available	
防禦：簡介	不適用	Available	
防禦：稽核	不適用	Available	
防禦：偵測	不適用	Available	
防禦：緩解動作	不適用	Available	
防禦：設定	不適用	Available	
動作：規則	Available	Available	
動作：目的地	Available	Available	
測試：Device Advisor	Available	Available	不適用於所有 AWS 區域

主控台頁面	原始體驗	全新體驗	說明
測試：MQTT 測試用戶端	Available	Available	
軟體	Available	Available	
設定	不適用	Available	
了解	Available	尚無法使用	

圖例

狀態值

- Available

  可使用此使用者介面體驗。
- 不適用

  無法使用此使用者介面體驗。
- 尚無法使用

  正在開發新的使用者介面體驗，但尚未準備好。
- In progress (正在進行)

  新的使用者介面體驗正在更新中。但是，某些網頁可能仍保有原始的使用者體驗。

AWS IoT 搭配 AWS SDK 使用

AWS 軟體開發套件 (SDKs) 適用於許多熱門的程式設計語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
適用於 C++ 的 AWS SDK	適用於 C++ 的 AWS SDK 程式碼範例
AWS CLI	AWS CLI 程式碼範例

SDK 文件	代碼範例
適用於 Go 的 AWS SDK	適用於 Go 的 AWS SDK 程式碼範例
適用於 Java 的 AWS SDK	適用於 Java 的 AWS SDK 程式碼範例
適用於 JavaScript 的 AWS SDK	適用於 JavaScript 的 AWS SDK 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
適用於 .NET 的 AWS SDK	適用於 .NET 的 AWS SDK 程式碼範例
適用於 PHP 的 AWS SDK	適用於 PHP 的 AWS SDK 程式碼範例
AWS Tools for PowerShell	AWS Tools for PowerShell 程式碼範例
AWS 適用於 Python v4 的 SDK	AWS 適用於 Python v4 的 SDK 程式碼範例
適用於 Python (Boto3) 的 AWS SDK	適用於 Python (Boto3) 的 AWS SDK 程式碼範例
適用於 Ruby 的 AWS SDK	適用於 Ruby 的 AWS SDK 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

可用性範例

找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

AWS IoT Core 教學課程入門

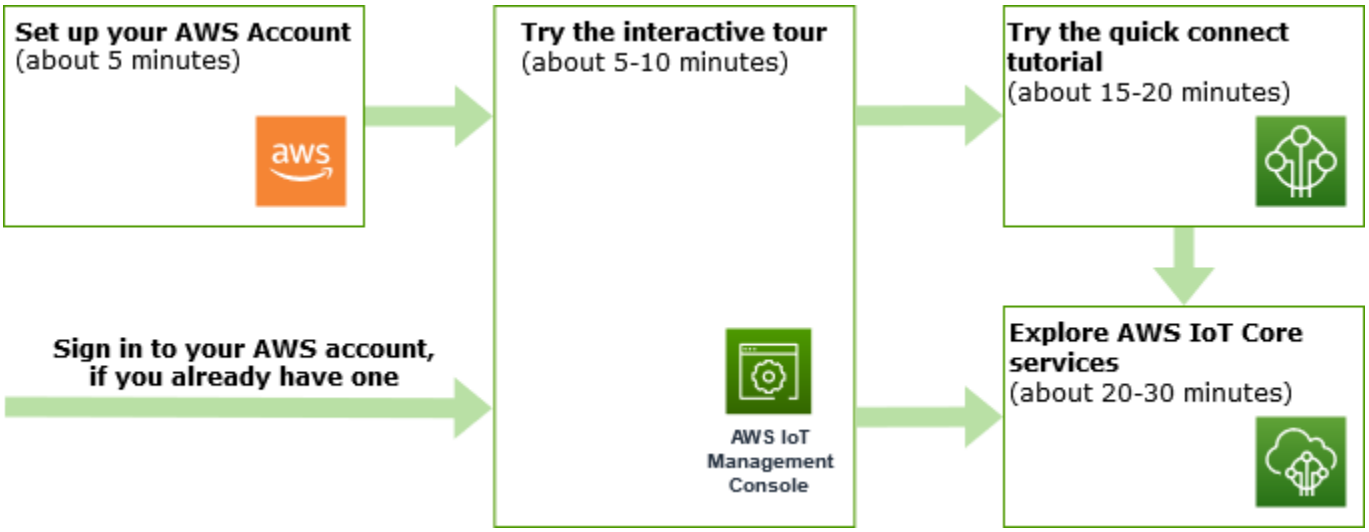
無論您是初次使用 IoT 還是擁有多年經驗，這些資源都會提供可協助您開始使用的 AWS IoT 概念和術語 AWS IoT。

- 在 中查看內部 AWS IoT 及其元件AWS IoT 運作方式。
- 從我們的培訓教材和影片集中進一步了解AWS IoT。本主題也包含 AWS IoT 可以連線的服務清單、社交媒體連結，以及通訊協定規格的連結。
- the section called “將您的第一個裝置連接至 AWS IoT Core”.
- 透過 連線至 AWS IoT Core 並探索 AWS IoT 教學課程 來開發您的 IoT 解決方案。
- 使用 Device Advisor 來測試並驗證您的 IoT 裝置，以進行安全可靠的通訊。
- 使用 AWS IoT Core 管理服務 (例如 機群索引、AWS IoT 任務 和 AWS IoT Device Defender) 來管理您的解決方案。
- 使用 AWS IoT 資料服務 來分析來自您裝置的資料。

將您的第一個裝置連接至 AWS IoT Core

AWS IoT Core 服務會將 IoT 裝置連接到 AWS IoT 服務和其他 AWS 服務。 AWS IoT Core 包含裝置閘道和訊息中介裝置，可連接和處理 IoT 裝置和雲端之間的訊息。

以下是開始使用 AWS IoT Core 和 的方式 AWS IoT。



本節提供 的導覽 AWS IoT Core ，介紹其金鑰服務，並提供數個範例，說明如何將裝置連線至 AWS IoT Core ，並在裝置之間傳遞訊息。在裝置和雲端之間傳遞訊息是每個 IoT 解決方案的基礎，也是您的裝置與其他 AWS 服務互動的方式。

- 設定 AWS 帳戶

  您必須先設定 ，才能使用 AWS IoT 服務 AWS 帳戶。如果您已有 AWS 帳戶 和 IAM 使用者，您可以使用它們並略過此步驟。

- 嘗試快速連接教學課程

  如果您想要快速開始使用 AWS IoT ，並查看它在有限情況下的運作方式，則本教學課程是最佳的。在本教學課程中，您將需要裝置，並在它上安裝一些 AWS IoT 軟體。如果沒有 IoT 裝置，您可以使用 Windows、Linux 或 macOS 個人電腦，作為本教學課程的裝置。如果您想要嘗試 AWS IoT，但沒有裝置，請嘗試下一個選項。

- 嘗試互動式教學課程

  如果您想要查看基本 AWS IoT 解決方案可以做什麼，而無需連接裝置或下載任何軟體，則此示範是最佳的。互動式教學課程提供以 AWS IoT Core 服務為基礎的模擬解決方案，說明其互動方式。

- 使用實作教學探索 AWS IoT Core 服務

  本教學課程最適合想要開始使用 的開發人員 AWS IoT ，讓他們可以繼續探索其他 AWS IoT Core 功能，例如規則引擎和陰影。本教學課程遵循類似於快速連接教學課程的程序，但在每個步驟上提供更多詳細資訊，以便可以更順暢地轉換至其他進階教學課程。

- 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

  了解如何使用 MQTT 測試用戶端，觀看您的第一部裝置將 MQTT 訊息發佈至 AWS IoT。MQTT 測試用戶端是監控和疑難排解裝置連線的實用工具。

> **Note**
>
> 如果想要嘗試其中一個以上的入門教學課程或重複相同的教學課程，您應該先刪除從先前教學課程建立的物件，然後再啟動另一個物件。如果未從先早的教學課程中刪除物件，您需要對後續教學課程使用不同的物件名稱。這是因為物件名稱在您的帳戶和 AWS 區域中必須是唯一的。

如需 的詳細資訊 AWS IoT Core，請參閱什麼是 AWS IoT Core？

設定 AWS 帳戶

AWS IoT Core 第一次使用 之前，請先完成下列任務：

主題

- 註冊 AWS 帳戶
- 開啟 AWS IoT 主控台

註冊 AWS 帳戶

若要開始使用 AWS，您需要 AWS 帳戶。如需建立 的詳細資訊 AWS 帳戶，請參閱《 AWS 帳戶管理參考指南》中的 入門 AWS 帳戶。

- 開啟 AWS IoT 主控台

開啟 AWS IoT 主控台

本節中的大多數主控台導向主題都從 AWS IoT 主控台開始。如果您尚未登入 AWS 帳戶，請登入，然後開啟AWS IoT 主控台並繼續前往下一節以繼續開始使用 AWS IoT。

互動式教學課程

互動式教學課程會顯示 AWS IoT上建置之簡易IoT解決方案的元件。本教學課程示範 IoT 裝置如何與 AWS IoT Core 服務互動。本主題提供 AWS IoT Core 互動式教學課程的預覽。

> **Note**
>
> 主控台中的影像包含未出現在此教學課程中影像的動畫。

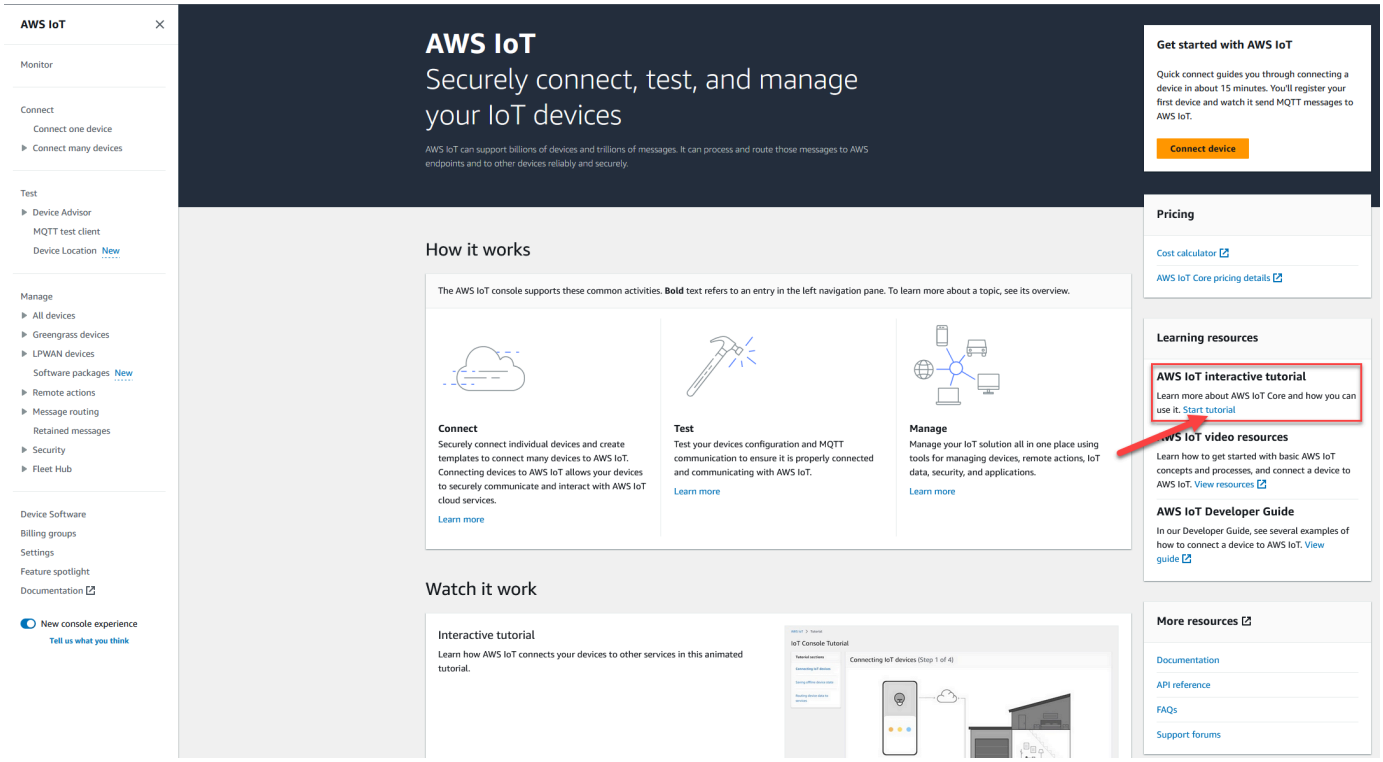
若要執行示範，您必須先 the section called “設定 AWS 帳戶”。不過，本教學課程不需要任何 AWS IoT 資源、其他軟體或任何編碼。

本示範預計大約需時 5-10 分鐘。請預留 10 分鐘，讓自己有充裕的時間來理解每一個步驟。

執行 AWS IoT Core 互動式教學課程

1. 在 AWS IoT 主控台中開啟 AWS IoT 首頁。

在 AWS IoT 首頁上，於學習資源視窗窗格中選擇開始教學課程。



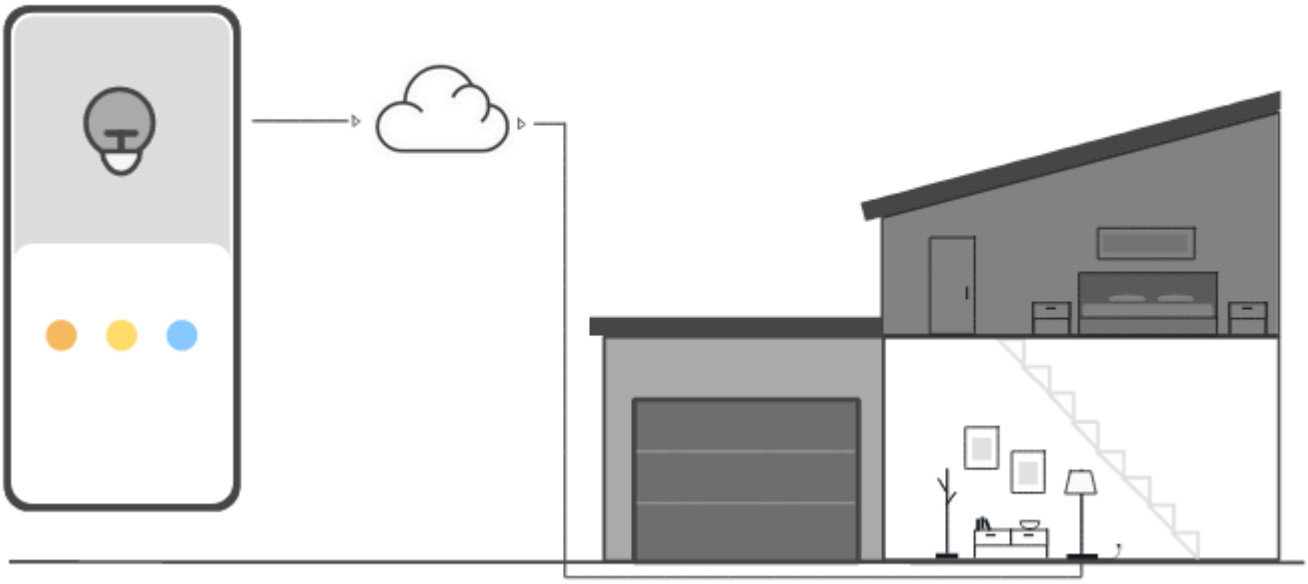
2. 在AWS IoT 主控台教學課程頁面中，檢閱教學課程的各個章節，然後在您準備好繼續進行時，選擇開始章節。

下列各節說明 AWS IoT 主控台教學課程如何呈現這些 AWS IoT Core 功能：

- 連線 IoT 裝置
- 儲存裝置離線狀態
- 將裝置資料路由到服務

連線 IoT 裝置

了解 IoT 裝置如何與 通訊 AWS IoT Core。

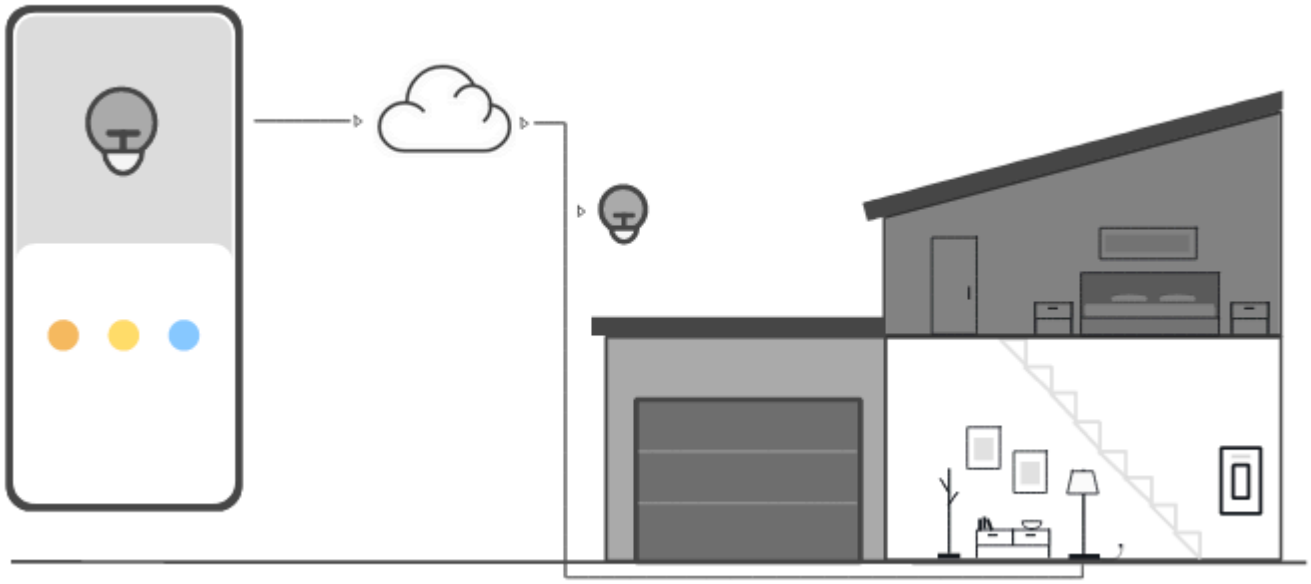


此步驟中的動畫顯示左側的控制器和右側房子中的智慧燈如何 AWS IoT Core 在雲端連接和通訊。動畫顯示裝置與其接收的訊息通訊 AWS IoT Core 和做出反應。

如需將裝置連線至 的詳細資訊 AWS IoT Core，請參閱 連線至 AWS IoT Core。

儲存裝置離線狀態

了解 如何在裝置或應用程式離線時 AWS IoT Core 儲存 的裝置狀態。



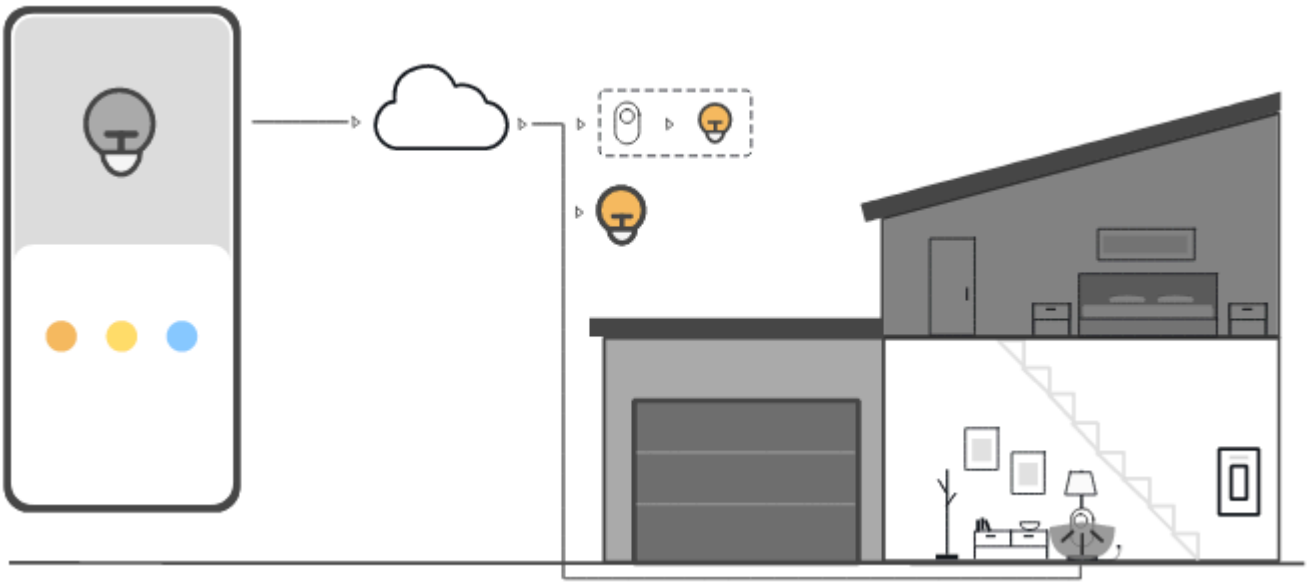
此步驟中的動畫顯示 中的 Device Shadow 服務如何 AWS IoT Core 儲存控制裝置和智慧燈的裝置狀態資訊。智慧燈離線時，裝置影子會儲存來自控制器的命令。

當智慧燈重新連線時 AWS IoT Core，它會擷取這些命令。當控制器離線時，裝置影子會從智慧燈中儲存狀態信息。當控制器重新連接時，它會擷取智慧燈最新的狀態以更新其顯示內容。

如需裝置影子的詳細資訊，請參閱 AWS IoT Device Shadow 服務。

將裝置資料路由到服務

了解如何 AWS IoT Core 將裝置狀態傳送至其他 AWS 服務。



此步驟中的動畫說明如何使用 AWS IoT rule. AWS IoT rules 訂閱來自裝置的特定訊息、解譯這些訊息中的資料，並將解譯的資料路由至其他服務， AWS 以將資料從裝置 AWS IoT Core 傳送至其他服務。在此範例中， AWS IoT 規則會從動作感應器解譯資料，並將命令傳送至 Device Shadow，然後將它們傳送至智慧燈泡。與先前的範例一樣，裝置影子存放控制器的裝置狀態資訊。

如需 AWS IoT 規則的詳細資訊，請參閱 的規則 AWS IoT。

嘗試 AWS IoT Core 快速連線教學課程

在本教學課程中，您將建立第一個物件、將裝置連接至其中，並觀看它傳送 MQTT 訊息。

您可以預期在本教學課程上花費 15-20 分鐘。

本教學課程最適合想要快速開始使用 的人 AWS IoT ，以了解它在有限情況下的運作方式。如果您正在尋找讓您入門的範例，以便可以探索更多功能和服務，請嘗試 在實作教學 AWS IoT Core 中探索。

在本教學課程中，您將在連線至 中物件資源的裝置上下載並執行軟體， AWS IoT Core 做為極小型 IoT 解決方案的一部分。裝置可以是 IoT 裝置 (例如 Raspberry Pi)，也可以是正在執行 Linux、OS 和 OSX，或 Windows 的電腦。如果您想要將長距離 WAN (LoRaWAN) 裝置連接到 AWS IoT，請參閱教學 >將裝置和閘道連接到 AWS IoT Core for LoRaWAN。

如果您的裝置支援可以執行 AWS IoT 主控台的瀏覽器，我們建議您在該裝置上完成本教學課程。

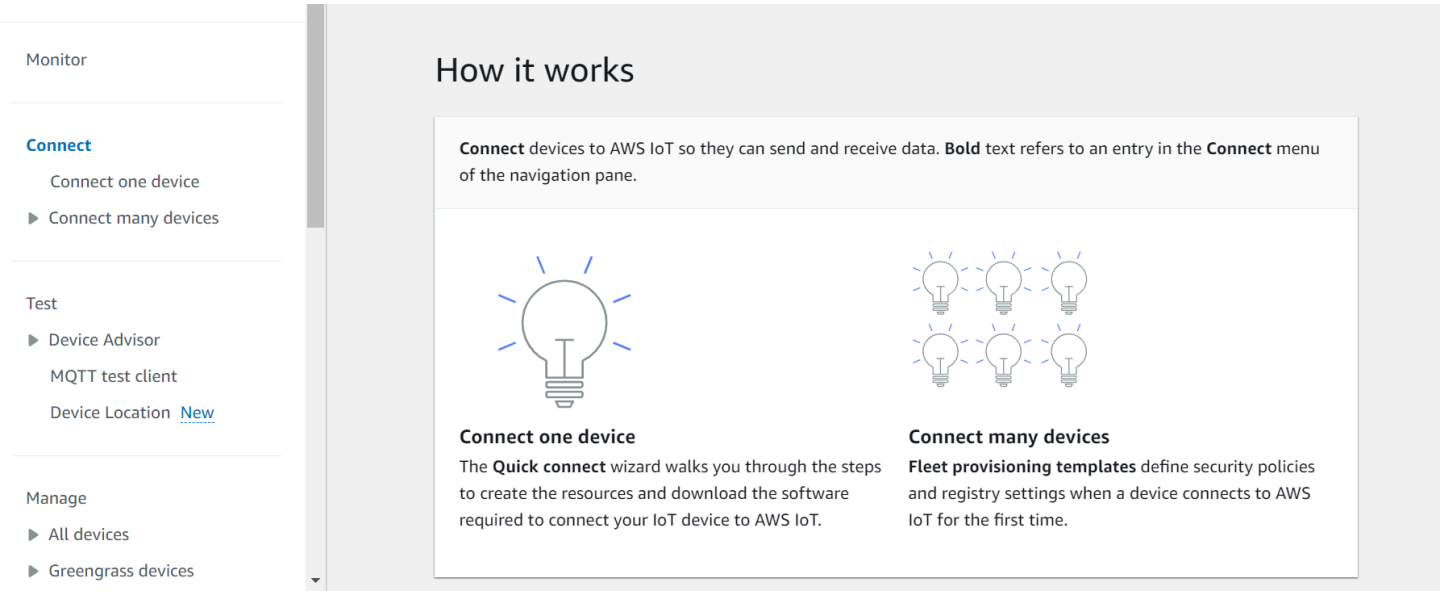
> **Note**
>
> 如果您的裝置沒有相容的瀏覽器，請在電腦上遵循本教學課程。當程序要求您下載檔案時，請將它下載到您的電腦，然後使用安全複製 (SCP) 或類似程序，將下載的檔案傳輸到您的裝置。

本教學課程需要您的 IoT 裝置，才能在 AWS 帳戶裝置的資料端點上與 8443 埠通訊。若要測試其是否可以存取該連接，請嘗試 測試與裝置資料端點的連線 中的操作程序。

步驟 1. 開始教學課程

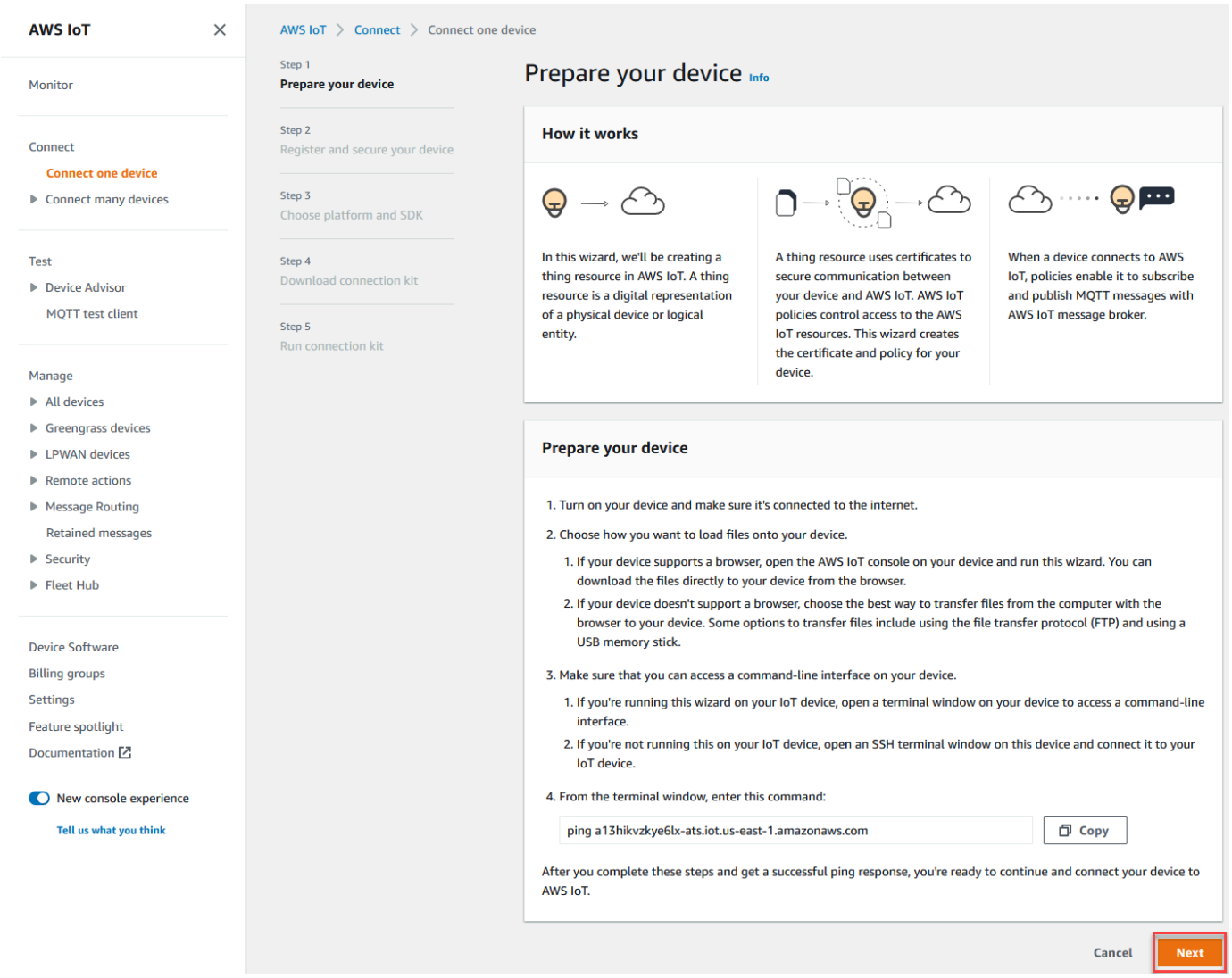
如果可能，請在您的裝置上完成此程序；否則，請準備好稍後在此程序中將檔案傳輸到您的裝置。

若要開始進行教學課程，請登入 AWS IoT 主控台。在 AWS IoT 主控台首頁的左側，選擇連線，然後選擇連線一個裝置。



步驟 2. 建立物件

1. 在 Prepare your device (準備您的裝置) 區段中，依照畫面上的指示，準備您的裝置以連線至 AWS IoT。

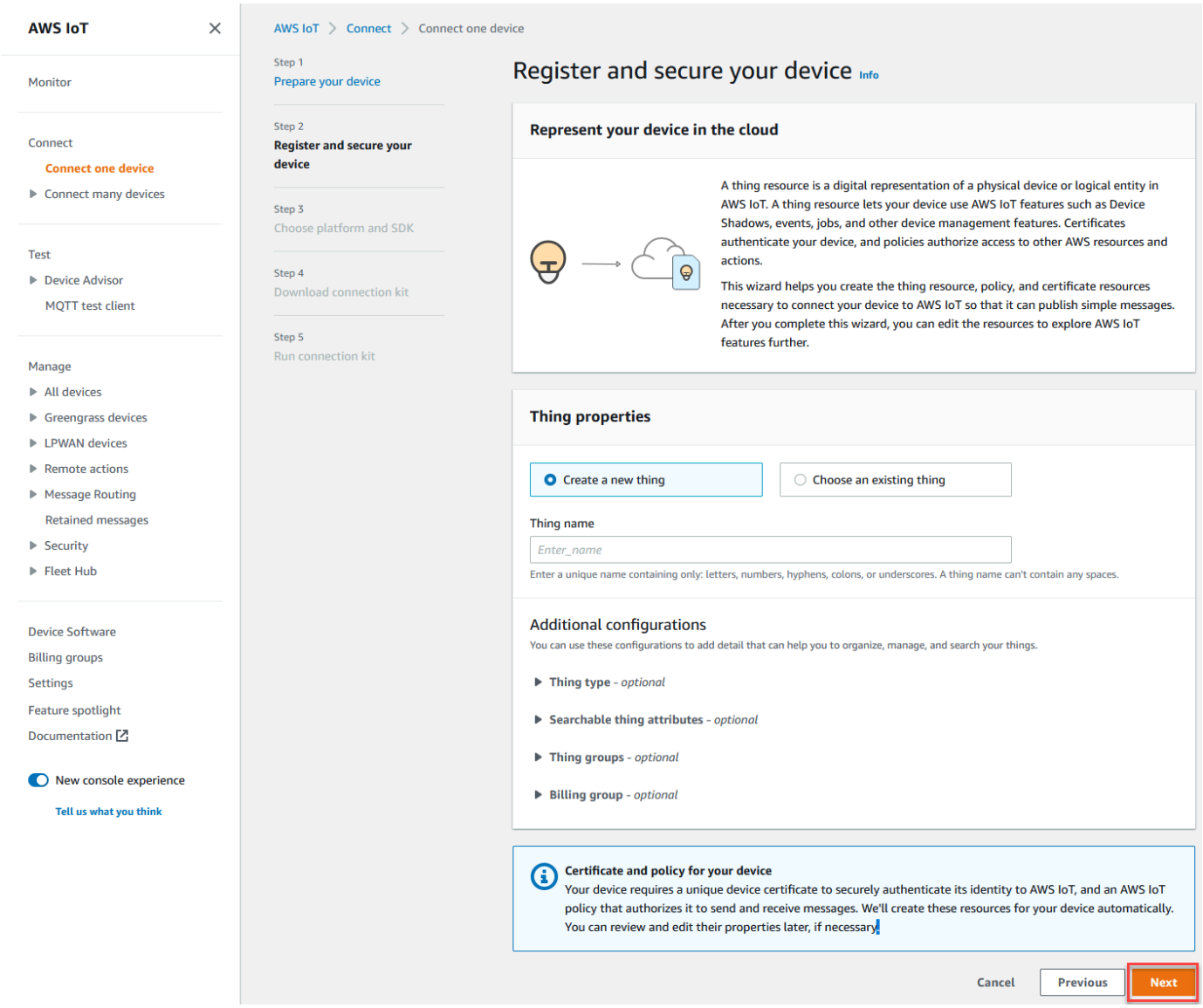


2. 在 Register and secure your device (註冊並保護您的裝置) 區段中，選擇 Create a new thing (建立新的物件) 或者 Choose an existing thing (選擇現有物件)。在 Thing name (物件名稱) 欄位中，為您的物件輸入名稱。在這個範例中使用的物件名稱為 **`TutorialTestThing`**。

> **Important**
>
> 繼續之前，請仔細檢查您的物件名稱。
> 建立物件之後，無法變更物件名稱。如果想要變更物件名稱，您必須建立物件名稱正確的新物件，然後刪除名稱不正確的物件。

在 Additional configurations (其他組態) 區段中，使用列出的選用組態進一步自訂物件資源。

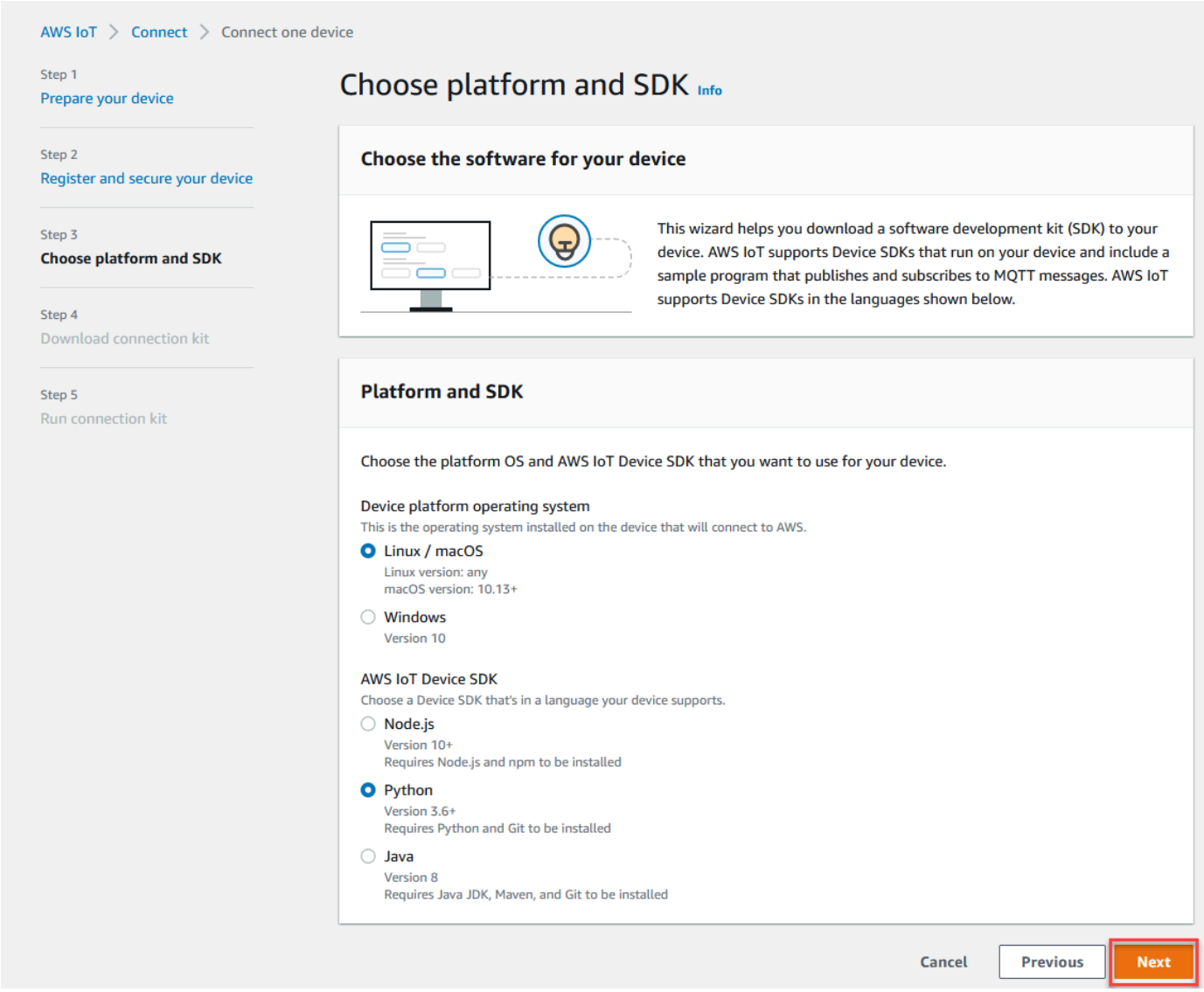
為物件提供名稱並選取任何其他組態後，請選擇 Next (下一步)。



3. 在選擇平台和軟體開發套件區段中，選擇您要使用的平台和 AWS IoT 裝置軟體開發套件語言。這個範例會使用 Linux/OSX 平台和 Python SDK。請確保在目標裝置上安裝了 python3 和 pip3，再繼續下一步。

> **Note**
>
> 請務必在主控台頁面底部檢查所選 SDK 所需的必要軟體清單。
> 您必須在目標電腦上安裝必要的軟體，然後才能繼續下一個步驟。

在您選擇平台和裝置 SDK 語言之後，請選擇 Next (下一步)。

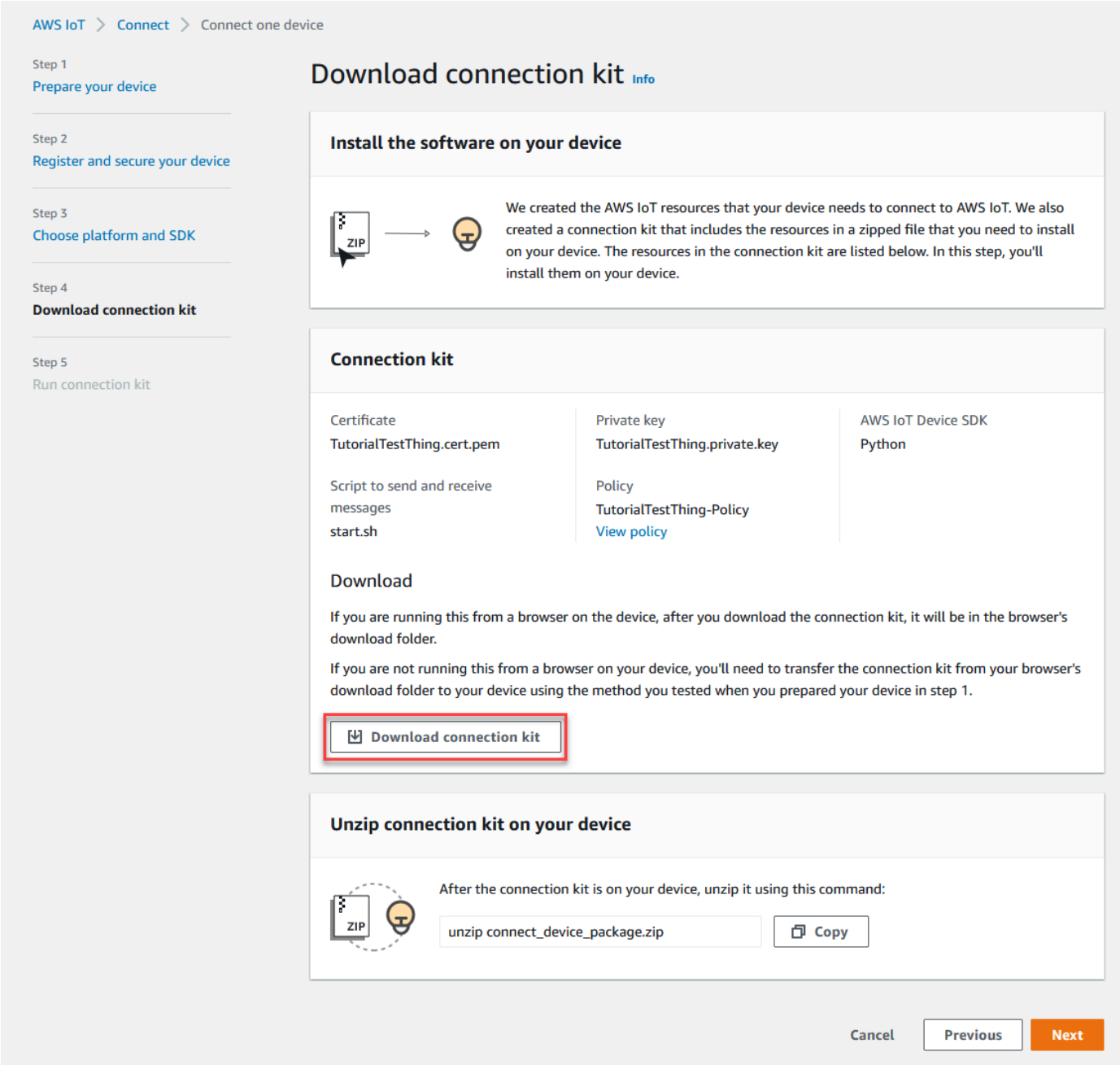


步驟 3。將檔案下載到您的裝置

此頁面會在 AWS IoT 建立連線套件後出現，其中包含您的裝置所需的下列檔案和資源：

- 物件的憑證檔案，用來驗證裝置
- 政策資源，用來授權您的物件可與 AWS IoT互動
- 下載 AWS 裝置 SDK 並在您的裝置上執行範例程式的指令碼

1. 當您準備好繼續時，請選擇 Download connection kit for (下載連線套件) 按鈕，為您先前選擇的平台下載適用的連線套件。



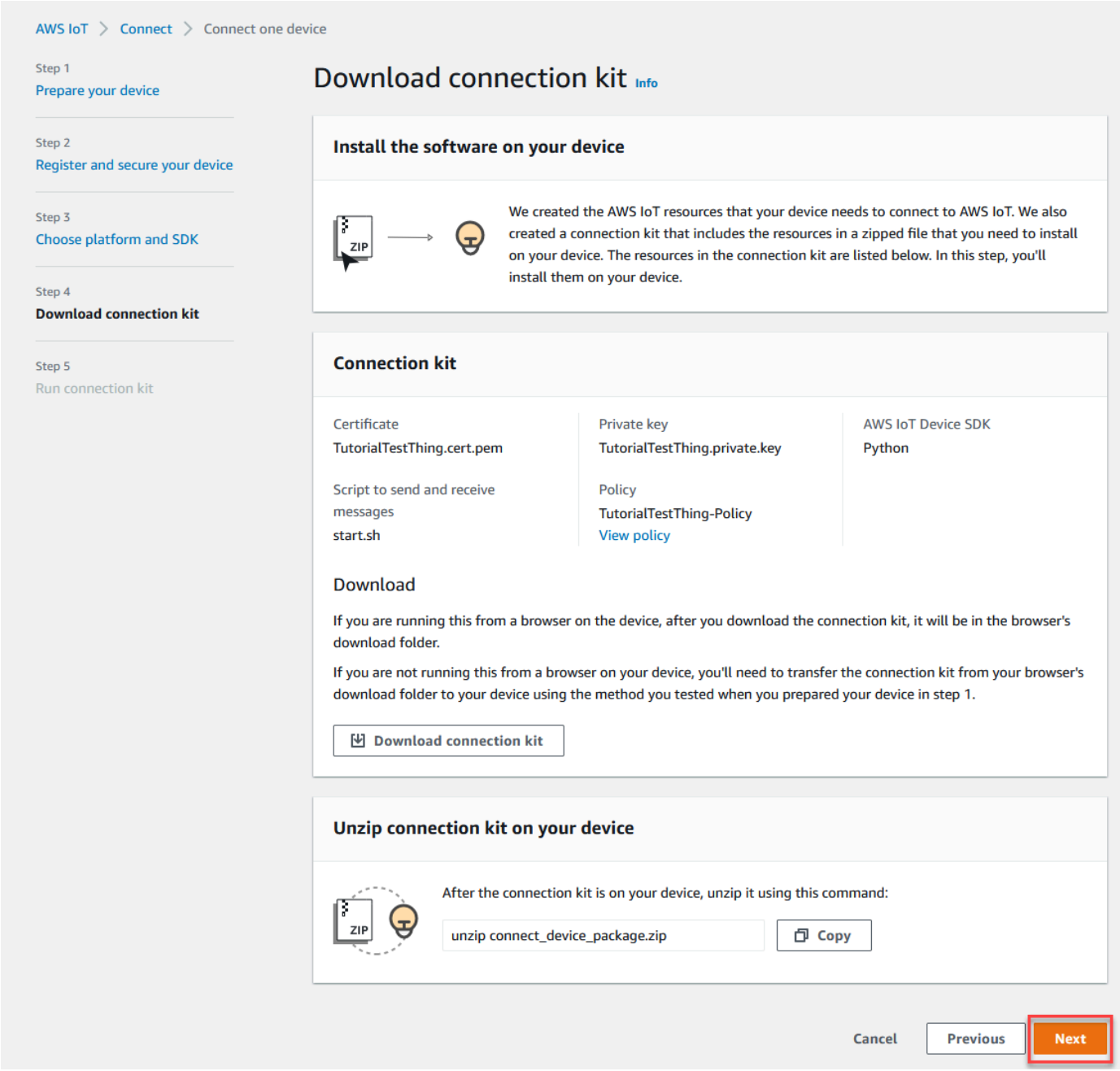
2. 如果您是在裝置上執行此程序，請將連線套件檔案儲存到可以從中執行命令列命令的目錄。

   如果您不是在裝置上執行此程序，請將連線套件檔案儲存至本機目錄，然後將檔案傳輸至您的裝置。

3. 在 Unzip connection kit on your device (解壓縮裝置上的連線套件) 區段中，輸入連線套件檔案所在目錄中的 unzip connect_device_package.zip。

   如果您是使用 Windows PowerShell 視窗，而且 unzip 命令無法運作，請將 unzip 取代為 expand-archive，然後再次嘗試命令列。

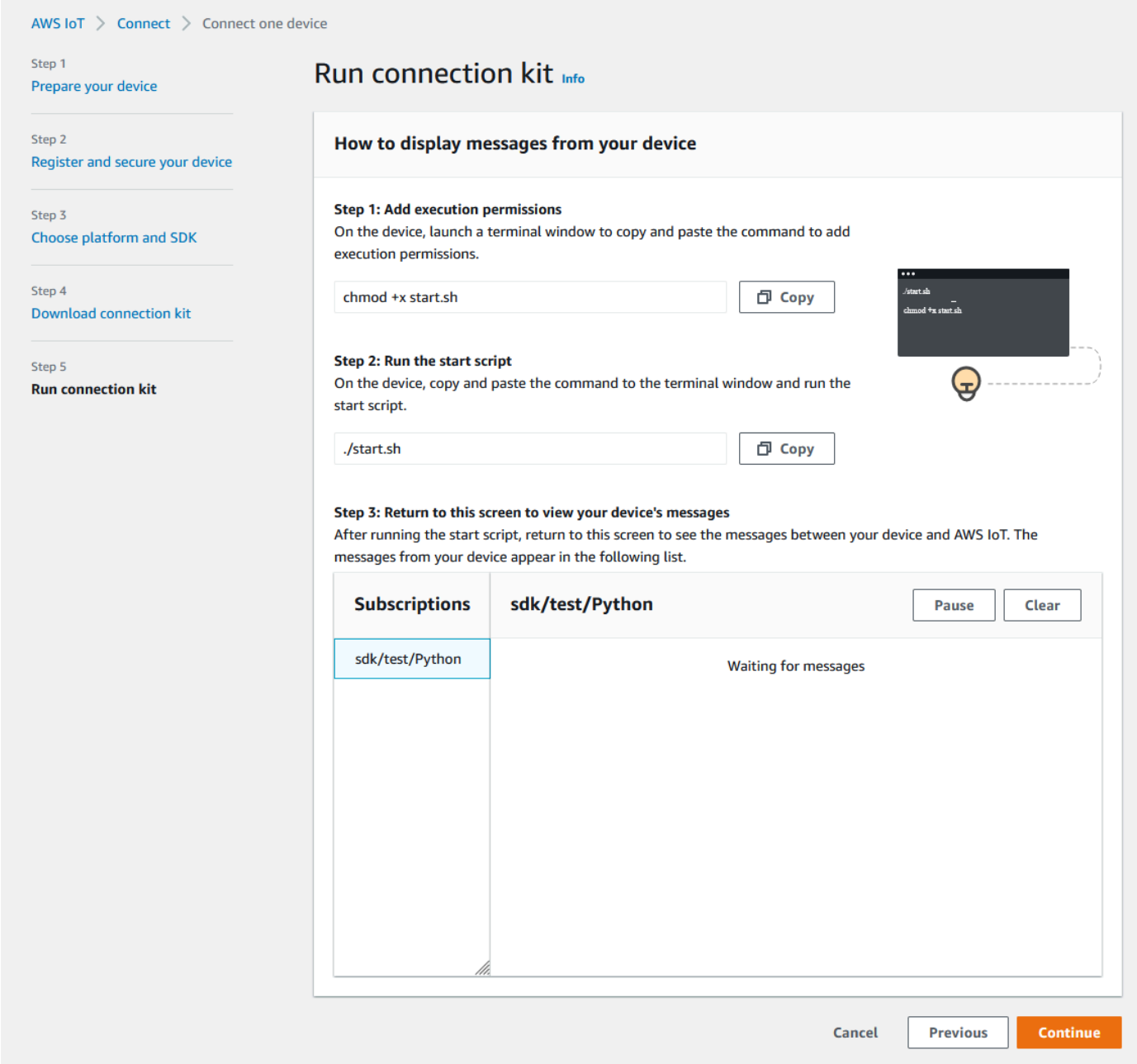
4. 在裝置上具有連線套件檔案之後，請選擇 Next (下一步) 來繼續教學課程。



步驟 4. 執行範例

您可以在裝置上的終端機或命令視窗中執行此程序，同時遵循主控台中顯示的指示。您在主控台中看見的命令是適用於 the section called “步驟 2. 建立物件” 中所選作業系統的命令。這裡顯示的命令適用於 Linux/OSX 作業系統。

1. 在您裝置上的終端機或命令視窗中，在具有連線套件檔案的 目錄中，執行 AWS IoT 主控台中顯示的步驟。



2. 在裝置的終端機或命令視窗中，於您輸入來自主控台中 Step 2 (步驟 2) 的命令之後，您應該會看到如下的輸出。此輸出來自程式傳送到 AWS IoT Core，然後從中收回的訊息。

```
Running pub/sub sample application...
Connecting to a13hikvzkye6lx-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'"Hello World! [1]"'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'"Hello World! [2]"'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'"Hello World! [3]"'
```

當範例程式正在執行時，也會出現測試訊息 `Hello World!`。測試訊息會顯示在終端機或裝置上的命令視窗中。

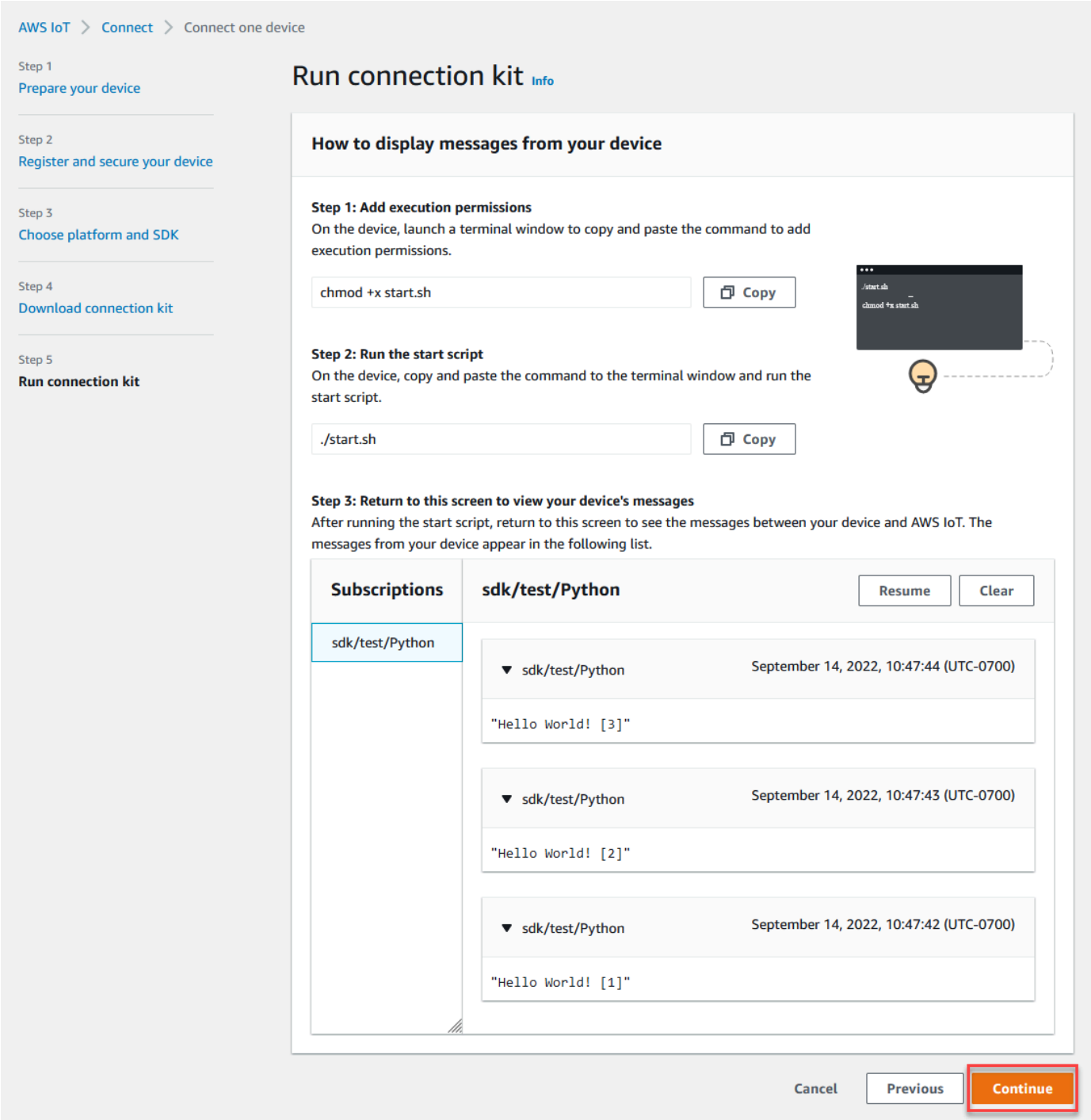
> **Note**
>
> 如需有關主題訂閱和發佈的詳細資訊，請參閱所選 SDK 的範例程式碼。

3. 您可以在此程序的主控台中重複來自 Step 2 (步驟 2) 的命令，以再次執行範例程式。
4. （選用）如果您想要在AWS IoT 主控台中查看來自 IoT 用戶端的訊息，請在 AWS IoT 主控台的測試頁面上開啟 MQTT 測試用戶端。 如果您選擇 Python SDK，則在 MQTT test client (MQTT 測試用戶端) 的 Topic filter (主題篩選條件) 中，輸入主題，例如 **`sdk/test/python`** 以訂閱裝置上的訊息。主題篩選條件區分大小寫，且取決於 Step 1 (步驟 1) 中所選 SDK 的程式設計語言。如需有關主題訂閱和發佈的詳細資訊，請參閱所選 SDK 的程式碼範例。
5. 訂閱測試主題後，請在您的裝置上執行 ./start.sh。如需詳細資訊，請參閱the section called “使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息”。

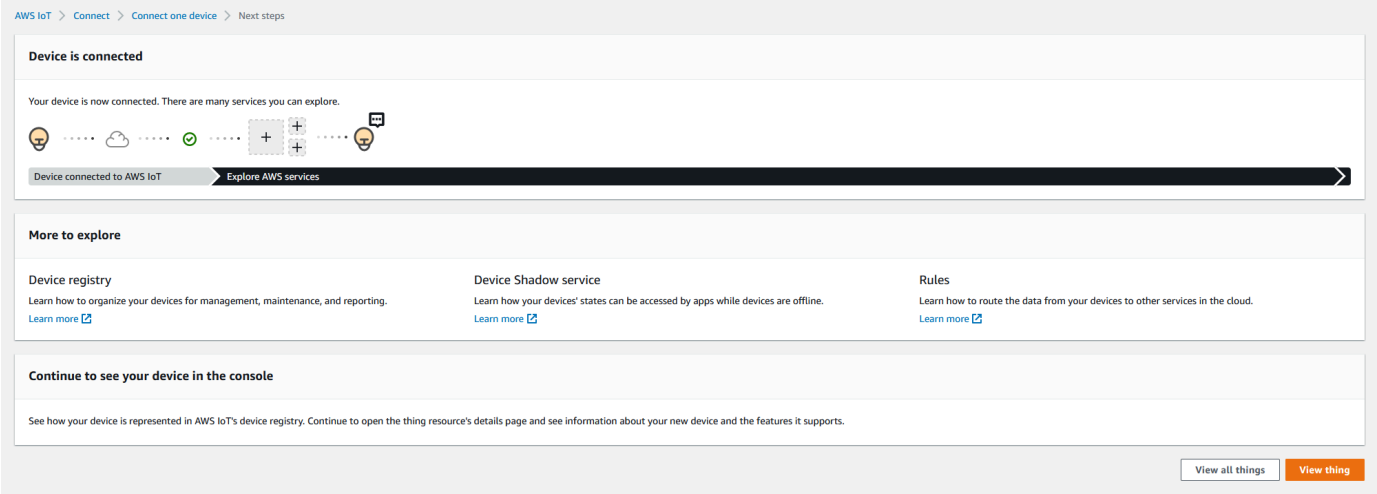
   執行 ./start.sh 之後，訊息會顯示在 MQTT 用戶端中，內容與下列相似：

   ```
   {
     "message": "Hello World!" [1]
   }
   ```

   每次收到新的 `Hello World!` 訊息，以 `[]` 括住的 `sequence` 號碼就會增加 1，並在您結束程式時停止。
6. 若要完成教學課程並查看摘要，請在 AWS IoT 主控台中選擇繼續。



7. 現在會顯示 AWS IoT 快速連線教學課程的摘要。



步驟 5. 深入探索

以下是在您完成快速入門後要 AWS IoT 進一步探索的一些想法。

- 在 MQTT 測試用戶端中檢視 MQTT 訊息

  從 AWS IoT 主控台中，您可以在 AWS IoT 主控台的 Test (測試) 頁面上開啟 MQTT 用戶端。在 MQTT test client (MQTT 測試用戶端) 中，訂閱 #，然後在裝置上執行程式 ./start.sh，如前一個步驟所述。如需詳細資訊，請參閱the section called “使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息”。
- 在您的裝置上，透過 Device Advisor 執行測試

  使用 Device Advisor 測試您的裝置是否可以安全可靠地連線至 ，並與之互動 AWS IoT。
- the section called “互動式教學課程”

  若要開始互動式教學課程，請從 AWS IoT 主控台的學習頁面，在查看運作方式 AWS IoT 圖磚中，選擇開始教學課程。
- 準備好探索其他教學課程

  此快速入門僅提供 的範例 AWS IoT。如果您想要 AWS IoT 進一步探索並了解讓它成為強大 IoT 解決方案平台的功能，請透過 開始準備您的開發平台在實作教學 AWS IoT Core 中探索。

測試與裝置資料端點的連線

本主題會描述如何測試裝置與您帳戶的裝置資料端點連線，也就是您的 IoT 裝置用於連接到 AWS IoT 的端點。

在要測試的裝置上執行這些程序，或使用連結到要測試裝置的 SSH 終端階段作業。

若要測試裝置與您裝置資料端點的連線

- 尋找您的裝置資料端點
- 快速測試連線
- 取得應用程式，以測試與裝置資料端點和通訊埠的連線
- 若要測試與裝置資料端點和通訊埠的連線

尋找您的裝置資料端點

此程序說明如何在 AWS IoT 主控台中尋找您的裝置資料端點，以測試 IoT 裝置的連線。

若要尋找您的裝置資料端點

1. 在 AWS IoT 主控台的 Connect 區段中，前往網域組態。
2. 在網域組態頁面中，前往網域組態容器，然後複製網域名稱。您的端點值對 是唯一的 AWS 帳戶 ，類似於此範例：`a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com`。
3. 儲存您的裝置資料端點，以便在下列程序中使用。

快速測試連線

此程序會測試與裝置資料端點的一般連線，但不測試您的裝置會使用的特定通訊埠。此測試會使用一般程式，通常足以知道您的裝置是否可以連線到 AWS IoT。

如果要測試與裝置將使用的特定通訊埠連線，請跳過此程序並繼續 取得應用程式，以測試與裝置資料端點和通訊埠的連線。

若要快速測試裝置資料端點

1. 在裝置上的終端或命令列視窗中，將範例裝置資料端點 (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) 替換為帳戶的裝置資料端點，然後輸入此命令。

   Linux

   ```
   ping -c 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

Windows

```
ping -n 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
```

2. 如果 ping 顯示類似於以下內容的輸出，則其已成功連結到您的裝置資料端點。雖然它不 AWS IoT 直接與 通訊，但它確實找到了伺服器，並且可能可以透過此端點 AWS IoT 使用。

```
PING a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (xx.xx.xxx.xxx) 56(84) bytes of
 data.
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=1 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=2 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=3 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=4 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=5 ttl=231 time=127 ms
```

如果您對此結果滿意，可以在此處停止測試。

如果要測試與 AWS IoT所使用之特定通訊埠的連線，請繼續進行 取得應用程式，以測試與裝置資料端點和通訊埠的連線。

3. 如果 ping 沒有傳回成功的輸出，請檢查終端值，以確保您具有正確的端點，並檢查裝置與網際網路的連線。

取得應用程式，以測試與裝置資料端點和通訊埠的連線

更徹底的連線測試可以通過使用 nmap 來執行。此程序會進行測試，以檢查 nmap 是否已安裝在您的裝置上。

若要檢查裝置上的 **nmap**

1. 在要測試的裝置上的終端或命令行列視窗中，輸入此命令列，以查看 nmap 是否已安裝。

```
nmap --version
```

2. 如果您看到類似下列的輸出，代表 nmap 已安裝，且您可以繼續 the section called “若要測試與裝置資料端點和通訊埠的連線”。

```
Nmap version 6.40 ( http://nmap.org )
Platform: x86_64-koji-linux-gnu
Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

3. 如果您沒有看到類似上述步驟中顯示的回覆，您必須在裝置上安裝 nmap。選擇適用於裝置作業系統的程序。

Linux

此程序需要您在電腦上安裝軟體的權限。

若要在 Linux 電腦上安裝 nmap

1. 在裝置上的終端或命令列視窗中，輸入與其執行的 Linux 版本相對應的命令。

   a. Debian 或 Ubunt：

   ```
   sudo apt install nmap
   ```

   b. CentOS 或 RHEL：

   ```
   sudo yum install nmap
   ```

2. 使用以下命令測試安裝：

   ```
   nmap --version
   ```

3. 如果您看到類似下列的輸出，代表已安裝 nmap，且您可以繼續 the section called “若要測試與裝置資料端點和通訊埠的連線”。

   ```
   Nmap version 6.40 ( http://nmap.org )
   Platform: x86_64-koji-linux-gnu
   Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
   libdnet-1.12 ipv6
   Compiled without:
   ```

```
Available nsock engines: epoll poll select
```

macOS

此程序需要您在電腦上安裝軟體的權限。

在 macOS 電腦上安裝 nmap

1. 在瀏覽器中，開啟 https://nmap.org/download#macosx，並下載最新穩定的安裝程式。

   出現提示時，請選取 Open with DiskImageInstaller (透過 DiskImageInstaller 開啟)。
2. 在安裝視窗中，將套件移動到 Applications (應用程式) 資料夾。
3. 在 Finder 中尋找 nmap-xxxx-mpkg 套件在 Applications (應用程式) 資料夾中的位置。在套件上 Ctrl-click，並選取 Open (開啟) 以開啟套件。
4. 檢閱安全對話方塊。如果您已準備好安裝 nmap，請選擇 Open (開啟) 以安裝 nmap。
5. 在 Terminal 中，使用此命令測試安裝。

   ```
   nmap --version
   ```
6. 如果您看到類似下列的輸出，代表已安裝 nmap，且您可以繼續 the section called “若要測試與裝置資料端點和通訊埠的連線”。

   ```
   Nmap version 7.92 ( https://nmap.org )
   Platform: x86_64-apple-darwin17.7.0
   Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 libz-1.2.11
    nmap-libpcre-7.6 nmap-libpcap-1.9.1 nmap-libdnet-1.12 ipv6 Compiled without:
   Available nsock engines: kqueue poll select
   ```

Windows

此程序需要您在電腦上安裝軟體的權限。

若要在 Windows 電腦上安裝 nmap

1. 在瀏覽器中，開啟https://nmap.org/download#windows並下載安裝程式的最新穩定版本。

   出現提示時，選擇 Save file (儲存檔案)。下載檔案後，從下載文件夾中將其打開。
2. 安裝檔案下載完成後，打開下載的 nmap-xxxx-setup.exe 以安裝應用程式。

3. 程式安裝時，接受預設設定。

   您不需要 Npcap 應用程式即可進行此測試。如果您不想安裝，可以取消選擇此選項。

4. 在 Command 中，使用此命令測試安裝。

   ```
   nmap --version
   ```

5. 如果您看到類似下列的輸出，代表已安裝 nmap，且您可以繼續 the section called “若要測試與裝置資料端點和通訊埠的連線”。

   ```
   Nmap version 7.92 ( https://nmap.org )
   Platform: i686-pc-windows-windows
   Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 nmap-
   libz-1.2.11 nmap-libpcre-7.6 Npcap-1.50 nmap-libdnet-1.12 ipv6
   Compiled without:
   Available nsock engines: iocp poll select
   ```

若要測試與裝置資料端點和通訊埠的連線

此程序會使用您選取的連接埠，測試 IoT 裝置與裝置資料端點的連線。

若要測試裝置資料端點和通訊埠

1. 在裝置上的終端或命令列視窗中，將範例裝置資料端點 (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) 替換為帳戶的裝置資料端點，然後輸入此命令。

   ```
   nmap -p 8443 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

2. 如果 nmap 會顯示類似下列內容的輸出，nmap 能夠在選定的通訊埠成功連結到您的裝置資料端點。

   ```
   Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-18 16:23 Pacific Standard Time
   Nmap scan report for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
    (xx.xxx.147.160)
   Host is up (0.036s latency).
   Other addresses for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (not scanned):
    xx.xxx.134.144 xx.xxx.55.139 xx.xxx.110.235 xx.xxx.174.233 xx.xxx.74.65
    xx.xxx.122.179 xx.xxx.127.126
   rDNS record for xx.xxx.147.160: ec2-EXAMPLE-160.eu-west-1.compute.amazonaws.com
   ```

```
PORT     STATE SERVICE
8443/tcp open  https-alt
MAC Address: 00:11:22:33:44:55 (Cimsys)

Nmap done: 1 IP address (1 host up) scanned in 0.91 seconds
```

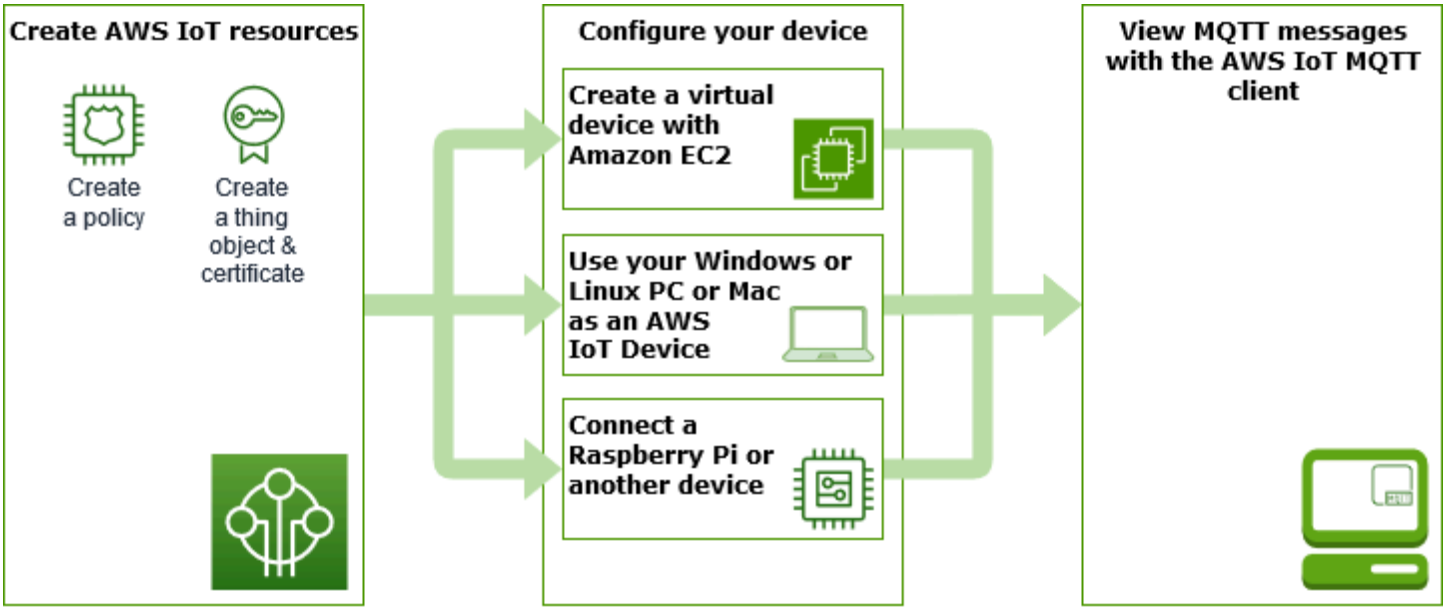
3. 如果 nmap 沒有傳回成功的輸出，請檢查端點值，以確保您的端點正確，並檢查裝置與網際網路的連線。

您可以測試裝置資料端點上的其他通訊埠，例如 443 通訊埠 (主要的 HTTPS 通訊埠)，只要將步驟 1 (**8443**) 中使用的通訊埠替換為您要測試的通訊埠即可。

在實作教學 AWS IoT Core 中探索

在本教學課程中，您將安裝軟體，並建立將裝置連線至 所需的 AWS IoT 資源， AWS IoT Core 讓裝置可以使用 傳送和接收 MQTT 訊息 AWS IoT Core。您會在 AWS IoT 主控台的 MQTT 用戶端中看到訊息。

您可以預期在本教學課程上花費 20-30 分鐘。當您使用 IoT 裝置或 Raspberry Pi 時，例如，如果您需要安裝作業系統並設定裝置，本教學課程可能需要更長的時間。



本教學課程最適合想要開始使用 的開發人員 AWS IoT Core ，讓他們可以繼續探索更進階的功能，例如規則引擎和陰影。本教學課程透過比快速入門教學更詳細地解釋步驟，讓您準備好繼續了解 AWS IoT Core 及其與其他 AWS 服務的互動方式。如果您只是尋找快速的 Hello World 體驗，請嘗試 嘗試 AWS IoT Core 快速連線教學課程。

設定 AWS 帳戶 和 AWS IoT 主控台之後，您將遵循下列步驟，了解如何連接裝置並使其傳送訊息 AWS IoT Core。

後續步驟

- 選擇最適合您的裝置選項
- 如果您不打算使用 Amazon EC2 建立虛擬裝置，請 the section called “建立 AWS IoT 資源”。
- the section called “設定您的裝置”
- the section called “使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息”

如需 的詳細資訊 AWS IoT Core，請參閱什麼是 AWS IoT Core？

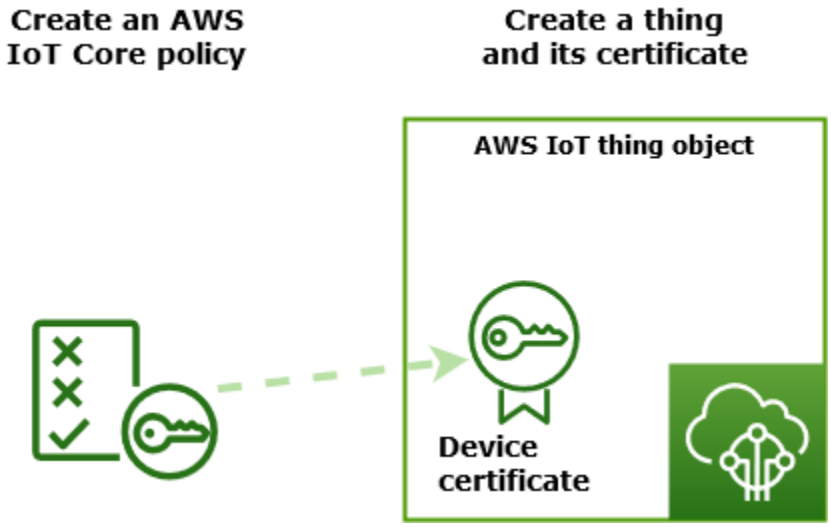
哪種裝置選項最適合您？

如果您不確定要挑選哪個選項，請使用下列各選項的優缺點清單，協助您決定哪個選項最適合您。

選項	這可能是一個好選項，如果：	這可能不是一個好選項，如果：
the section called “使用 Amazon EC2 建立虛擬裝置”	• 您沒有自己的裝置進行測試。 • 您不想要在自己的系統上安裝任何軟體。 • 您想要在 Linux 作業系統上進行測試。	• 您不習慣使用命令列命令。 • 您不想要產生任何額外的 AWS 費用。 • 您不想要在 Linux 作業系統上進行測試。
the section called “使用您的 Windows 或 Linux PC 或 Mac 做為 AWS IoT 裝置”	• 您不想要產生任何額外的 AWS 費用。 • 您不想要設定任何額外裝置。	• 您不想要在個人電腦上安裝任何軟體。 • 您想要一個更具代表性的測試平台。
the section called “連接 Raspberry Pi 或其他裝置”	• 您想要 AWS IoT 使用實際裝置進行測試。 • 您已有測試用的裝置。 • 您具備將硬體整合至系統的經驗。	• 您不想要只為了試用而購買或設定裝置。 • 目前，您想要盡可能簡單 AWS IoT 地測試 。

建立 AWS IoT 資源

在本教學課程中，您將建立裝置連線 AWS IoT Core 和交換訊息所需的 AWS IoT 資源。



1. 建立 AWS IoT 政策文件，其會授權您的裝置與 AWS IoT 服務互動。
2. 在 AWS IoT 及其 X.509 裝置憑證中建立物件，然後連接政策文件。物件是登錄 AWS IoT 檔中裝置的虛擬表示法。憑證會對您的裝置進行身分驗證 AWS IoT Core，而政策文件會授權您的裝置與 互動 AWS IoT。

> **Note**
>
> 若您打算 the section called “使用 Amazon EC2 建立虛擬裝置”，則可跳過本頁，並繼續 the section called “設定您的裝置”。建立您的虛擬物件時，您將建立這些資源。

本教學課程使用 AWS IoT 主控台來建立 AWS IoT 資源。如果您的裝置支援網頁瀏覽器，在裝置的網頁瀏覽器上執行此程序可能會比較容易，因為您可以直接將憑證檔案下載到您的裝置。如果您在其他電腦上執行此程序，則必須先將憑證檔案複製到您的裝置，然後範例應用程式才能使用這些檔案。

建立 AWS IoT 政策

裝置使用 X.509 憑證進行身分驗證。 AWS IoT Core憑證已連接 AWS IoT 政策。這些政策決定了裝置可執行哪些 AWS IoT 操作 (例如訂購或發佈至 MQTT 主題)。您的裝置會在連線並傳送訊息時顯示其憑證 AWS IoT Core。

遵循這些步驟，以建立一個政策，讓您的裝置執行 AWS IoT 執行範例程式所需的作業。您必須先建立 AWS IoT 政策，才能將其附加到之後會建立的裝置憑證中。

建立 AWS IoT 政策

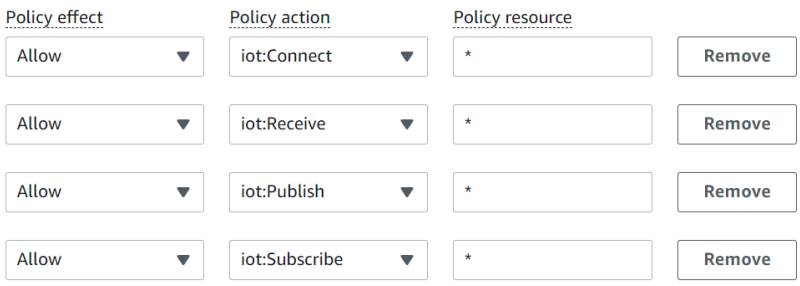
1. 在 AWS IoT 主控台的左側選單中，選擇安全，然後選擇政策。
2. 請在 You don't have a policy yet (尚無任何政策) 頁面上，選擇 Create a policy (建立政策)。

   如果您的帳戶已有現有政策，請選擇建立政策。
3. 在 Create policy (建立政策) 頁面上：

   1. 在 Policy properties (政策內容) 區段的 Policy name (政策名稱) 欄位中，輸入政策名稱 (例如 **My_Iot_Policy**)。請勿在政策名稱中使用個人識別資訊。
   2. 在 Policy document (政策文件) 區段中，建立可授予或拒絕授予 AWS IoT Core 作業資源存取權的政策陳述式。若要建立政策陳述式，授予所有用戶端執行 **iot:Connect**，請執行這些步驟：

      - 在 Policy effect (政策效果) 欄位中，選擇允許。這允許已將此政策附加至其憑證的所有用戶端執行 Policy action (政策動作) 欄位中列出的動作。
      - 在 Policy action (政策動作) 欄位中，選擇政策操作 (例如 **iot:Connect**)。政策動作是您的裝置從裝置 SDK 執行範例程式時，需要許可才能執行的動作。
      - 在 Policy resource (政策資源) 欄位中，輸入資源 Amazon Resource Name (ARN) 或 *****。一個 ***** 以選擇任何用戶端 (裝置)。

      若要建立 **iot:Receive**、**iot:Publish**以及 **iot:Subscribe** 的政策陳述式，請選擇 Add new statement (新增陳述式) 並重複這些步驟。



> **Note**
>
> 在這個快速入門中，萬用字元 (*) 字元是為了簡單起見而使用。如需更高的安全性，您應該指定用戶端 ARN，而非將萬用字元指定為資源，以限制哪些用戶端 (裝置) 可以連接和發佈訊息。用戶端 ARN 遵循以下格式：`arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`*。

不過，您必須先建立資源 (例如用戶端裝置，或物件影子)，然後才能將其 ARN 指配給政策。如需詳細資訊，請參閱 AWS IoT Core 動作資源。

4. 政策相關資訊輸入完畢後，請選擇 Create (建立)。

如需詳細資訊，請參閱AWS IoT 如何使用 IAM。

建立物件

連接到 的裝置 AWS IoT Core 由 AWS IoT 登錄檔中的物件表示。「物件」代表特定的裝置或邏輯實體。它可以是實體裝置或感應器 (例如燈泡或牆上的燈光開關)。它也可以是邏輯實體，例如未連線但與其他裝置 （ 例如具有引擎感應器或控制面板的汽車 ） AWS IoT相關的應用程式或實體實體執行個體。

在 AWS IoT 主控台中建立物件

1. 在 AWS IoT 主控台的左側選單中，選擇所有裝置，然後選擇物件。
2. 在 Things (物件) 頁面上，選擇 Create things (建立物件)。
3. 在 Create things (建立物件) 頁面上，選擇 Create a single thing (建立單一物件)，然後選擇 Next (下一步)。
4. 在 Specify thing properties (指定物件屬性) 頁面上，針對 Thing name (物件名稱)，輸入您的物件名稱，例如 **`MyIotThing`**。

   請小心選擇物件名稱，因為您之後就無法變更物件名稱。

   要變更物件的名稱，您必須建立一個新的物件並為它命名，然後刪除舊的物件。

   > **Note**
   > 請勿在物件名稱中使用個人識別資訊。物件名稱可以出現在未加密的通訊和報告中。

5. 將此頁面上的其餘欄位保持空白。選擇下一步。
6. 在 Configure device certificate - optional (設定裝置憑證 - 選用) 頁面上，選擇 Auto-generate a new certificate (recommended) (自動產生新憑證 (建議動作))。選擇下一步。
7. 在 Attach policies to certificate - optional (將策略連接到憑證 - 選用) 頁面上，選取您在上節所建立的政策。在該節中，政策已命名為 **`My_Iot_Policy`**。選擇 Create thing (建立物件)。
8. 在 Download certificates and keys (下載憑證和金鑰) 頁面上：

   1. 下載每個憑證和金鑰檔案，並儲存它們以供稍後使用。您必須在裝置上安裝這些檔案。

當您儲存憑證檔案時，請給與它們下表中的名稱。這些是稍後範例中使用的檔案名稱。

憑證檔案名稱

檔案	檔案路徑
私有金鑰	`private.pem.key`
公有金鑰	(未在這些範例中使用)
裝置憑證	`device.pem.crt`
根 CA 憑證	`Amazon-root-CA-1.pem`

2. 若要下載這些檔案的根 CA 檔案，請選擇根 CA 憑證檔案的 Download (下載) 連結，此憑證檔案對應於資料端點類型以及您正在使用的密碼套件。在本教學課程中，選擇 RSA 2048 bit key: Amazon Root CA 1 (RSA 2048 位元金鑰：Amazon 根 CA 1) 右側的 Download (下載)，然後下載 RSA 2048 bit key: Amazon Root CA 1 (RSA 2048 位元金鑰：Amazon 根 CA 1) 憑證檔案。

> **Important**
>
> 您必須先儲存憑證檔案，然後才能離開此頁面。在主控台中離開此頁面之後，您再也不能存取憑證檔案。
> 如果忘了下載在此步驟中所建立的憑證檔案，您必須結束此主控台畫面、移至主控台中的物件清單、刪除您建立的物件，然後從頭重新啟動此程序。

3. 選擇 Done (完成)。

完成此程序之後，您應該會在物件清單中看到新的物件。

設定您的裝置

本節描述如何設定您的裝置以連接至 AWS IoT Core。如果您想要開始使用 AWS IoT Core 但還沒有裝置，您可以使用 Amazon EC2 建立虛擬裝置，也可以使用 Windows PC 或 Mac 做為 IoT 裝置。

選取要嘗試的最佳裝置選項 AWS IoT Core。當然，您可以嘗試它們全部，但一次只能嘗試一個。如果您不確定哪個裝置選項最適合您，請閱讀如何選擇哪個是最好的裝置選項，然後返回此頁面。

裝置選項

- 使用 Amazon EC2 建立虛擬裝置
- 使用您的 Windows 或 Linux PC 或 Mac 做為 AWS IoT 裝置
- 連接 Raspberry Pi 或其他裝置

使用 Amazon EC2 建立虛擬裝置

在本教學課程中，您將建立 Amazon EC2 執行個體，充當雲端中的虛擬裝置。

若要完成本教學課程，您需要 AWS 帳戶。若您沒有帳戶，請完成 設定 AWS 帳戶 所述的步驟，然後再繼續。

在本教學課程中，您將會執行下列動作：

- 設定 Amazon EC2 執行個體
- 安裝 Git、Node.js 並設定 AWS CLI
- 為您的虛擬裝置建立 AWS IoT 資源
- 安裝適用於 JavaScript 的 AWS IoT 裝置 SDK
- 執行範例應用程式
- 在 AWS IoT 主控台中檢視來自範例應用程式的訊息

設定 Amazon EC2 執行個體

下列步驟顯示如何建立 Amazon EC2 執行個體，其將充當您的虛擬裝置取代實體裝置。

如果您是第一次建立 Amazon EC2 執行個體，開始使用 Amazon EC2Linux 執行個體中的相關指示可能會更有幫助。

啟動執行個體

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 從左側的主控台功能表展開 Instances (執行個體) 區段，然後選擇 Instances (執行個體)。在 Instances (執行個體) 儀表板中，選擇右側的 Launch instances (啟動執行個體) 以顯示基本組態清單。
3. 在 Name and tags (名稱和標籤) 區段中，輸入執行個體的名稱，並選擇性地新增標籤。

4. 在 Application in OS Images (Amazon Machine Image) (應用程式和作業系統映像) 區段中選擇執行個體的 AMI 範本，例如 Amazon Linux 2 AMI (HVM)。請注意，此 AMI 會標示為 "Free tier eligible." (符合免費方案。)
5. 在 Instance Type (執行個體類型) 頁面中，您可以選取您執行個體的硬體組態。選取 `t2.micro` 類型，其預設為選取。請注意，此執行個體類型符合免費方案資格。
6. 在 Key pair (login) (金鑰對 (登入)) 區段中，從下拉式清單中選擇金鑰對名稱，或選擇 Create a new key pair (建立新金鑰對) 以建立新的金鑰對。建立新金鑰對時，請務必下載私密金鑰檔案並將其儲存在安全的地方，因為這是您下載並儲存的唯一機會。每次您連線至執行個體來啟動執行個體與對應的私有金鑰時，都需要提供您的金鑰對名稱。

   > ⚠ Warning
   >
   > 不要選取 Proceed without a key pair (不使用金鑰對而繼續) 選項。如果不使用金鑰對而啟動執行個體，就無法與它連線。

7. 在 Network settings (網路設定) 區段和 Configure storage (設定儲存區) 區段中，您可以保留預設設定。當您就緒後，選擇 Launch Instances (啟動執行個體)。
8. 會有確認頁面讓您知道您的執行個體正在啟動。選擇 View Instances (檢視執行個體) 關閉確認頁面並返回主控台。
9. 您可以在 Instances (執行個體) 畫面中檢視啟動狀態。啟動執行個體無須費時。當您啟動執行個體時，其初始狀態是 `pending`。在執行個體啟動後，其狀態會變更為 `running`，並得到公有的 DNS 名稱。(如果隱藏 Public DNS (IPv4) (公有 DNS (IPv4)) 欄，請選擇頁面右上角的 Show/Hide Columns (顯示/隱藏欄) (齒輪狀圖示)，然後選取 Public DNS (IPv4) (公有 DNS (IPv4))。)
10. 執行個體可能需要幾分鐘的時間準備就緒讓您連線。確認您的執行個體是否已通過狀態檢查，您可以在 Status Checks (狀態檢查) 欄檢視此資訊。

    在新的執行個體通過其狀態檢查之後，請繼續下一個程序並連接至該執行個體。

連結到您的執行個體

您可以從 Amazon EC2 主控台選取執行個體，並選擇使用 Amazon EC2 Instance Connect 來進行連線，以使用瀏覽器型用戶端連接至執行個體。Instance Connect 會處理許可並提供成功的連線。

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在左側選單中，選擇 Instances (執行個體)。
3. 選取執行個體，然後選取 Connect (連線)。

4. 選擇 Amazon EC2 Instance Connect，Connect (連線)。

您現在應該具有已登入新 Amazon EC2 執行個體的 Amazon EC2 Instance Connect 視窗。

安裝 Git、Node.js 並設定 AWS CLI

在本節中，您會在 Linux 執行個體上安裝 Git 和 Node.js。

安裝 Git

1. 在您的 Amazon EC2 Instance Connect 視窗中，使用下列命令更新您的執行個體。

```
sudo yum update -y
```

2. 在您的 Amazon EC2 Instance Connect 視窗中，使用下列命令安裝 Git。

```
sudo yum install git -y
```

3. 若要檢查 Git 是否已安裝，以及 Git 的目前版本，請執行下列命令：

```
git --version
```

安裝 Node.js

1. 在您的 Amazon EC2 Instance Connect 視窗中，使用下列命令安裝 Node Version Manager (NVM)。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```

nvm 可以安裝多種 Node.js 版本且允許您在各版本間進行切換，因此我們會使用 nvm 安裝 Node.js。

2. 在您的 Amazon EC2 Instance Connect 視窗中，使用此命令啟用 nvm。

```
. ~/.nvm/nvm.sh
```

3. 在您的 Amazon EC2 Instance Connect 視窗中，使用 nvm 來透過使用此命令安裝最新版本的 Node.js。

```
nvm install 16
```

> **Note**
>
> 這會安裝 Node.js 的最新 LTS 版本。

安裝 Node.js 時，系統會一併安裝節點套件管理工具 (npm)，您可以視需要安裝額外的模組。

4. 在您的 Amazon EC2 Instance Connect 視窗中，使用此命令來測試是否已安裝並正確地執行 Node.js。

```
node -e "console.log('Running Node.js ' + process.version)"
```

本教學課程需要 Node v10.0 或更新版本。如需詳細資訊，請參閱 Tutorial: Setting Up Node.js on an Amazon EC2 Instance (教學課程：在 Amazon EC2 執行個體上設定 Node.js)。

設定 AWS CLI

您的 Amazon EC2 執行個體已預先載入 AWS CLI。不過，您必須完成您的 AWS CLI 設定檔。如需如何設定 CLI 的詳細資訊，請參閱設定 AWS CLI。

1. 下列範例顯示範本值。將它們取代為您自己的值。您可以在 AWS 主控台中於您在 My Security Credentials (我的安全憑證) 下的帳戶資訊中找到這些值。

   在您的 Amazon EC2 Instance Connect 視窗中，輸入此命令：

```
aws configure
```

   然後在顯示的提示中輸入來自您帳戶的值。

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

2. 您可以使用此命令測試您的 AWS CLI 組態：

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果您的 AWS CLI 設定正確，命令應該會從 傳回端點地址 AWS 帳戶。

為您的虛擬裝置建立 AWS IoT 資源

本節說明如何使用 AWS CLI 直接在虛擬裝置上建立物件物件及其憑證檔案。這是直接在裝置上完成，以避免從另一台電腦將它們複製到裝置時可能造成的潛在複雜性。在本節中，您將為虛擬裝置建立下列資源：

- 代表虛擬裝置所在的物件 AWS IoT。
- 要驗證虛擬裝置的憑證。
- 政策文件，用來授權虛擬裝置連接至 AWS IoT，以及用來發佈、接收和訂閱訊息。

在 Linux 執行個體中建立 AWS IoT 物件

連接到 的裝置 AWS IoT 由 AWS IoT 登錄檔中的物件表示。「物件」代表特定的裝置或邏輯實體。在此案例中，您的「物件」將代表您的虛擬裝置，即 Amazon EC2 執行個體。

1. 在您的 Amazon EC2 Instance Connect 視窗中，執行下列命令來建立您的物件。

```
aws iot create-thing --thing-name "MyIotThing"
```

2. JSON 回應應該如下所示：

```
{
    "thingArn": "arn:aws:iot:your-region:your-aws-account:thing/MyIotThing",
    "thingName": "MyIotThing",
    "thingId": "6cf922a8-d8ea-4136-f3401EXAMPLE"
}
```

在 Linux 執行個體中建立和連接 AWS IoT 金鑰和憑證

create-keys-and-certificate 命令會建立由 Amazon 根憑證授權機構簽署的用戶端憑證。此憑證用來驗證虛擬裝置的身分。

1. 在您的 Amazon EC2 Instance Connect 視窗中，建立目錄以存放您的憑證和金鑰檔案。

```
mkdir ~/certs
```

2. 在您的 Amazon EC2 Instance Connect 視窗中，使用此命令下載 Amazon 憑證授權機構 (CA) 憑證的副本。

```
curl -o ~/certs/Amazon-root-CA-1.pem \
    https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

3. 在您的 Amazon EC2 Instance Connect 視窗中，執行下列命令來建立您的私有金鑰、公有金鑰，以及 X.509 憑證檔案。此命令也會向 註冊和啟用憑證 AWS IoT。

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "~/certs/device.pem.crt" \
    --public-key-outfile "~/certs/public.pem.key" \
    --private-key-outfile "~/certs/private.pem.key"
```

回傳的結果如下所示。儲存 `certificateArn`，以便您可以在後續命令中使用它。您需要它，將您的憑證連接到您的物件，並在稍後的步驟中將政策連接到憑證。

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp3780D8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
```

```
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC
 KEY-----\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h
\nMMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/
gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB\n-----
END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omitted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

4. 在您的 Amazon EC2 Instance Connect 視窗中，將您的物件連接到您剛建立的憑證，方法是使用下列命令，以及來自前一個命令之回應中的 *certificateArn*。

```
aws iot attach-thing-principal \
    --thing-name "MyIotThing" \
    --principal "certificateArn"
```

如果成功，此命令不會顯示任何輸出。

建立並連接政策

1. 在您的 Amazon EC2 Instance Connect 視窗中，複製此策略文件並將其貼到名為 **~/policy.json** 的檔案來建立政策檔案。

如果您沒有最愛的 Linux 編輯器，則可以使用此命令開啟 nano。

```
nano ~/policy.json
```

將 policy.json 的政策文件貼入其中。輸入 ctrl-x 以結束 nano 編輯器並儲存檔案。

policy.json 的政策文件內容

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

2. 在您的 Amazon EC2 Instance Connect 視窗中，使用下列命令建立您的政策。

```
aws iot create-policy \
    --policy-name "MyIotThingPolicy" \
    --policy-document "file://~/policy.json"
```

輸出：

```
{
    "policyName": "MyIotThingPolicy",
    "policyArn": "arn:aws:iot:us-east-1:your-aws-account:policy/
MyIotThingPolicy",
    "policyDocument": "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [
            {
                \"Effect\": \"Allow\",
                \"Action\": [
                    \"iot:Publish\",
                    \"iot:Receive\",
                    \"iot:Subscribe\",
                    \"iot:Connect\"
                ],
                \"Resource\": [
                    \"*\"
                ]
        }
```

```
        ]
    }",
    "policyVersionId": "1"
}
```

3. 在您的 Amazon EC2 Instance Connect 視窗中，使用下列命令將政策連接至虛擬裝置的憑證。

```
aws iot attach-policy \
    --policy-name "MyIotThingPolicy" \
    --target "certificateArn"
```

如果成功，此命令不會顯示任何輸出。

安裝適用於 JavaScript 的 AWS IoT 裝置 SDK

在本節中，您將安裝適用於 JavaScript 的 AWS IoT 裝置開發套件，其中包含應用程式可用來與 AWS IoT 和範例程式通訊的程式碼。如需詳細資訊，請參閱適用於 JavaScript GitHub 儲存庫的AWS IoT SDK。

在 Linux 執行個體上安裝適用於 JavaScript 的 AWS IoT 裝置開發套件

1. 在 Amazon EC2 Instance Connect 視窗中，使用此命令將適用於 JavaScript 的 AWS IoT 裝置 SDK 儲存庫複製到主目錄的`aws-iot-device-sdk-js-v2`目錄中。

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

2. 導覽至您在上述步驟中建立的 `aws-iot-device-sdk-js-v2` 目錄。

```
cd aws-iot-device-sdk-js-v2
```

3. 使用 npm 來安裝 SDK。

```
npm install
```

執行範例應用程式

以下各節命令假設您的金鑰和憑證檔案存放在虛擬裝置上，如下表所示。

憑證檔案名稱

檔案	檔案路徑
私有金鑰	~/certs/private.pem.key
裝置憑證	~/certs/device.pem.crt
根 CA 憑證	~/certs/Amazon-root-CA-1.pem

在本節中，您將安裝並執行可在適用於 JavaScript 的 AWS IoT 裝置 SDK aws-iot-device-sdk-js-v2/samples/node目錄中找到pub-sub.js的範例應用程式。此應用程式顯示裝置 (您的 Amazon EC2 執行個體) 如何使用 MQTT 程式庫來發佈和訂閱 MQTT 訊息。pub-sub.js 範例應用程式會訂閱主題 topic_1、將 10 則訊息發佈至該主題，以及在從訊息代理程式接收訊息時顯示這些訊息。

安裝並執行範例應用程式

1. 在您的 Amazon EC2 Instance Connect 視窗中，導覽至 SDK 已建立的 aws-iot-device-sdk-js-v2/samples/node/pub_sub 目錄，並使用這些命令安裝範例應用程式。

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. 在您的 Amazon EC2 Instance Connect 視窗中，使用此命令 AWS IoT 從 取得*your-iot-endpoint*。

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

3. 在您的 Amazon EC2 Instance Connect 視窗中，依指示插入 *your-iot-endpoint* 並執行此命令。

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

範例應用程式：

1. 連接至 AWS IoT Core 您帳戶的 。

2. 訂閱訊息主題 topic_1，並顯示其收到關於該主題的訊息。
3. 將 10 則訊息發佈至主題 topic_1。
4. 顯示類似下列內容的輸出：

```
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":1}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":2}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":3}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":4}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":5}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":6}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":7}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":8}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":9}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":10}
```

若在執行範例應用程式時發生問題，請檢閱 the section called “對範例應用程式的問題進行故障診斷”。

您也可以將 `--verbosity debug` 參數新增至命令列，以便範例應用程式顯示有關其正在做什麼的詳細訊息。這些資訊可能會提供您更正問題所需的協助。

在 AWS IoT 主控台中檢視來自範例應用程式的訊息

您可以在範例應用程式的訊息透過訊息代理程式傳遞時看到這些訊息，方法為使用 AWS IoT 主控台中的 MQTT test client (MQTT 測試用戶端)。

檢視範例應用程式所發佈的 MQTT 訊息

1. 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息。這有助於您了解如何使用 AWS IoT 主控台中的 MQTT 測試用戶端，在 MQTT 訊息通過訊息代理程式時，檢視訊息。
2. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。

3. 在Subscribe to a topic (訂閱主題)中，訂閱主題，topic_1。
4. 在您的 Amazon EC2 Instance Connect 視窗中，再次執行範例應用程式，並在 AWS IoT 主控台的 MQTT test client (MQTT 測試用戶端) 中觀看訊息。

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

如需 MQTT 和 如何 AWS IoT Core 支援通訊協定的詳細資訊，請參閱 MQTT。

使用您的 Windows 或 Linux PC 或 Mac 做為 AWS IoT 裝置

在本教學課程中，您將設定個人電腦以搭配 使用 AWS IoT。這些指示支援 Windows 和 Linux PC 以及 Mac。若要完成此目的，您需要在電腦上安裝一些軟體。如果您不想要在電腦上安裝軟體，則可以嘗試使用 Amazon EC2 建立虛擬裝置，它會在虛擬機器上安裝所有軟體。

在本教學課程中，您將會執行下列動作：

- 設定您的個人電腦
- 安裝 Git、Python 和適用於 Python AWS IoT 的裝置 SDK
- 設定政策和執行範例應用程式
- 在 AWS IoT 主控台中檢視來自範例應用程式的訊息
- 在 Python 中執行共享訂閱範例

設定您的個人電腦

若要完成本教學課程，您需要一部 Windows 或 Linux PC，或是一部可連接到網際網路的 Mac。

繼續下一個步驟之前，請確定您可以在電腦上開啟命令列視窗。在 Windows PC 上使用 cmd.exe。在 Linux PC 或 Mac 上，使用 Terminal。

安裝 Git、Python 和適用於 Python AWS IoT 的裝置 SDK

在本節中，您將在電腦上安裝 Python 和適用於 Python 的 AWS IoT 裝置 SDK。

安裝最新版本的 Git 和 Python

此程序說明如何在個人電腦上安裝最新版本的 Git 和 Python。

在您的電腦上下載並安裝 Git 和 Python

1. 查看您的電腦上是否已安裝 Git。在命令列中輸入此命令。

   ```
   git --version
   ```

   如果命令顯示 Git 版本，表示已安裝 Git，而且您可以繼續進行下一個步驟。

   如果命令顯示錯誤，請開啟 https://git-scm.com/download 並為您的電腦安裝 Git。
2. 查看您是否已安裝 Python。在命令列中輸入此命令。

   ```
   python -V
   ```

   > **Note**
   >
   > 如果此命令提供錯誤：`Python was not found`，原因可能是您的作業系統呼叫 Python v3.x 可執行檔作為 `Python3`。在此情況下，將 `python` 的所有執行個體取代為 `python3`，並繼續進行本教學課程的其餘部分。

   如果命令顯示 Python 版本，則表示已安裝 Python。本教學課程需要 Python v3.7 或更新版本。
3. 如果已安裝 Python，則可以跳過本節的其餘步驟。如果未安裝，請繼續。
4. 開啟 https://www.python.org/downloads/，並下載適用於您電腦的安裝程式。
5. 如果下載未自動開始安裝，請執行下載的程式來安裝 Python。
6. 驗證 Python 的安裝。

   ```
   python -V
   ```

   確認命令顯示 Python 版本。如果未顯示 Python 版本，請嘗試再次下載並安裝 Python。

安裝適用於 Python 的 AWS IoT 裝置 SDK

在電腦上安裝適用於 Python 的 AWS IoT 裝置 SDK

1. 安裝適用於 Python 的 AWS IoT 裝置 SDK v2。

```
python3 -m pip install awsiotsdk
```

2. 將適用於 Python 的 AWS IoT 裝置 SDK 儲存庫複製到主目錄的 aws-iot-device-sdk-python-v2 目錄中。此程序會將所安裝檔案的目錄稱為 *home*。

   home 目錄的實際位置取決於您的作業系統。

   Linux/macOS

   在 macOS 和 Linux 中，*home* 目錄為 ~。

   ```
   cd ~
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   Windows

   在 Windows 中，您可以在 cmd 視窗中執行此命令，來尋找 *home* 目錄路徑。

   ```
   echo %USERPROFILE%
   cd %USERPROFILE%
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   Note

   如果您是使用 Windows PowerShell，而不是 cmd.exe，則會使用下列命令。

   ```
   echo $home
   ```

如需詳細資訊，請參閱適用於 Python GitHub 儲存庫的AWS IoT 裝置 SDK。

準備執行範例應用程式

準備您的系統以執行範例應用程式

- 建立 certs 目錄。將下列項目複製至 certs 目錄：私有金鑰、裝置憑證，以及您在 the section called “建立 AWS IoT 資源” 建立和註冊物件時所儲存的根憑證授權機構憑證檔案。目的地目錄中每個檔案的檔案名稱應與表格中的檔案名稱相符。

  下節命令假設您的金鑰和憑證檔案儲存在您的裝置上，如下表所示。

  Linux/macOS

  執行此命令來建立 certs 子目錄，您會在執行範例應用程式時使用這個子目錄。

  ```
  mkdir ~/certs
  ```

  在新的子目錄中，將檔案複製到下表所示的目的地檔案路徑。

  憑證檔案名稱

檔案	檔案路徑
私有金鑰	~/certs/private.pem.key
裝置憑證	~/certs/device.pem.crt
根 CA 憑證	~/certs/Amazon-root-CA-1.pem

  執行此命令來列出 certs 目錄中的檔案，並將其與表格中列出的檔案進行比較。

  ```
  ls -l ~/certs
  ```

  Windows

  執行此命令來建立 certs 子目錄，您會在執行範例應用程式時使用這個子目錄。

  ```
  mkdir %USERPROFILE%\certs
  ```

  在新的子目錄中，將檔案複製到下表所示的目的地檔案路徑。

憑證檔案名稱

檔案	檔案路徑
私有金鑰	`%USERPROFILE%\certs\private.pem.key`
裝置憑證	`%USERPROFILE%\certs\device.pem.crt`
根 CA 憑證	`%USERPROFILE%\certs\Amazon-root-CA-1.pem`

執行此命令來列出 `certs` 目錄中的檔案，並將其與表格中列出的檔案進行比較。

```
dir %USERPROFILE%\certs
```

設定政策和執行範例應用程式

在本節中，您會設定政策，並執行 `pubsub.py` 範例指令碼，其位於 適用於 Python 的 AWS IoT Device SDK的 `aws-iot-device-sdk-python-v2/samples` 目錄中。此指令碼顯示您的裝置如何使用 MQTT 程式庫來發佈和訂閱 MQTT 訊息。

`pubsub.py` 範例應用程式會訂閱主題 `test/topic`、將 10 則訊息發佈至該主題，以及在從訊息代理程式接收訊息時顯示這些訊息。

若要執行 `pubsub.py` 範例指令碼，您需要以下資訊：

應用程式參數值

參數	可在哪裡找到值
your-iot-endpoint	1. 在 AWS IoT 主控台的左側選單中，選擇 Settings (設定)。 2. 在 Settings (設定) 頁面上，您的端點會顯示在 Device data endpoint (裝置資料端點) 區段中。

your-iot-endpoint 值的格式為：*endpoint_id*-ats.iot.*region*.amazonaws.com，例如 a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com。

執行指令碼之前，請確定您物件的政策許可範例指令碼連接、訂閱、發佈和接收。

尋找並檢閱物件資源的政策文件

1. 在 AWS IoT 主控台的 Things (物件) 清單中，尋找代表您裝置的資源。
2. 選擇代表您裝置物件資源的 Name (名稱) 連結，以開啟 Thing details (物件詳細資訊) 頁面。
3. 在 Thing details (物件詳細資訊) 頁面中的 Certificates (憑證) 索引標籤上，選擇與物件資源相連的憑證。清單中應該只有一個憑證。如果有一個以上的憑證，請選擇其檔案已安裝在裝置上且將用來連接至 AWS IoT Core的憑證。

   於 Certificate (憑證) 詳細資訊頁面的 Policies (政策) 索引標籤中，選擇與憑證相連的政策。其中應只有一個政策。如果有多個政策，請針對每個政策重複下一個步驟，以確保至少有一個政策授予必要的存取權。
4. 在 Policy (政策) 概觀頁面中，找到 JSON 編輯器並選擇 Edit policy document (編輯政策文件)，視需要檢閱和編輯政策文件。
5. 政策 JSON 會顯示在下列範例中。在 "Resource"元素中，將 取代*region:account*為每個Resource值 AWS 帳戶 中的 AWS 區域 和 。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
            "arn:aws:iot:us-east-1:123456789012:topicfilter/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/test-*"
            ]
        }
    ]
}
```

Linux/macOS

在 Linux/macOS 上執行範本指令碼

1. 在命令列視窗中，導覽至 SDK 使用這些命令所建立的 ~/aws-iot-device-sdk-python-v2/samples/node/pub_sub 目錄。

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 在您的命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key
```

Windows

在 Windows PC 上執行範例應用程式

1. 在命令列視窗中，導覽至 SDK 所建立的 %USERPROFILE%\aws-iot-device-sdk-python-v2\samples 目錄，然後使用這些命令來安裝範例應用程式。

```
cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
```

2. 在您的命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file %USERPROFILE%
\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key
 %USERPROFILE%\certs\private.pem.key
```

範例指令碼：

1. 連線至 AWS IoT Core 您帳戶的 。
2. 訂閱訊息主題 test/topic，並顯示其收到關於該主題的訊息。
3. 將 10 則訊息發佈至主題 test/topic。
4. 顯示類似下列內容的輸出：

```
Connected!
Subscribing to topic 'test/topic'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'test/topic': Hello World! [1]
Received message from topic 'test/topic': b'"Hello World! [1]"'
Publishing message to topic 'test/topic': Hello World! [2]
Received message from topic 'test/topic': b'"Hello World! [2]"'
Publishing message to topic 'test/topic': Hello World! [3]
Received message from topic 'test/topic': b'"Hello World! [3]"'
Publishing message to topic 'test/topic': Hello World! [4]
Received message from topic 'test/topic': b'"Hello World! [4]"'
Publishing message to topic 'test/topic': Hello World! [5]
Received message from topic 'test/topic': b'"Hello World! [5]"'
Publishing message to topic 'test/topic': Hello World! [6]
Received message from topic 'test/topic': b'"Hello World! [6]"'
Publishing message to topic 'test/topic': Hello World! [7]
Received message from topic 'test/topic': b'"Hello World! [7]"'
Publishing message to topic 'test/topic': Hello World! [8]
Received message from topic 'test/topic': b'"Hello World! [8]"'
Publishing message to topic 'test/topic': Hello World! [9]
Received message from topic 'test/topic': b'"Hello World! [9]"'
Publishing message to topic 'test/topic': Hello World! [10]
Received message from topic 'test/topic': b'"Hello World! [10]"'
10 message(s) received.
Disconnecting...
Disconnected!
```

若在執行範例應用程式時發生問題，請檢閱 the section called “對範例應用程式的問題進行故障診斷”。

您也可以將 `--verbosity Debug` 參數新增至命令列，以便範例應用程式顯示有關其正在做什麼的詳細訊息。該資訊可能會協助您更正問題。

在 AWS IoT 主控台中檢視來自範例應用程式的訊息

您可以在範例應用程式的訊息透過訊息代理程式傳遞時看到這些訊息，方法為使用 AWS IoT 主控台中的 MQTT test client (MQTT 測試用戶端)。

檢視範例應用程式所發佈的 MQTT 訊息

1. 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息。這有助於您了解如何使用 AWS IoT 主控台中的 MQTT 測試用戶端，在 MQTT 訊息通過訊息代理程式時，檢視訊息。
2. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。
3. 在 Subscribe to a topic (訂閱主題) 中，訂閱主題 test/topic。
4. 在您的命令列視窗中，再次執行範例應用程式，並在 AWS IoT 主控台的 MQTT client (MQTT 用戶端) 中觀看訊息。

   Linux/macOS

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic test/topic --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   Windows

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   python3 pubsub.py --topic test/topic --ca_file %USERPROFILE%\certs\Amazon-root-
   CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs
   \private.pem.key --endpoint your-iot-endpoint
   ```

如需 MQTT 和 如何 AWS IoT Core 支援通訊協定的詳細資訊，請參閱 MQTT。

在 Python 中執行共享訂閱範例

AWS IoT Core 同時支援 MQTT 3 和 MQTT 5 的共用訂閱。共享訂閱允許多個用戶端共享一個主題的訂閱，而且只有一個用戶端會使用隨機分佈接收發佈至該主題的訊息。若要使用共享訂閱，用戶端會訂閱共享訂閱的主題篩選條件：`$share/{ShareName}/{TopicFilter}`。

設定政策並執行共享訂閱範例

1. 若要執行共享訂閱範例，您必須依照 MQTT 5 共享訂閱中所述來設定物件的政策。
2. 若要執行共享訂閱，請執行下列命令。

   Linux/macOS

   在 Linux/macOS 上執行範本指令碼

   1. 在命令列視窗中，導覽至 SDK 使用這些命令所建立的 `~/aws-iot-device-sdk-python-v2/samples` 目錄。

      ```
      cd ~/aws-iot-device-sdk-python-v2/samples
      ```

   2. 在您的命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

      ```
      python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
       ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
      private.pem.key --group_identifier consumer
      ```

   Windows

   在 Windows PC 上執行範例應用程式

   1. 在命令列視窗中，導覽至 SDK 所建立的 `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples` 目錄，然後使用這些命令來安裝範例應用程式。

      ```
      cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
      ```

   2. 在您的命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

      ```
      python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
       %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs
      \device.pem.crt --key %USERPROFILE%\certs\private.pem.key --group_identifier
       consumer
      ```

> **Note**
>
> 執行範例 (例如 `--group_identifier consumer`) 時，您可以根據需求選擇性地指定群組識別碼。如果您沒有指定一個，則 `python-sample` 是預設群組識別碼。

3. 此命令列的輸出如下所示：

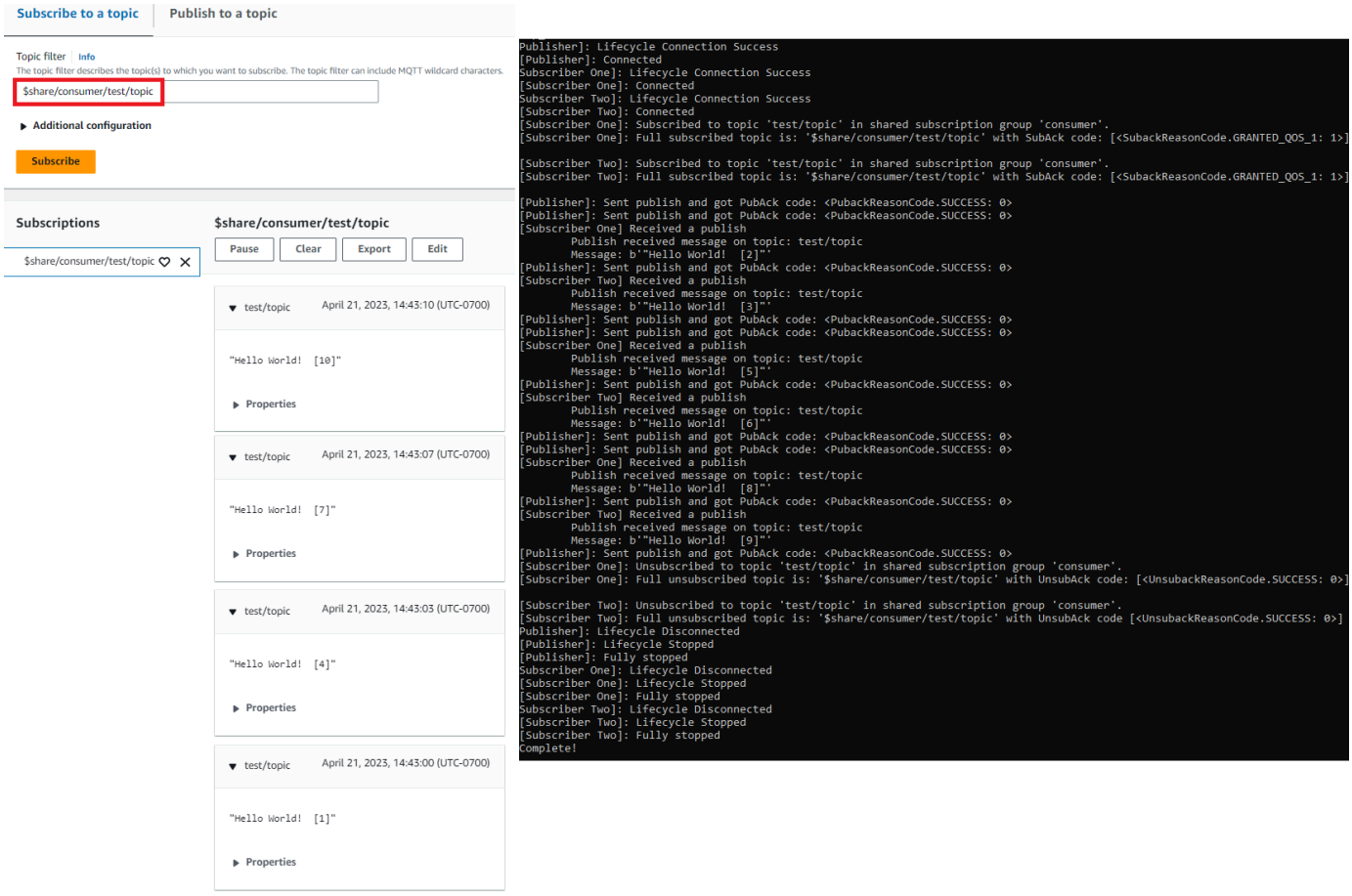
```
Publisher]: Lifecycle Connection Success
[Publisher]: Connected
Subscriber One]: Lifecycle Connection Success
[Subscriber One]: Connected
Subscriber Two]: Lifecycle Connection Success
[Subscriber Two]: Connected
[Subscriber One]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Subscriber Two]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [1]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [2]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [3]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [4]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [5]"'
```

```
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [6]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [7]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [8]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [9]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [10]"'
[Subscriber One]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code: [<UnsubackReasonCode.SUCCESS: 0>]
[Subscriber Two]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code [<UnsubackReasonCode.SUCCESS: 0>]
Publisher]: Lifecycle Disconnected
[Publisher]: Lifecycle Stopped
[Publisher]: Fully stopped
Subscriber One]: Lifecycle Disconnected
[Subscriber One]: Lifecycle Stopped
[Subscriber One]: Fully stopped
Subscriber Two]: Lifecycle Disconnected
[Subscriber Two]: Lifecycle Stopped
[Subscriber Two]: Fully stopped
Complete!
```

4. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。在訂閱主題中，訂閱共享訂閱的主題，例如：$share/consumer/test/topic。執行範例 (例如 --group_identifier consumer) 時，您可以根據需求指定群組識別碼。如果您未指定群組識別碼，則預設值為 python-

sample。如需詳細資訊，請參閱來自《AWS IoT Core 開發人員指南》中的 MQTT 5 共享訂閱 Python 範例和共享訂閱。

在您的命令列視窗中，再次執行範例應用程式，並在 AWS IoT 主控台的 MQTT 測試用戶端，以及命令列中觀看訊息的分佈。



連接 Raspberry Pi 或其他裝置

在本節中，我們將設定 Raspberry Pi 以搭配 使用 AWS IoT。如果您有另一個想要連接的裝置，Raspberry Pi 的指示會包括參考資料，其可以協助您調整這些指示以適用您的裝置。

這通常約需 20 分鐘，但是如果您有許多要安裝的系統軟體升級，則所需時間會更長。

在本教學課程中，您將會執行下列動作：

- 設定您的裝置
- 安裝 AWS IoT 裝置 SDK 所需的工具和程式庫
- 安裝 AWS IoT 裝置 SDK

- 安裝並執行範例應用程式
- 在 AWS IoT 主控台中檢視來自範例應用程式的訊息

> **Important**
>
> 調整這些指示以適用其他裝置和作業系統可能是一項艱鉅的挑戰。您必須充分了解您的裝置，才能解譯這些指示並將其套用至您的裝置。
> 如果您在設定裝置時遇到困難 AWS IoT，可以嘗試其他裝置選項做為替代方案，例如 使用 Amazon EC2 建立虛擬裝置或 使用您的 Windows 或 Linux PC 或 Mac 做為 AWS IoT 裝置。

設定您的裝置

此步驟的目標就是收集您設定裝置所需的資訊，使其可以啟動作業系統 (OS)、連接至網際網路，並可讓您在命令列界面與其互動。

為了完成本教學，您需要以下項目：

- AWS 帳戶。若您沒有帳戶，請完成 設定 AWS 帳戶 所述的步驟，然後再繼續。
- Raspberry Pi 3 代 B 型或更新的型號。這可能適用於較早版本的 Raspberry Pi，但尚未對其進行測試。
- Raspberry Pi OS (32 位元) 或更新版本。我們建議您使用最新版本的 Raspberry Pi 作業系統。較早版本的 OS 可能適用，但尚未對其進行測試。

  若要執行此範例，您不需要安裝具有圖形使用者界面 (GUI) 的桌面；但是，如果您是初次使用 Raspberry Pi，而且您的 Raspberry Pi 支援它，使用具有 GUI 的桌面可能會更輕鬆。
- 乙太網路或 Wi-Fi 連線。
- 裝置所需的鍵盤、滑鼠、顯示器、纜線、電源供應器和其他硬體。

> **Important**
>
> 在您繼續下一個步驟之前，您的裝置必須已安裝、設定和執行作業系統。裝置必須連接至網際網路，而且您必須能夠使用其命令列界面存取裝置。可以透過直接連接的鍵盤、滑鼠和顯示器，或透過 SSH 終端機遠端界面存取命令列。

如果您是在具有圖形使用者界面 (GUI) 的 Raspberry Pi 上執行作業系統，請在裝置上開啟終端機視窗，並在該視窗中執行下列指示。否則，如果您是使用遠端終端機 (例如 PuTTY) 連接到您的裝置，請開啟您裝置的遠端終端機並使用該終端機。

安裝 AWS IoT 裝置 SDK 所需的工具和程式庫

安裝 AWS IoT 裝置 SDK 和範本程式碼之前，請確定您的系統是最新的，並具備安裝 SDKs所需的工具和程式庫。

1. 更新作業系統並安裝必要的程式庫

   安裝 AWS IoT 裝置 SDK 之前，請在裝置上的終端機視窗中執行這些命令，以更新作業系統並安裝所需的程式庫。

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```

   ```
   sudo apt-get install cmake
   ```

   ```
   sudo apt-get install libssl-dev
   ```

2. 安裝 Git

   如果您裝置的作業系統未安裝 Git，您必須安裝它才能安裝適用於 JavaScript 的 AWS IoT 裝置 SDK。

   a. 執行此命令來測試看看是否已安裝 Git。

      ```
      git --version
      ```

   b. 如果上一個命令傳回 Git 版本，表示已安裝 Git，而且您可以跳到步驟 3。

   c. 若在執行 git 命令時顯示錯誤，請執行此命令來安裝 Git。

      ```
      sudo apt-get install git
      ```

   d. 執行此命令來再次測試看看是否已安裝 Git。

      ```
      git --version
      ```

e. 如果已安裝 Git，請繼續進行下一節。如果未安裝，請在繼續之前疑難排解並更正錯誤。您需要 Git 才能安裝適用於 JavaScript 的 AWS IoT 裝置 SDK。

安裝 AWS IoT 裝置 SDK

安裝 AWS IoT 裝置 SDK。

Python

在本節中，您將在裝置上安裝 Python、其開發工具和適用於 Python 的 AWS IoT 裝置 SDK。這些指示適用於執行最新 Raspberry Pi OS 的 Raspberry Pi。若有其他裝置或正在使用其他作業系統，您可能需要調整這些指示以適用您的裝置。

1. 安裝 Python 及其開發工具

   適用於 Python 的 AWS IoT 裝置 SDK 需要在 Raspberry Pi 上安裝 Python v3.5 或更新版本。

   在裝置的終端機視窗中，執行這些命令。

   1. 執行此命令以判定您裝置上安裝的 Python 版本。

      ```
      python3 --version
      ```

      如果已安裝 Python，會顯示其版本。
   2. 如果顯示的版本為 `Python 3.5` 或更新版本，您可以跳到步驟 2。
   3. 如果顯示的版本小於 `Python 3.5`，您可以執行此命令來安裝正確的版本。

      ```
      sudo apt install python3
      ```
   4. 執行此命令以確認現在已安裝正確版本的 Python。

      ```
      python3 --version
      ```
2. 測試 pip3

   在裝置的終端機視窗中，執行這些命令。

   1. 執行此命令以查看是否已安裝 pip3。

```
pip3 --version
```

2. 如果命令傳回版本編號，表示已安裝 pip3，而且您可以跳到步驟 3。
3. 如果上一個命令傳回錯誤，請執行此命令來安裝 pip3。

```
sudo apt install python3-pip
```

4. 執行此命令以查看是否已安裝 pip3。

```
pip3 --version
```

3. 安裝適用於 Python 的目前 AWS IoT 裝置 SDK

   安裝適用於 Python 的 AWS IoT 裝置 SDK，並將範例應用程式下載至您的裝置。

   在您的裝置上，執行這些命令。

```
cd ~
python3 -m pip install awsiotsdk
```

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

JavaScript

在本節中，您將在裝置上安裝 Node.js、npm 套件管理員和適用於 JavaScript 的 AWS IoT 裝置開發套件。這些指示適用於執行 Raspberry Pi OS 的 Raspberry Pi。若有其他裝置或正在使用其他作業系統，您可能需要調整這些指示以適用您的裝置。

1. 安裝最新版本的 Node.js

   適用於 JavaScript 的 AWS IoT 裝置 SDK 需要將 Node.js 和 npm 套件管理員安裝在 Raspberry Pi 上。

   a. 輸入此命令，下載最新版本的 Node 儲存庫。

```
cd ~
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

   b. 安裝 Node 和 npm。

```
sudo apt-get install -y nodejs
```

c. 驗證 Node 的安裝。

```
node -v
```

確認命令顯示 Node 版本。本教學課程需要 Node v10.0 或更新版本。如果未顯示 Node 版本，請嘗試再次下載 Node 儲存庫。

d. 驗證 npm 的安裝。

```
npm -v
```

確認命令顯示 npm 版本。如果未顯示 npm 版本，請嘗試再次安裝 Node 和 npm。

e. 重新啟動裝置。

```
sudo shutdown -r 0
```

在裝置重新啟動之後繼續。

2. 安裝適用於 JavaScript 的 AWS IoT 裝置 SDK

在 Raspberry Pi 上安裝適用於 JavaScript 的 AWS IoT 裝置 SDK。

a. 將適用於 JavaScript 的 AWS IoT 裝置 SDK 儲存庫複製到#aws-iot-device-sdk-js-v2目錄的 目錄中。在 Raspberry Pi 上，*home* 目錄是 ~/，用作下列命令中的 *home* 目錄。如果您的裝置對 *home* 目錄使用不同的路徑，您必須在下列命令中將 ~/ 取代為您裝置的正確路徑。

這些命令會建立 ~/aws-iot-device-sdk-js-v2 目錄並將 SDK 程式碼複製到其中。

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

b. 變更為您在上述步驟中建立的 aws-iot-device-sdk-js-v2 目錄，並執行 npm install 以安裝 SDK。命令 npm install 將會呼叫 aws-crt 程式庫建置，可能需要幾分鐘時間才能完成。

```
cd ~/aws-iot-device-sdk-js-v2
```

```
npm install
```

安裝並執行範例應用程式

在本節中，您將安裝並執行 AWS IoT 裝置開發套件中找到pubsub的範例應用程式。此應用程式顯示您的裝置如何使用 MQTT 程式庫來發佈和訂閱 MQTT 訊息。範例應用程式會訂閱主題 `topic_1`、將 10 則訊息發佈至該主題，以及在從訊息代理程式接收訊息時顯示這些訊息。

安裝憑證檔案

範例應用程式需要驗證裝置是否安裝在裝置上的憑證檔案。

安裝範例應用程式的裝置憑證檔案

1. 執行這些命令，在您的 *home* 目錄中建立 `certs` 子目錄。

   ```
   cd ~
   mkdir certs
   ```

2. 將私有金鑰、裝置憑證，以及您先前在 the section called “建立 AWS IoT 資源” 中建立的根憑證授權機構憑證複製到 `~/certs` 目錄中。

   將憑證檔案複製到裝置的方式取決於裝置和作業系統，此處不會加以描述。不過，如果您的裝置支援圖形使用者界面 (GUI) 且具有網頁瀏覽器，您可以從裝置的網頁瀏覽器執行 the section called “建立 AWS IoT 資源” 中所述的程序，將產生的檔案直接下載到您的裝置。

   下節命令假設您的金鑰和憑證檔案儲存在裝置上，如下表所示。

   憑證檔案名稱

檔案	檔案路徑
根 CA 憑證	`~/certs/Amazon-root-CA-1.pem`
裝置憑證	`~/certs/device.pem.crt`
私有金鑰	`~/certs/private.pem.key`

若要執行範例應用程式，您需要以下資訊：

應用程式參數值

參數	可在哪裡找到值
your-iot-endpoint	於 AWS IoT 主控台中，依序選擇 Manage (管理) 和 Things (物件)。 在選單的設定 AWS IoT 頁面上。您的端點會顯示在 Device data endpoint (裝置資料端點) 區段中。

your-iot-endpoint 值的格式為：*endpoint_id*-ats.iot.*region*.amazonaws.com，例如 a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com。

Python

安裝並執行範例應用程式

1. 導覽至範例應用程式目錄。

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 在命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

```
python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

3. 請注意，範例應用程式：

   1. 連線至您帳戶的 AWS IoT 服務。
   2. 訂閱訊息主題 topic_1，並顯示其收到關於該主題的訊息。
   3. 將 10 則訊息發佈至主題 topic_1。
   4. 顯示類似下列內容的輸出：

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to topic 'topic_1'...
```

```
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'topic_1': Hello World! [1]
Received message from topic 'topic_1': b'Hello World! [1]'
Publishing message to topic 'topic_1': Hello World! [2]
Received message from topic 'topic_1': b'Hello World! [2]'
Publishing message to topic 'topic_1': Hello World! [3]
Received message from topic 'topic_1': b'Hello World! [3]'
Publishing message to topic 'topic_1': Hello World! [4]
Received message from topic 'topic_1': b'Hello World! [4]'
Publishing message to topic 'topic_1': Hello World! [5]
Received message from topic 'topic_1': b'Hello World! [5]'
Publishing message to topic 'topic_1': Hello World! [6]
Received message from topic 'topic_1': b'Hello World! [6]'
Publishing message to topic 'topic_1': Hello World! [7]
Received message from topic 'topic_1': b'Hello World! [7]'
Publishing message to topic 'topic_1': Hello World! [8]
Received message from topic 'topic_1': b'Hello World! [8]'
Publishing message to topic 'topic_1': Hello World! [9]
Received message from topic 'topic_1': b'Hello World! [9]'
Publishing message to topic 'topic_1': Hello World! [10]
Received message from topic 'topic_1': b'Hello World! [10]'
10 message(s) received.
Disconnecting...
Disconnected!
```

若在執行範例應用程式時發生問題，請檢閱 the section called “對範例應用程式的問題進行故障診斷”。

您也可以將 `--verbosity Debug` 參數新增至命令列，以便範例應用程式顯示有關其正在做什麼的詳細訊息。這些資訊可能會提供您更正問題所需的協助。

JavaScript

安裝並執行範例應用程式

1. 在命令列視窗中，導覽至 SDK 所建立的 `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub` 目錄，然後使用這些命令來安裝範例應用程式。命令 `npm install` 將會呼叫 `aws-crt` 程式庫建置，可能需要幾分鐘時間才能完成。

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
```

```
npm install
```

2. 在命令列視窗中，按照指示取代 *your-iot-endpoint* 並執行此命令。

```
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

3. 請注意，範例應用程式：

   1. 連線至您帳戶的 AWS IoT 服務。
   2. 訂閱訊息主題 topic_1，並顯示其收到關於該主題的訊息。
   3. 將 10 則訊息發佈至主題 topic_1。
   4. 顯示類似下列內容的輸出：

```
Publish received on topic topic_1
{"message":"Hello world!","sequence":1}
Publish received on topic topic_1
{"message":"Hello world!","sequence":2}
Publish received on topic topic_1
{"message":"Hello world!","sequence":3}
Publish received on topic topic_1
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

   若在執行範例應用程式時發生問題，請檢閱 the section called “對範例應用程式的問題進行故障診斷”。

您也可以將 `--verbosity Debug` 參數新增至命令列，以便範例應用程式顯示有關其正在做什麼的詳細訊息。這些資訊可能會提供您更正問題所需的協助。

在 AWS IoT 主控台中檢視來自範例應用程式的訊息

您可以在範例應用程式的訊息透過訊息代理程式傳遞時看到這些訊息，方法為使用 AWS IoT 主控台中的 MQTT test client (MQTT 測試用戶端)。

檢視範例應用程式所發佈的 MQTT 訊息

1. 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息。這有助於您了解如何使用 AWS IoT 主控台中的 MQTT 測試用戶端，在 MQTT 訊息通過訊息代理程式時，檢視訊息。
2. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。
3. 訂閱主題 topic_1。
4. 在您的命令列視窗中，再次執行範例應用程式，並在 AWS IoT 主控台的 MQTT client (MQTT 用戶端) 中觀看訊息。

   Python

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   JavaScript

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

對範例應用程式的問題進行故障診斷

如果您在嘗試執行範例應用程式時遇到錯誤，以下是一些需要檢查的事項。

檢查憑證

如果憑證未處於作用中狀態， AWS IoT 將不會接受任何用於授權的連線嘗試。建立憑證時，很容易忽略 Activate (啟用) 按鈕。幸好，您可以從 AWS IoT 主控台啟用憑證。

檢查您的憑證是否啟用

1. 在 AWS IoT 主控台 的左側選單中，依序選擇 Secure (安全) 和 Certificates (憑證)。
2. 在憑證清單中，找出您為練習建立的憑證，並在 Status (狀態) 欄位中檢查其狀態。

   如果您不記得憑證的名稱，請檢查是否有任何為 Inactive (非作用中) 的憑證，以查看它們是否可能是您正在使用的憑證。

   選擇清單中的憑證來開啟其詳細資訊頁面。在詳細資訊頁面中，您可以看到其 Create date (建立日期) 來協助您識別憑證。
3. 若要啟動非作用中的憑證，請從憑證的詳細資訊頁面中，選擇 Actions (動作)，然後選擇 Activate (啟用)。

如果您找到正確的憑證且為作用中，但執行範例應用程式時仍有問題，請檢查其政策，如下一個步驟所述。

您也可以嘗試遵循 the section called “建立物件” 中的步驟，建立新的物件和新的憑證。如果建立新的物件，您需要給它新的物件名稱，並將新的憑證檔案下載到您的裝置。

檢查連接至憑證的政策

政策授權 中的動作 AWS IoT。如果用來連接至 AWS IoT 的憑證沒有政策，或者沒有允許它連接的政策，則即使憑證處於作用中狀態，連線也會遭到拒絕。

檢查連接至憑證的政策

1. 按照上一個項目中所述尋找憑證，並開啟其詳細資訊頁面。
2. 於憑證詳細資訊頁面的左側選單中，選擇 Policies (政策)，查看連接至憑證的政策。
3. 如果沒有任何政策連接至憑證，請選擇 Actions (動作) 選單，然後選擇 Attach policy (連接政策) 來新增一個政策。

   選擇您先前在the section called “建立 AWS IoT 資源”中建立的政策。
4. 若有連接的政策，請選擇政策圖標來開啟其詳細資訊頁面。

在詳細資訊頁面中，檢閱 Policy document (政策文件)，以確定它與您在 the section called “建立 AWS IoT 政策” 中建立的政策包含相同的資訊。

檢查命令列

確定您已針對系統使用正確的命令列。在 Linux 和 macOS 系統上使用的命令通常與在 Windows 系統上使用的命令不同。

檢查端點地址

檢閱您輸入的命令，然後仔細檢查命令中的端點地址是否為 AWS IoT 主控台中的端點地址。

檢查憑證檔案的檔案名稱

將您輸入至命令中的檔案名稱與 `certs` 目錄中的憑證檔案名稱進行比較。

有些系統可能需要以引號括住檔案名稱才能正常運作。

檢查 SDK 安裝

確定您的 SDK 安裝完成且正確。

如有疑問，請在裝置上重新安裝 SDK。在大多數情況下，尋找名為安裝適用於 ***SDK ##***的 AWS IoT 裝置 SDK 教學課程章節並再次遵循程序至關重要。

如果您是使用適用於 JavaScript 的AWS IoT 裝置 SDK，請記住先安裝範例應用程式，然後再嘗試執行它們。安裝 SDK 不會自動安裝範例應用程式。在安裝了 SDK 之後必須手動安裝範例應用程式。

使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

本節說明如何使用AWS IoT 主控台中的 AWS IoT MQTT 測試用戶端來監看由 傳送和接收的 MQTT 訊息 AWS IoT。本節中使用的範例與 AWS IoT Core 教學課程入門 中使用的範例相關；不過，您可以將範例中使用的 *topicName* 取代為 IoT 解決方案所使用的任何主題名稱或主題篩選條件。

裝置會發佈主題識別的 MQTT 訊息，以向其傳達狀態 AWS IoT，並 AWS IoT 發佈 MQTT 訊息，以通知裝置和應用程式變更和事件。您可以使用 MQTT 用戶端來訂閱這些主題，並在發生訊息時觀看它們。您也可以使用 MQTT 測試用戶端，將 MQTT 訊息發佈至 中訂閱的裝置和服務 AWS 帳戶。

目錄

- 在 MQTT 用戶端中檢視 MQTT 訊息

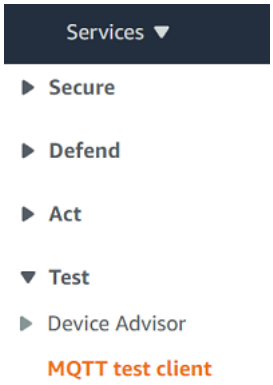
- 從 MQTT 用戶端發佈 MQTT 訊息
- 在 MQTT 用戶端中測試共享訂閱

在 MQTT 用戶端中檢視 MQTT 訊息

下列程序說明如何訂閱裝置發佈訊息的特定 MQTT 主題，並在 AWS IoT 主控台中檢視這些訊息。

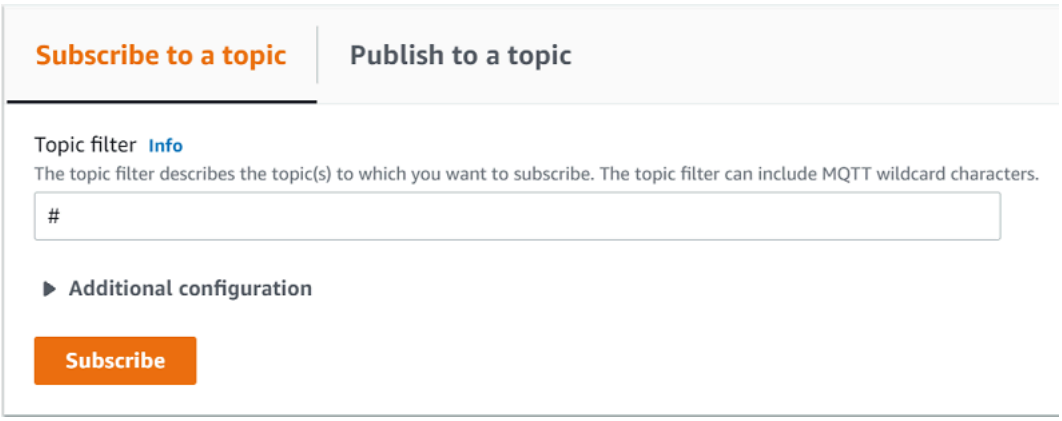
在 MQTT 測試用戶端中檢視 MQTT 訊息

1. 在 AWS IoT 主控台中的左側選單中，依序選擇 Test (測試) 和 MQTT test client (MQTT 測試用戶端)。



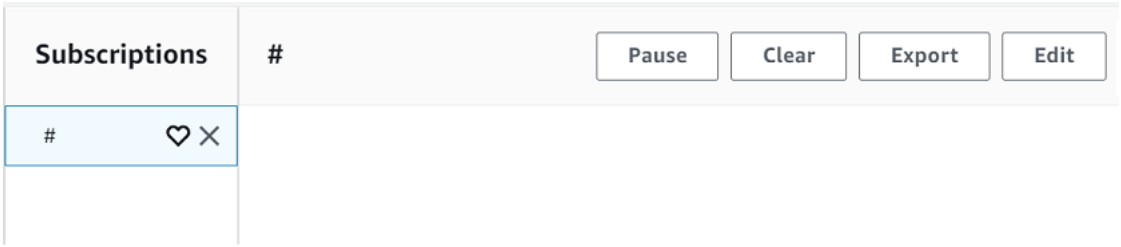
2. 在 Subscribe to a topic (訂閱主題) 索引標籤中，輸入 *topicName* 來訂閱裝置發佈的主題。如需入門範例應用程式，請訂閱 **#**，其會訂閱所有訊息主題。

   繼續入門範例，在 Subscribe to a topic (訂閱主題) 標籤的 Topic filter (主題篩選條件) 欄位中，輸入 **#**，然後選擇 Subscribe (訂閱)。



   主題訊息日誌頁面 # 即會開啟，而且 **#** 會出現在 Subscriptions (訂閱) 清單中。如果您在 中設定的裝置the section called “設定您的裝置”正在執行範例程式，您應該會在 # 訊息日誌 AWS IoT

中看到傳送至 的訊息。收到具有訂閱主題的訊息時，訊息日誌項目會出現在發佈區段下方 AWS IoT。



3. 在 # 訊息日誌頁面上，您也可以將訊息發佈至主題，但您必須指定主題名稱。您無法發佈至 # 主題。

   發佈至已訂閱主題的訊息會在收到時出現在訊息日誌中，最近的訊息首先出現。

疑難排解 MQTT 訊息

使用萬用字元主題篩選條件

如果您的訊息未如預期般顯示在訊息日誌中，請嘗試訂閱萬用字元主題篩選條件，如 主題名稱篩選條件 中所述。MQTT 多層萬用字元主題篩選條件是雜湊或井號 (#)，並且可以用作 Subscription topic (訂閱主題) 欄位中的主題篩選條件。

訂閱 # 主題篩選條件會訂閱訊息代理程式接收的每個主題。您可以縮小篩選範圍，方法是將主題篩選條件路徑的元素取代為 # 多層萬用字元或 '+' 單層萬用字元。

在主題篩選條件中使用萬用字元時

- 多層萬用字元必須是主題篩選條件中的最後一個字元。
- 主題篩選條件路徑的每個主題層只能有一個單層萬用字元。

例如：

主題篩選條件	顯示訊息與
#	任何主題名稱
topic_1/#	開頭為 topic_1/ 的主題名稱
topic_1/level_2/#	開頭為 topic_1/level_2/ 的主題名稱

主題篩選條件	顯示訊息與
`topic_1/+/level_3`	開頭為 `topic_1/`、結尾為 `/level_3`，並且在兩者之間有任何值之一個元素的主題名稱。

如需主題篩選條件的詳細資訊，請參閱 主題名稱篩選條件。

檢查主題名稱錯誤

MQTT 主題和主題篩選條件會區分大小寫。例如，如果您的裝置正在將訊息發佈至 `Topic_1` (具有大寫 T) 而不是 `topic_1` (您已訂閱的主題)，其訊息將不會出現在 MQTT 測試用戶端中。不過，訂閱萬用字元主題篩選條件會顯示裝置正在發佈訊息，而且您可以看到它使用的主題名稱不是您預期的。

從 MQTT 用戶端發佈 MQTT 訊息

將訊息發佈至 MQTT 主題

1. 在 MQTT 測試用戶端頁面的 Publish to a topic (發佈到主題) 標籤中，於 Topic name (主題名稱) 欄位中輸入您訊息的 *topicName*。在此範例中，使用 **my/topic**。

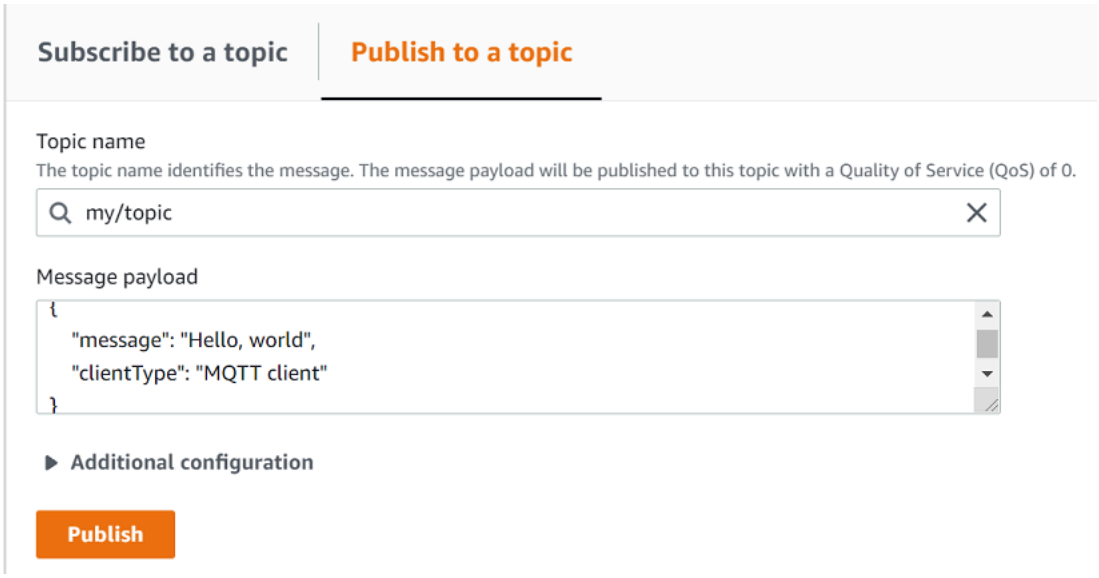
   > **Note**
   >
   > 無論是在 MQTT 測試用戶端還是在系統實作中使用主題名稱，請不要在這些主題名稱中使用個人識別資訊。主題名稱可以出現在未加密的通訊和報告中。

2. 在訊息承載視窗中，輸入下列 JSON：

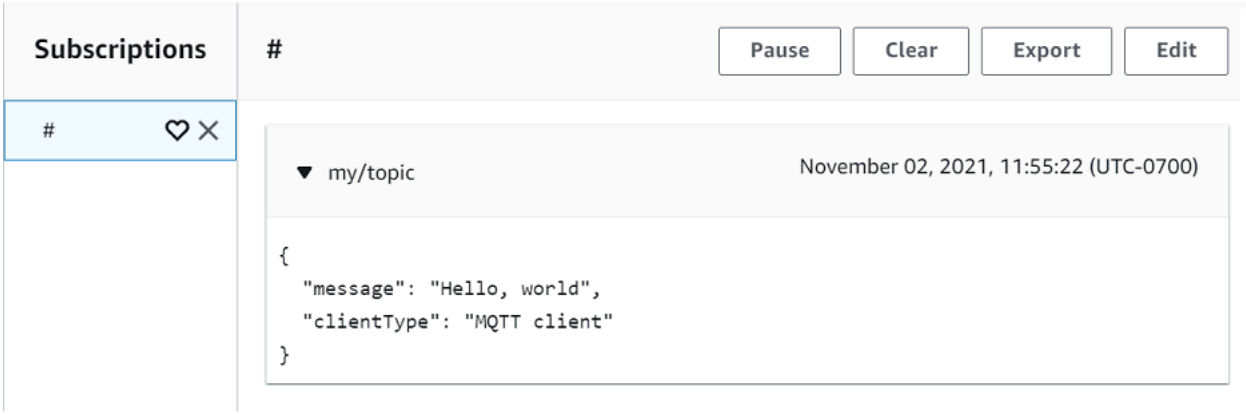
   ```
   {
       "message": "Hello, world",
       "clientType": "MQTT test client"
   }
   ```

3. 選擇 Publish (發佈) 以將您的訊息發佈至 AWS IoT。

   > **Note**
   >
   > 確定您已訂閱 my/topic 主題，然後再發佈您的訊息。



4. 在 Subscriptions (訂閱) 清單中，選擇 my/topic 來查看訊息。您應該會在發佈訊息承載視窗下方看到訊息出現在 MQTT 測試用戶端中。



您可以變更 *Topic name* (主題名稱) 欄位中的 topicName，然後選擇 Publish (發佈) 按鈕，以將 MQTT 郵件發佈至其他主題。

> **Important**
>
> 當您利用重疊主題建立多個訂閱時 (例如 probe1/temperature 和 probe1/#)，同時發布到與兩個訂閱相符的主題的單一訊息，有可能會多次交付，每個重疊訂閱交付一次。

在 MQTT 用戶端中測試共享訂閱

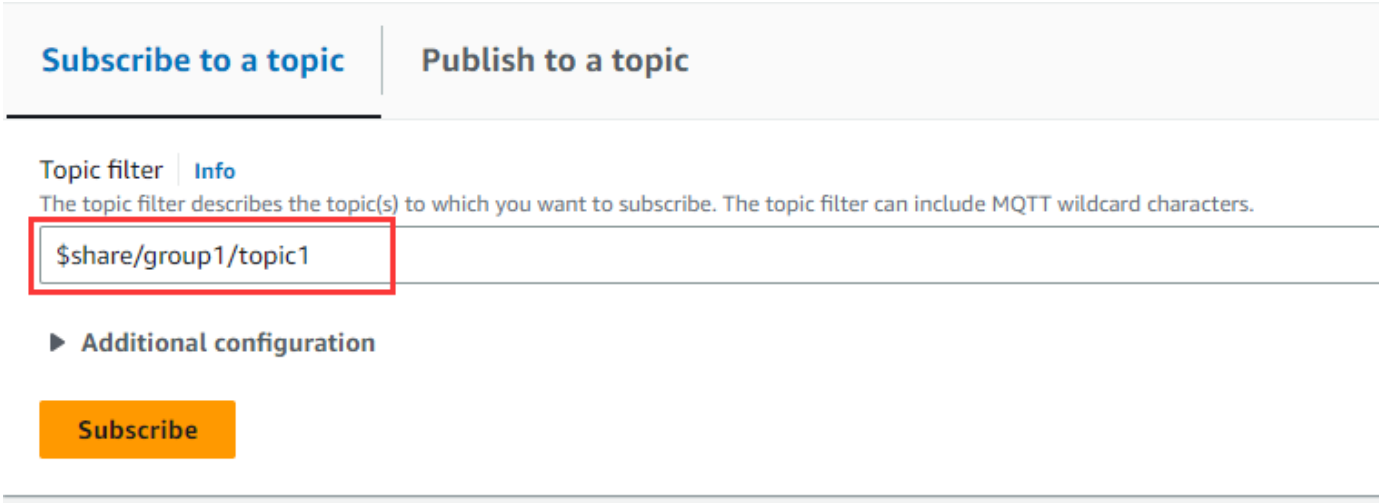
本節說明如何使用AWS IoT 主控台中的 AWS IoT MQTT 用戶端來監看 AWS IoT 使用共用訂閱傳送和接收的 MQTT 訊息。 ??? 允許多個用戶端共用訂閱主題，只有一個用戶端使用隨機分佈接收發佈至該主題的訊息。若要模擬共用相同訂閱的多個 MQTT 用戶端（在此範例中為兩個 MQTT 用戶端），您可以從多個 Web 瀏覽器在AWS IoT 主控台中開啟 AWS IoT MQTT 用戶端。本節中使用的範例與 AWS IoT Core 教學課程入門 中使用的範例無關。如需詳細資訊，請參閱共享訂閱。

共享 MQTT 主題的訂閱

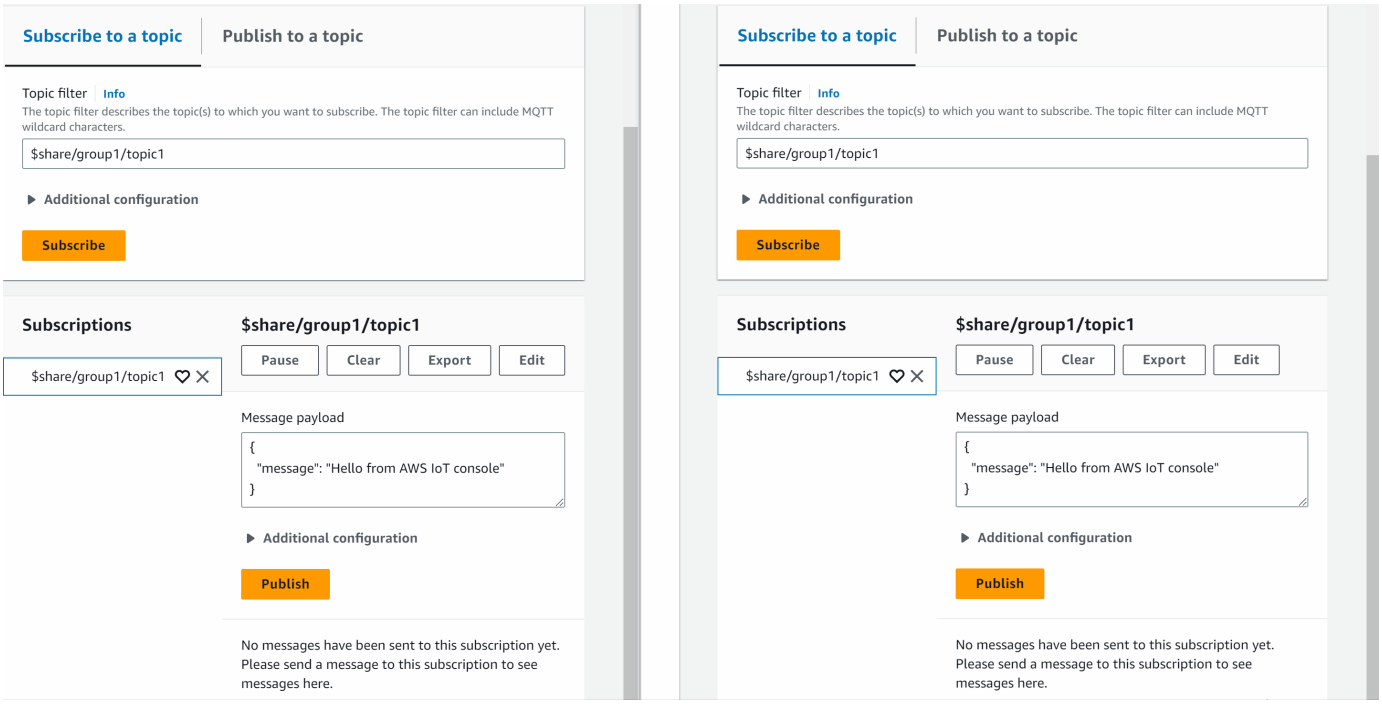
1. 在 AWS IoT 主控台的導覽窗格中，選擇測試，然後選擇 MQTT 測試用戶端。
2. 在 Subscribe to a topic (訂閱主題) 索引標籤中，輸入 *topicName* 來訂閱裝置發佈的主題。若要使用共享訂閱，請訂閱共享訂閱的主題篩選條件，如下所示：

   ```
   $share/{ShareName}/{TopicFilter}
   ```

   範例主題篩選條件可以是 **$share/group1/topic1**，其會訂閱訊息主題 **topic1**。

3. 開啟另一個 Web 瀏覽器，然後重複步驟 1 和步驟 2。透過此方式，您模擬兩個共享相同訂閱 **$share/group1/topic1** 的不同 MQTT 客戶端。
4. 選擇 MQTT 用戶端，在發佈到主題標籤的主題名稱欄位中輸入您訊息的 *topicName*。在此範例中，使用 **topic1**。請嘗試發佈訊息幾次。從這兩個 MQTT 用戶端的訂閱清單中，您應該能夠看到用戶端使用隨機分佈接收訊息。在這個範例中，我們發佈相同的訊息 "Hello from AWS IoT console" 三次。左側的 MQTT 用戶端收到兩次訊息，右側的 MQTT 用戶端收到一次訊息。


Subscribe to a topic
Publish to a topic
Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
$share/group1/topic1
Additional configuration
Subscribe
Subscriptions
$share/group1/topic1
Pause
Clear
Export
Edit
Message payload
{
"message": "Hello from AWS IoT console"
}
Additional configuration
Publish
No messages have been sent to this subscription yet. Please send a message to this subscription to see messages here.

AWS IoT 教學課程

AWS IoT 教學課程分為兩個學習路徑，以支援兩個不同的目標。為目標選擇最佳的學習路徑。

- 您想要建置proof-of-concept，以測試或示範 AWS IoT 解決方案想法

  若要在裝置上使用 AWS IoT 裝置用戶端示範常見的 IoT 任務和應用程式，請遵循the section called "使用 AWS IoT 裝置用戶端建置示範"學習路徑。 AWS IoT Device Client 提供裝置軟體，您可以將自己的雲端資源套用至其中，以示範具有最低開發程度的end-to-end解決方案。

  如需有關 AWS IoT 裝置用戶端的資訊，請參閱 AWS IoT 裝置用戶端。

- 想了解如何建置生產軟體來部署解決方案

  若要使用 AWS IoT 裝置 SDK 建立符合您特定需求的解決方案軟體，請遵循the section called "使用 AWS IoT 裝置SDKs建置解決方案"學習路徑。

  如需可用 AWS IoT 裝置SDKs的詳細資訊，請參閱 ???。如需 AWS SDKs的相關資訊，請參閱要建置的工具 AWS。

AWS IoT 教學課程學習路徑選項

- 使用 AWS IoT 裝置用戶端建置示範
- 使用 AWS IoT 裝置SDKs建置解決方案

使用 AWS IoT 裝置用戶端建置示範

此學習路徑中的教學課程將逐步引導您使用 AWS IoT 裝置用戶端開發示範軟體。 AWS IoT Device Client 提供在您的 IoT 裝置上執行的軟體，以測試和示範 建置之 IoT 解決方案的各個層面 AWS IoT。

這些教學課程的目標是促進探索和實驗，讓您有信心在開發裝置軟體之前 AWS IoT 支援您的解決方案。

您會在本教學課程中學到什麼：

- 如何準備 Raspberry Pi 以用作具有 的 IoT 裝置 AWS IoT
- 如何在裝置上使用 AWS IoT 裝置用戶端示範 AWS IoT 功能

在此學習路徑中，您將在自己的 Raspberry Pi 上安裝 AWS IoT 裝置用戶端，並在雲端中建立 AWS IoT 資源以示範 IoT 解決方案想法。雖然此學習路徑中的教學課程會透過使用 Raspberry Pi 來示範功能，但它們會說明目標和程序，以便協助您調整它們來適應其他裝置。

使用 AWS IoT 裝置用戶端建置示範的先決條件

本節說明在此學習路徑中開始教學課程之前，您需要擁有的先決條件。

若要完成此學習路徑中的教學課程，您需要：

- 一個 AWS 帳戶

  如果您有現有 AWS 帳戶，您可以使用現有的 ，但您可能需要新增其他角色或許可，才能使用 AWS IoT 這些教學課程使用的功能。

  如果您需要建立新的 AWS 帳戶，請參閱 the section called “設定 AWS 帳戶”。

- Raspberry Pi 或兼容的 IoT 裝置

  教學課程使用 Raspberry Pi，因為它有不同的形式，它無處不在，也是一個相對便宜的示範裝置。教學課程已在 Raspberry Pi 3 Model B+、Raspberry Pi 4 Model B，以及在執行 Ubuntu Server 20.04 LTS (HVM) 的 Amazon EC2 執行個體上進行測試。若要使用 AWS CLI 並執行命令，建議您使用最新版的 Raspberry Pi OS (Raspberry Pi OS (64 位元) 或 OS Lite)。較早版本的作業系統可能適用，但我們尚未對其進行測試。

  > **Note**
  >
  > 教學課程會說明每個步驟的目標，以便協助您調整它們來適應尚未嘗試過的 IoT 硬體；不過，這些教學課程並不會明確說明如何調整它們來適應其他裝置。

- 熟悉 IoT 裝置的作業系統

  這些教學課程中的步驟是假設您熟悉使用基本的 Linux 命令和操作 Raspberry Pi 支援的命令列介面。如果不熟悉這些作業，您就需要給自己更多時間來完成教學課程。

  若要完成這些教學課程，您必須了解如何：

  - 安全地執行基本裝置作業，例如組裝和連接元件、將裝置連接至所需電源，以及安裝和移除記憶卡。
  - 在裝置上上傳及下載系統軟體和檔案。如果裝置不使用卸除式儲存裝置 (例如 microSD 卡)，您必須要知道如何連接至裝置，並在裝置上上傳及下載系統軟體和檔案。

- 將裝置連接至打算使用該裝置的網路上。
- 使用 SSH 終端機或類似程式從其他電腦連接至裝置。
- 使用命令列介面來建立、複製、移動、重新命名以及設定裝置上的檔案和目錄許可。
- 在裝置上安裝新程式。
- 使用 FTP 或 SCP 等工具在裝置之間傳輸檔案。

- IoT 解決方案的開發與測試環境

  教學課程會說明所需的軟體和硬體；不過，教學課程會假設您可以執行可能未明確說明的作業。這類硬體和操作的範例包括：

  - 要下載並存放檔案的本機主機電腦

    對於 Raspberry Pi，這通常是個人電腦或筆記型電腦，可以讀取和寫入 microSD 記憶卡。本機主機電腦必須：

    - 連線到網際網路。
    - 已安裝並設定妥 AWS CLI。
    - 擁有支援 AWS 主控台的 Web 瀏覽器。
  - 將本機主機電腦連接到裝置，以便與裝置進行通訊、輸入命令及傳輸檔案的方式

    在 Raspberry Pi 上，這通常是使用 SSH 和 SCP 從本機主機電腦完成。
  - 要連接至 IoT 裝置的顯示器和鍵盤

    這些設備很有幫助，但不需要這些設備就能完成教學課程。
  - 本機主機電腦和 IoT 裝置連線至網際網路的方式

    這可能是連接至網際網路的路由器或閘道的有線或無線網路連線。本機主機也必須能夠連接至 Raspberry Pi。這可能需要它們位於相同的區域網路上。教學課程無法示範如何針對特定裝置或裝置組態進行設定，但是會示範如何測試此連線。
  - 存取區域網路的路由器，檢視連網裝置

    若要完成此學習路徑中的教學課程，您必須能夠找到 IoT 裝置的 IP 地址。

    在區域網路上，您可以透過存取裝置所連線之網路路由器的管理介面來完成這項操作。如果可以在路由器中為裝置指派固定 IP 地址，則可以在每次裝置重新啟動後簡化重新連線的流程。

    如果裝置連接了鍵盤和顯示器，ifconfig 可顯示出裝置的 IP 地址。

    如果這些都不可行，您必須在每次重新啟動裝置後找到一種方法來識別出裝置的 IP 地址。

在擁有所有材料之後，請繼續 the section called “準備使用 IoT 裝置用戶端”。

此學習路徑中的教學課程

- 教學課程：為裝置用戶端準備 AWS IoT 裝置
- 教學課程：安裝和設定 AWS IoT 裝置用戶端
- 教學課程：示範與 AWS IoT 裝置用戶端的 MQTT 訊息通訊
- 教學課程：使用 AWS IoT 裝置用戶端示範遠端動作（任務）
- 教學課程：執行 AWS IoT Device Client 教學課程後清除

教學課程：為裝置用戶端準備 AWS IoT 裝置

本教學課程會帶您逐步進行 Raspberry Pi 的初始化，準備好用於學習路徑的後續教學課程。

本教學課程的目標是安裝裝置作業系統的當前版本，並確保您可以在開發環境中與裝置進行通訊。

先決條件

開始本教學課程之前，請確定您已備妥the section called “使用 AWS IoT 裝置用戶端建置示範的先決條件”可用且可供使用的項目。

此教學課程約需 90 分鐘方能完成。

於本教學課程中，您將會：

- 安裝和更新裝置的作業系統。
- 安裝並驗證執行教學課程所需的任何其他軟體。
- 測試裝置的連線能力並安裝必要的憑證。

完成本教學課程後，下一個教學課程將為使用 裝置用戶端的示範準備 AWS IoT 您的裝置。

本教學課程中的程序

- 安裝和更新裝置的作業系統
- 在裝置上安裝並驗證所需的軟體
- 測試您的裝置並儲存 Amazon CA 憑證

安裝和更新裝置的作業系統

本節中的程序是描述如何將 Raspberry Pi 用於其系統磁碟機的 microSD 卡初始化。Raspberry Pi 的 microSD 卡包含其作業系統 (OS) 軟體及其應用程式檔案儲存的空間。如果不使用 Raspberry Pi，請依照裝置的指示安裝並更新裝置的作業系統軟體。

完成本節之後，您應該可以啟動 IoT 裝置，並從本機主機電腦上的終端機程式連線至該裝置。

必要設備：

- 本機開發和測試環境
- 可以連接至網際網路的 Raspberry Pi 或 IoT 裝置
- 具有至少 8 GB 容量或足夠儲存作業系統和所需軟體的 microSD 記憶卡。

> **Note**
>
> 在為這些練習選取 microSD 卡時，請盡量選擇容量大但體積小的卡片。
> 小型 SD 卡的備份和更新速度會更快。在 Raspberry Pi 上，完成這些教學課程不需要超過 8 GB 的 microSD 卡。如果特定應用程式需要更多空間，您在這些教學課程中儲存的較小映像檔案可以調整較大記憶卡上的檔案系統大小，以使用所選記憶卡的所有支援空間。

選用設備：

- 連接到 Raspberry Pi 的 USB 鍵盤
- HDMI 顯示器和將顯示器至 Raspberry Pi 的纜線

本節中的程序：

- 將裝置的作業系統載入至 microSD 卡
- 使用新的作業系統啟動 IoT 裝置
- 將本機主機電腦連接至裝置

將裝置的作業系統載入至 microSD 卡

此程序會使用本機主機電腦，將裝置的作業系統載入至 microSD 卡。

> **Note**
>
> 如果裝置的作業系統未使用卸除式儲存媒體，請使用該裝置的程序安裝作業系統，然後繼續進行 the section called “啟動您的 IoT 裝置”。

要在 Raspberry Pi 上安裝作業系統

1. 在本機主機電腦上，下載並解壓縮要使用的 Raspberry Pi 作業系統映像。最新版本可從 https://www.raspberrypi.com/software/operating-systems/ 上下載

   選擇 Raspberry Pi 作業系統的版本

   本教學課程會使用 Raspberry Pi OS Lite 版本，因為這是在此學習路徑中支援這些教學課程的最小版本。這個版本的 Raspberry Pi 作業系統只有一個命令列介面，沒有圖形使用者介面。具有圖形化使用者介面的最新 Raspberry Pi 作業系統版本也可以使用這些教學課程；不過，此學習路徑中描述的程序只會使用命令列介面，以便在 Raspberry Pi 上執行操作。
2. 請將 microSD 卡插入主機電腦。
3. 使用 SD 卡影像工具，將解壓縮的作業系統映像檔案寫入 microSD 卡。
4. 寫入 Raspberry Pi 作業系統映像至 microSD 卡後：

   a. 在命令列視窗或檔案總管視窗中，開啟 microSD 卡上的 BOOT 分割區。

   b. 在 microSD 卡的 BOOT 分割區中，於根目錄中建立一個名為 `ssh`且沒有檔案副檔名和內容的空白檔案。這樣會告知 Raspberry Pi 在它第一次啟動時啟用 SSH 通訊。
5. 退出 microSD 卡並安全地從本機主機電腦移除。

您的 microSD 記憶卡已準備進行 the section called “啟動您的 IoT 裝置”。

使用新的作業系統啟動 IoT 裝置

此程序會安裝 microSD 卡，並使用下載的作業系統首次啟動 Raspberry Pi。

使用新的作業系統啟動 IoT 裝置

1. 當電源與裝置斷開時，在上一個步驟中將 microSD 卡 (the section called “載入作業系統”) 插入 Raspberry Pi。
2. 將裝置連線至有線網路。
3. 這些教學課程將使用 SSH 終端機與本機主機電腦的 Raspberry Pi 進行互動。

如果也想直接與裝置互動，您可以：

a. 在將 HDMI 顯示器連接至裝置來查看 Raspberry Pi 的主控台消息之前，您可以將本機主機電腦上的終端機視窗連接至 Raspberry Pi。

b. 如果想直接與 Raspberry Pi 互動，請將 USB 鍵盤連接至裝置。

4. 連接電源至 Raspberry Pi，接著等待大約一分鐘來進行初始化。

   如果有顯示器連接至 Raspberry Pi，您可以觀看它的啟動過程。

5. 找出裝置的 IP 地址：

   - 如果將 HDMI 顯示器連接至 Raspberry Pi，IP 地址會出現在顯示器上的訊息中
   - 如果能存取 Raspberry Pi 連接的路由器，您可以在路由器的管理介面中查看它的地址。

在取得 Raspberry Pi 的 IP 地址後，您已準備好進行 the section called “連接您的主機電腦”。

將本機主機電腦連接至裝置

此程序會使用本機主機電腦上的終端程式連接至 Raspberry Pi，並更改其預設密碼。

將本機主機電腦連接至裝置

1. 在本機主機電腦上，開啟 SSH 終端程式：

   - Windows：PuTTY
   - Linux/macOS：`Terminal`

   > **Note**
   >
   > PuTTY 不會在 Windows 上自動安裝。如果電腦中沒有，您需要下載並安裝 PuTTY。

2. 將終端程式連接至 Raspberry Pi 的 IP 地址，並使用其預設憑證登入。

   ```
   username: pi
   password: raspberry
   ```

3. 登入至 Raspberry Pi 之後，變更 `pi` 使用者密碼。

   ```
   passwd
   ```

依照提示變更密碼。

```
Changing password for pi.
Current password: raspberry
New password: YourNewPassword
Retype new password: YourNewPassword
passwd: password updated successfully
```

在終端視窗收到 Raspberry Pi 命令列提示並更改密碼之後，即表示您已經準備好繼續進行 the section called “安裝並驗證必要的軟體”。

在裝置上安裝並驗證所需的軟體

本節中的程序會從上一節繼續，讓 Raspberry Pi 的作業系統保持在最新狀態，並在 Raspberry Pi 上安裝軟體，該軟體將在下一節中用於建置和安裝 AWS IoT 裝置用戶端。

完成本節後， Raspberry Pi 會擁有最新的作業系統、本學習路徑教學課程所需的軟體，且會根據位置進行設定。

必要設備：

- 上一節中的本機開發和測試環境
- 上一節中使用的 Raspberry Pi
- 上一節中的 microSD 記憶卡

> **Note**
>
> Raspberry Pi Model 3+ 和 Raspberry Pi Model 4 可執行此學習路徑中描述的所有命令。如果您的 IoT 裝置無法編譯軟體或執行 AWS Command Line Interface，您可能需要在本機主機電腦上安裝必要的編譯器，才能建置軟體，然後將其傳輸至 IoT 裝置。如需有關如何為裝置安裝和建置軟體的詳細資訊，請參閱裝置軟體的說明文件。

本節中的程序：

- 更新作業系統軟體
- 安裝必要的應用程式和程式庫
- (選用) 儲存 microSD 卡映像

更新作業系統軟體

此程序會更新作業系統軟體。

更新 Raspberry Pi 上的作業系統軟體

在本機主機電腦的終端機視窗中執行這些步驟。

1. 輸入這些命令來更新 Raspberry Pi 上的系統軟體。

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y autoremove
```

2. 更新 Raspberry Pi 的地區設定和時區設定 (選用)。

   輸入此指令來更新裝置的地區設定和時區設定。

```
sudo raspi-config
```

   a. 若要設定裝置的地區設定：

      i. 在 Raspberry Pi Software Configuration Tool (raspi-config) (Raspberry Pi 軟體組態工具 (raspi-config)) 畫面中，選擇選項 5。

         5 Localisation Options Configure language and regional settings

         使用 Tab 鍵來移動至 <Select> (選擇)，然後按 space bar。

      ii. 在當地語系化選項選單中，選擇選項 L1。

         L1 Locale Configure language and regional settings

         使用 Tab 鍵來移動至 <Select> (選擇)，然後按 space bar。

      iii. 在地區選項清單中，使用方向鍵捲動並使用 space bar 標記所需地區，選擇要在 Raspberry Pi 上安裝的地區設定。

         建議為美國選擇 **en_US.UTF-8**。

      iv. 選取裝置的地區後，請使用 Tab 鍵選擇 <OK> (確定)，然後按 space bar 來顯示 Configuring locales (設定地區設定) 的確認頁面。

   b. 若要設定裝置的時區：

i. 在 raspi-config 畫面中，選擇選項 5。

5 Localisation Options Configure language and regional settings

使用 Tab 鍵來移動至 <Select> (選擇)，然後按 space bar。

ii. 在當地語系化選項選單中，使用方向鍵選擇選項 L2：

L2 time zone Configure time zone

使用 Tab 鍵來移動至 <Select> (選擇)，然後按 space bar。

iii. 在 Configuring tzdata 選單中，從清單中選擇地理區域。

使用 Tab 鍵來移動至 <OK> (確定)，然後按 space bar。

iv. 在城市清單中，使用方向鍵來選擇您所在時區的城市。

若要設定時區，請使用 Tab 鍵來移動至 <OK> (確定)，然後按 space bar。

c. 完成更新設定時，請使用 Tab 鍵來移動至 <Finish> (完成)，然後按 space bar 來關閉 raspi-config 應用程式。

3. 輸入此命令來重新啟動 Raspberry Pi。

```
sudo shutdown -r 0
```

4. 等待 Raspberry Pi 重新啟動。
5. 待 Raspberry Pi 重新啟動後，請將本機主機電腦上的終端機視窗重新連接至 Raspberry Pi。

Raspberry Pi 系統軟體已設定完成，您已經準備好繼續進行 the section called “安裝應用程式和程式庫”。

安裝必要的應用程式和程式庫

此程序會安裝後續教學課程使用的應用程式軟體和程式庫。

如果正在使用 Raspberry Pi，或者如果可以在 IoT 裝置上編譯所需軟體，請在本機主機電腦上的終端機視窗中執行這些步驟。如果必須在本機主機電腦上編譯 IoT 裝置的軟體，請檢閱 IoT 裝置的軟體說明文件，取得如何在裝置上執行這些步驟的相關資訊。

在 Raspberry Pi 上安裝應用程式軟體和程式庫

1. 輸入此指令來安裝應用程式軟體和程式庫。

```
sudo apt-get -y install build-essential libssl-dev cmake unzip git python3-pip
```

2. 輸入這些指令來確認是否已安裝正確版本的軟體。

```
gcc --version
cmake --version
openssl version
git --version
```

3. 確認已安裝下列版本的應用程式軟體：

   - `gcc`：9.3.0 或更新版本
   - `cmake`：3.10.x 或更新版本
   - OpenSSL：1.1.1 或更新版本
   - `git`：2.20.1 或更新版本

如果 Raspberry Pi 有所需應用程式軟體的可接受版本，您就可以繼續進行 the section called “（選用）儲存 microSD 映像”。

(選用) 儲存 microSD 卡映像

在此學習路徑的整個教學課程中，您會遇到這些程序來將 Raspberry Pi 的 microSD 卡映像儲存至本機主機電腦上的檔案。雖然鼓勵這樣操作，但這不是必要任務。透過在建議位置上儲存 microSD 卡映像，您可以略過此學習路徑中儲存點之前的程序；若您發現需要重試某些項目，這就可以節省時間。未定期儲存 microSD 卡映像的結果是，當 microSD 卡損壞或意外錯誤設定應用程式或其設定時，您可能需要從頭開始重新開始學習路徑中的教學課程。

此時，Raspberry Pi 的 microSD 卡已經擁有更新的作業系統和加載的基本應用程式軟體。您現在可以將 microSD 卡的內容儲存至檔案，節省完成上述步驟所花費的時間。擁有裝置 microSD 卡映像目前的映像，可讓您從此開始繼續或重試教學課程或程序，無需從頭開始安裝和更新軟體。

將 microSD 卡映像儲存至檔案

1. 輸入此命令來關閉 Raspberry Pi。

```
sudo shutdown -h 0
```

2. Raspberry Pi 完全關閉後，請移除其電源。
3. 從 Raspberry Pi 中取出 microSD 卡。

4. 在本機主機電腦上：
   a. 插入 microSD 卡。
   b. 使用 SD 卡映像工具，將 microSD 卡的映像儲存至檔案中。
   c. 儲存 microSD 卡的映像後，請從本機主機電腦中退出記憶卡。
5. 在 Raspberry Pi 斷開電源後，將 microSD 卡插入 Raspberry Pi。
6. 將電源連接至 Raspberry Pi。
7. 等待大約一分鐘後，在本機主機電腦上重新連線至連接至 Raspberry Pi 的本機主機電腦終端機視窗，然後登入 Raspberry Pi。

測試您的裝置並儲存 Amazon CA 憑證

本節中的程序會從上一節繼續，安裝用於驗證您連線的 AWS Command Line Interface 和 Certificate Authority 憑證 AWS IoT Core。

完成本節後，您會知道 Raspberry Pi 具有安裝 AWS IoT 裝置用戶端所需的系統軟體，且其具有與網際網路的運作連線。

必要設備：

- 上一節中的本機開發和測試環境
- 上一節中使用的 Raspberry Pi
- 上一節中的 microSD 記憶卡

本節中的程序：

- 安裝 AWS Command Line Interface
- 設定您的 AWS 帳戶 登入資料
- 下載 Amazon 根憑證授權機構憑證
- (選用) 儲存 microSD 卡映像

安裝 AWS Command Line Interface

此程序會將 安裝 AWS CLI 到您的 Raspberry Pi。

如果使用的是 Raspberry Pi，或者如果可以在 IoT 裝置上編譯軟體，請在本機主機電腦上的終端機視窗中執行這些步驟。如果必須在本機主機電腦上編譯 IoT 裝置的軟體，請檢閱 IoT 裝置的軟體說明文件，取得其所需之程式庫的相關資訊。

在 Raspberry Pi AWS CLI 上安裝

1. 執行這些命令來下載和安裝 AWS CLI。

```
export PATH=$PATH:~/.local/bin # configures the path to include the directory with
 the AWS CLI
git clone https://github.com/aws/aws-cli.git # download the AWS CLI code from
 GitHub
cd aws-cli && git checkout v2 # go to the directory with the repo and checkout
 version 2
pip3 install -r requirements.txt # install the prerequisite software
```

2. 執行此命令來安裝 AWS CLI。此命令最多需要 15 分鐘的時間即可完成。

```
pip3 install . # install the AWS CLI
```

3. 執行此命令以確認 AWS CLI 已安裝正確的 版本。

```
aws --version
```

   的版本 AWS CLI 應為 2.2 或更新版本。

如果 AWS CLI 顯示其目前版本，您就可以繼續 the section called “設定帳戶登入資料”。

設定您的 AWS 帳戶 登入資料

在此程序中，您將取得登入資料，並新增這些 AWS 帳戶 登入資料，以便在 Raspberry Pi 上使用。

將 AWS 帳戶 登入資料新增至您的裝置

1. 從 取得存取金鑰 ID 和私密存取金鑰 AWS 帳戶 ，以驗證 AWS CLI 裝置上的 。

   如果您是初次使用 AWS IAM，https AWS ：// AWS https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/。

2. 在連接至 Raspberry Pi 的本機主機電腦終端機視窗上，以及裝置的 Access Key ID (存取金鑰 ID) 和 Secret Access Key (私密存取金鑰) 憑證：

a. 使用此命令執行 AWS configure 應用程式：

```
aws configure
```

b. 出現提示時，請輸入憑證和組態資訊：

```
AWS Access Key ID: your Access Key ID
AWS Secret Access Key: your Secret Access Key
Default region name: your AWS ## code
Default output format: json
```

3. 執行此命令來測試裝置對 AWS 帳戶 和 AWS IoT Core 端點的存取。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

它應該會傳回您的 AWS 帳戶特定 AWS IoT 資料端點，例如此範例：

```
{
    "endpointAddress": "a3EXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

如果您看到 AWS 帳戶特定的 AWS IoT 資料端點，您的 Raspberry Pi 具有繼續 的連線能力和許可the section called “下載 Amazon 根憑證授權機構憑證”。

> **Important**
>
> 您的 AWS 帳戶 登入資料現在會存放在 Raspberry Pi 中的 microSD 卡上。雖然這可讓您在未來 AWS 輕鬆地與 互動，以及您將在這些教學課程中建立的軟體互動，但在預設情況下，它們也會儲存在您在此步驟之後建立的任何 microSD 卡映像中並複製。
>
> 為了保護 AWS 帳戶 登入資料的安全性，在您儲存任何其他 microSD 卡映像之前，請考慮aws configure再次執行 並輸入存取金鑰 ID 和私密存取金鑰的隨機字元來清除登入資料，以防止您的 AWS 帳戶 登入資料洩露。
>
> 如果您發現不小心儲存了 AWS 帳戶 登入資料，您可以在 IAM AWS 主控台中停用登入資料。

下載 Amazon 根憑證授權機構憑證

此程序會下載並儲存 Amazon 根憑證憑證授權機構 (CA) 的憑證副本。下載並儲存此憑證以供後續教學課程使用，也可以測試裝置與 AWS 服務之間的連線能力。

下載並儲存 Amazon 根憑證授權機構憑證

1. 執行此命令，為憑證建立目錄。

```
mkdir ~/certs
```

2. 執行此命令來下載 Amazon 根憑證授權機構憑證。

```
curl -o ~/certs/AmazonRootCA1.pem https://www.amazontrust.com/repository/
AmazonRootCA1.pem
```

3. 執行這些命令來設定憑證目錄及其檔案的存取權。

```
chmod 745 ~
chmod 700 ~/certs
chmod 644 ~/certs/AmazonRootCA1.pem
```

4. 執行此命令來查看新目錄中的憑證授權機構憑證檔案。

```
ls -l ~/certs
```

   您應該會看到類似這樣的項目。日期和時間會有所不同；不過，檔案大小和所有其他資訊應該與此處所示相同。

```
-rw-r--r-- 1 pi pi 1188 Oct 28 13:02 AmazonRootCA1.pem
```

   如果檔案大小不是 1188，請檢查 curl 命令參數。您可能下載了不正確的檔案。

(選用) 儲存 microSD 卡映像

此時，Raspberry Pi 的 microSD 卡已經擁有更新的作業系統和加載的基本應用程式軟體。

將 microSD 卡映像儲存至檔案

1. 在本機主機電腦上的終端機視窗中清除 AWS 憑證。

a. 使用此命令執行 AWS configure 應用程式：

```
aws configure
```

b. 出現提示時，請取代憑證。您可以按 Enter 鍵，讓 Default region name (預設區域名稱) 和 Default output format (預設輸出格式) 保持不變。

```
AWS Access Key ID [****************YT2H]: XYXYXYXYX
AWS Secret Access Key [****************9plH]: XYXYXYXYX
Default region name [us-west-2]:
Default output format [json]:
```

2. 輸入此命令來關閉 Raspberry Pi。

```
sudo shutdown -h 0
```

3. Raspberry Pi 完全關閉之後，移除其電源連接器。
4. 從裝置中取出 microSD 卡。
5. 在本機主機電腦上：

   a. 插入 microSD 卡。

   b. 使用 SD 卡映像工具，將 microSD 卡的映像儲存至檔案中。

   c. 儲存 microSD 卡的映像後，請從本機主機電腦中退出記憶卡。
6. 在 Raspberry Pi 斷開電源後，將 microSD 卡插入 Raspberry Pi。
7. 將電源連接至裝置。
8. 大約一分鐘後，在本機主機電腦上重新啟動終端機視窗工作階段，然後登入裝置。

   尚未重新輸入您的 AWS 帳戶 登入資料。

在重新啟動並登入至 Raspberry Pi 之後，您就可以繼續進行 the section called “安裝和設定 IoT 裝置用戶端”。

教學課程：安裝和設定 AWS IoT 裝置用戶端

本教學課程會逐步解說 AWS IoT 裝置用戶端的安裝和組態，以及您將在此和其他示範中使用的 AWS IoT 資源建立。

若要開始此教學課程：

- 準備好先前教學課程中所述的本機主機電腦和 Raspberry Pi。

此教學課程約需 90 分鐘方能完成。

完成此主題時：

- 您的 IoT 裝置已準備好在其他 AWS IoT 裝置用戶端示範中使用。
- 您會在 中佈建 IoT 裝置 AWS IoT Core。
- 您會在裝置上下載並安裝 AWS IoT 裝置用戶端。
- 您已經儲存了裝置 microSD 卡的映像，以供後續教學課程使用。

必要設備：

- 上一節中的本機開發和測試環境
- 上一節中使用的 Raspberry Pi
- 上一節中使用的 Raspberry Pi microSD 記憶卡

本教學課程中的程序

- 下載並儲存 AWS IoT 裝置用戶端
- 在 中佈建 Raspberry Pi AWS IoT
- 設定 AWS IoT 裝置用戶端以測試連線

下載並儲存 AWS IoT 裝置用戶端

本節中的程序會下載 AWS IoT 裝置用戶端、編譯裝置用戶端，並將其安裝在 Raspberry Pi 上。測試安裝後，您可以儲存 Raspberry Pi 的 microSD 卡映像，以便在再次嘗試教學課程時使用。

本節中的程序：

- 下載並建置 AWS IoT 裝置用戶端
- 建立教學課程所使用的目錄
- (選用) 儲存 microSD 卡映像

下載並建置 AWS IoT 裝置用戶端

此程序會在 Raspberry Pi 上安裝 AWS IoT 裝置用戶端。

在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中執行這些命令。

在 Raspberry Pi 上安裝 AWS IoT 裝置用戶端

1. 輸入這些命令，在您的 Raspberry Pi 上下載並建置 AWS IoT 裝置用戶端。

```
cd ~
git clone https://github.com/awslabs/aws-iot-device-client aws-iot-device-client
mkdir ~/aws-iot-device-client/build && cd ~/aws-iot-device-client/build
cmake ../
```

2. 執行此命令來建置 AWS IoT 裝置用戶端。此命令最多需要 15 分鐘的時間即可完成。

```
cmake --build . --target aws-iot-device-client
```

   顯示為 AWS IoT Device Client 編譯的警告訊息可以忽略。

   這些教學課程已在 gcc2021 年 10 月 30 日版本的 Raspberry Pi OS (bullseye)、版本 (Raspbian 8.3.010.2.1-6+rpi1) 8.3.0 的 2021 年 5 月 7 日版本的 Raspberry Pi OS (buster) 上使用建置於 gcc、版本 (Raspbian 10.2.1-6+rpi1) 20210110 的 AWS IoT Device Client 進行測試。

3. AWS IoT 裝置用戶端完成建置後，請執行此命令來測試它。

```
./aws-iot-device-client --help
```

如果您看到 AWS IoT Device Client 的命令列說明，表示 AWS IoT Device Client 已成功建置並準備好供您使用。

建立教學課程所使用的目錄

此程序會在 Raspberry Pi 上建立目錄，用來存放教學課程在此學習路徑中使用的檔案。

若要在此學習路徑中建立教學課程所使用的目錄：

1. 執行這些命令來建立必要的目錄。

```
mkdir ~/dc-configs
```

```
mkdir ~/policies
mkdir ~/messages
mkdir ~/certs/testconn
mkdir ~/certs/pubsub
mkdir ~/certs/jobs
```

2. 執行這些命令來設定新目錄的許可。

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 700 ~/certs/pubsub
chmod 700 ~/certs/jobs
```

建立這些目錄並設定其許可後，請繼續進行 the section called “(選用) 儲存 microSD 卡映像”。

(選用) 儲存 microSD 卡映像

此時，Raspberry Pi 的 microSD 卡具有更新的作業系統、基本應用程式軟體和 AWS IoT 裝置用戶端。

如果想再次嘗試這些練習和教學課程，可以略過上述程序，方法是將隨此程序儲存的 microSD 卡映像寫入新的 microSD 卡，然後繼續進行 the section called “佈建 Raspberry Pi”。

若要將 microSD 記憶卡映像儲存至檔案中：

在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中：

1. 確認您的 AWS 帳戶 登入資料尚未儲存。

   a. 使用此命令執行 AWS configure 應用程式：

   ```
   aws configure
   ```

   b. 如果憑證已儲存 (若顯示在提示中)，請在提示出現時輸入 **XYXYXYXYX** 字串，如下所示。將 Default region name (預設區域名稱) 和 Default output format (預設輸出格式) 保留為空白。

   ```
   AWS Access Key ID [****************YXYX]: XYXYXYXYX
   AWS Secret Access Key [****************YXYX]: XYXYXYXYX
   Default region name:
   Default output format:
   ```

2. 輸入此命令來關閉 Raspberry Pi。

```
sudo shutdown -h 0
```

3. Raspberry Pi 完全關閉之後，移除其電源連接器。
4. 從裝置中取出 microSD 卡。
5. 在本機主機電腦上：

   a. 插入 microSD 卡。
   b. 使用 SD 卡映像工具，將 microSD 卡的映像儲存至檔案中。
   c. 儲存 microSD 卡的映像後，請從本機主機電腦中退出記憶卡。

您可以在 the section called “佈建 Raspberry Pi” 中繼續使用此 microSD 卡。

在 中佈建 Raspberry Pi AWS IoT

本節中的程序會從已安裝 和 AWS IoT 裝置用戶端的已儲存 microSD AWS CLI 映像開始，並建立佈建 Raspberry Pi AWS IoT 的資源和裝置憑證 AWS IoT。

在 Raspberry Pi 中安裝 microSD 卡

此程序會在 Raspberry Pi 中安裝已載入並設定必要軟體的 microSD 卡，並設定您的 ， AWS 帳戶 以便您可以繼續此學習路徑中的教學課程。

使用 the section called “(選用) 儲存 microSD 卡映像” 中的 microSD 卡，其中包含此學習路徑中的練習和教學課程所需的軟體。

在 Raspberry Pi 安裝 microSD 卡

1. 在 Raspberry Pi 斷開電源後，將 microSD 卡插入 Raspberry Pi。
2. 將電源連接至 Raspberry Pi。
3. 大約一分鐘後，在本機主機電腦上重新啟動終端機視窗工作階段，然後登入 Raspberry Pi。
4. 在本機主機電腦的終端機視窗中，使用 Raspberry Pi 的 Access Key ID (存取金鑰 ID) 和 Secret Access Key (私密存取金鑰) 憑證：

   a. 使用此命令執行 AWS configure 應用程式：

      ```
      aws configure
      ```

   b. 出現提示時，輸入您的 AWS 帳戶 登入資料和組態資訊：

```
AWS Access Key ID [****************YXYX]: your Access Key ID
AWS Secret Access Key [****************YXYX]: your Secret Access Key
Default region name [us-west-2]: your AWS ## code
Default output format [json]: json
```

還原 AWS 帳戶 登入資料後，您就可以繼續 the section called “在 中佈建您的裝置 AWS IoT Core”。

在 中佈建您的裝置 AWS IoT Core

本節中的程序會建立佈建 Raspberry Pi AWS IoT 的資源 AWS IoT。建立這些資源時，系統會要求您記錄各種資訊。裝置 AWS IoT 用戶端組態會在下一個程序中使用此資訊。

若要讓您的 Raspberry Pi 使用 AWS IoT，必須進行佈建。佈建是建立和設定支援 Raspberry Pi 作為 IoT 裝置所需的 AWS IoT 資源的程序。

在 Raspberry Pi 開啟電源和重新啟動之後，請在本機主機電腦上的終端機視窗中連接至 Raspberry Pi 並完成這些程序。

本節中的程序：

- 建立並下載裝置憑證檔案
- 建立 AWS IoT 資源

建立並下載裝置憑證檔案

此程序會建立此示範的裝置憑證檔案。

為 Raspberry Pi 建立並下載裝置憑證檔案

1. 在本機主機電腦的終端機視窗中，輸入這些命令來建立裝置的裝置憑證檔案。

```
mkdir ~/certs/testconn
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/testconn/device.pem.crt" \
--public-key-outfile "~/certs/testconn/public.pem.key" \
--private-key-outfile "~/certs/testconn/private.pem.key"
```

此命令會傳回類似以下的回應。記錄 *certificateArn* 值，供之後使用。

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

2. 輸入下列命令來設定憑證目錄及其檔案的許可。

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 644 ~/certs/testconn/*
chmod 600 ~/certs/testconn/private.pem.key
```

3. 執行此命令來檢閱憑證目錄和檔案的許可。

```
ls -l ~/certs/testconn
```

命令的輸出應該與您在此處看到的相同，但檔案日期和時間會有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

此時，您已在 Raspberry Pi 上安裝了裝置憑證檔案，可以繼續進行 the section called “建立 AWS IoT 資源”。

建立 AWS IoT 資源

此程序 AWS IoT 會透過建立裝置存取 AWS IoT 功能和服務所需的資源，在 中佈建您的裝置。

在 中佈建您的裝置 AWS IoT

1. 在本機主機電腦的終端機視窗中輸入下列命令，取得 AWS 帳戶裝置資料端點的地址。

```
aws iot describe-endpoint --endpoint-type IoT:Data-ATS
```

先前步驟的命令會傳回類似以下的回應：記錄 *endpointAddress* 值，供之後使用。

```
{
    "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. 輸入此命令來建立 Raspberry Pi 的 AWS IoT 物件資源。

```
aws iot create-thing --thing-name "DevCliTestThing"
```

如果您的 AWS IoT 物件資源已建立，命令會傳回類似這樣的回應。

```
{
    "thingName": "DevCliTestThing",
    "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/DevCliTestThing",
    "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 在終端機視窗中：

a. 開啟文字編輯器，例如 nano。

b. 複製此 JSON 政策文件並將其貼入開啟的文字編輯器中。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
```

```
            "Resource": [
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> 本政策文件會慷慨地授予每個資源許可，以便進行連線、接收、發佈和訂閱。通常，政策只會授予特定資源許可，以便執行特定動作。不過，對於初始裝置連線能力測試，這個過度寬鬆的政策能夠在此測試期間盡力降低出現存取問題的機率。在後續教學課程中，將使用範圍較窄的政策文件來示範政策設計的更佳實務。

c. 將文字編輯器中的檔案儲存為 **~/policies/dev_cli_test_thing_policy.json**。

4. 執行此命令以使用先前步驟的政策文件來建立 AWS IoT 政策。

```
aws iot create-policy \
--policy-name "DevCliTestThingPolicy" \
--policy-document "file://~/policies/dev_cli_test_thing_policy.json"
```

如果建立此政策，此命令會傳回類似以下的回應。

```
{
    "policyName": "DevCliTestThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/DevCliTestThingPolicy",
    "policyDocument": "{\n    \"Version\": \"2012-10-17\",       \n    \"Statement
\": [\n        {\n            \"Effect\": \"Allow\",\n            \"Action\": [\n
                \"iot:Publish\",\n                \"iot:Subscribe\",\n
    \"iot:Receive\",\n                \"iot:Connect\"\n            ],\n
 \"Resource\": [\n                \"*\"\n            ]\n        }\n    ]\n}\n",
    "policyVersionId": "1"
}
```

5. 執行此命令，將此政策連接至裝置憑證。使用之前儲存的 `certificateArn` 值來取代 *certificateArn*

```
aws iot attach-policy \
--policy-name "DevCliTestThingPolicy" \
```

```
--target "certificateArn"
```

若成功，此命令不會傳回任何內容。

6. 執行此命令，將裝置憑證連接至 AWS IoT 物件資源。使用之前儲存的 certificateArn 值來取代 *certificateArn*

```
aws iot attach-thing-principal \
--thing-name "DevCliTestThing" \
--principal "certificateArn"
```

若成功，此命令不會傳回任何內容。

成功在 中佈建裝置後 AWS IoT，您就可以繼續 the section called “設定裝置用戶端和測試連線”。

設定 AWS IoT 裝置用戶端以測試連線

本節中的程序會將 AWS IoT 裝置用戶端設定為從 Raspberry Pi 發佈 MQTT 訊息。

本節中的程序：

- 建立組態檔
- 開啟 MQTT 測試用戶端
- 執行 AWS IoT 裝置用戶端

建立組態檔

此程序會建立組態檔案來測試 AWS IoT 裝置用戶端。

建立組態檔案以測試 AWS IoT 裝置用戶端

- 在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中：

  a. 輸入下列命令來建立組態檔的目錄，並設定目錄的許可：

  ```
  mkdir ~/dc-configs
  chmod 745 ~/dc-configs
  ```

  b. 開啟文字編輯器，例如 nano。

  c. 複製此 JSON 文件並將其貼入開啟的文字編輯器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/testconn/device.pem.crt",
  "key": "~/certs/testconn/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "DevCliTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
```

```
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

d. 將##值取代為您 AWS 帳戶 在 中找到之 的裝置資料端點the section called “在 中佈建您的裝置 AWS IoT Core”。

e. 將文字編輯器中的檔案儲存為 **~/dc-configs/dc-testconn-config.json**。

f. 執行此命令來設定新組態檔的許可。

```
chmod 644 ~/dc-configs/dc-testconn-config.json
```

儲存檔案之後，您就可以繼續進行 the section called “開啟 MQTT 測試用戶端”。

開啟 MQTT 測試用戶端

此程序會準備 AWS IoT 主控台中的 MQTT 測試用戶端，以訂閱 AWS IoT 裝置用戶端在執行時發佈的 MQTT 訊息。

準備 MQTT test client (MQTT 測試用戶端) 來訂閱所有 MQTT 訊息

1. 在 AWS IoT 主控台的本機主機電腦上，選擇 MQTT test client (MQTT 測試用戶端)。
2. 在 Subscribe to a topic (訂閱主題) 索引標籤的 Topic filter (主題篩選條件) 中，輸入 **#** (單個井字符號)，然後選擇 Subscribe (訂閱)，即可訂閱每個 MQTT 主題。
3. 在 Subscriptions (訂閱) 標籤下方，請確認看到 **#** (單個井字符號)。

在繼續進行 the section called “執行 AWS IoT 裝置用戶端” 時，請讓有 MQTT test client (MQTT 測試用戶端) 的視窗保持開啟。

執行 AWS IoT 裝置用戶端

此程序會執行 AWS IoT Device Client，以便發佈 MQTT 測試用戶端接收和顯示的單一 MQTT 訊息。

從 AWS IoT 裝置用戶端傳送 MQTT 訊息

1. 請確保這兩個連接到 Raspberry Pi 的終端機視窗和有 MQTT test client (MQTT 測試用戶端) 的視窗會在您執行此程序時顯示出來。

2. 在終端機視窗中，輸入這些命令，以使用在 中建立的組態檔案執行 AWS IoT 裝置用戶端the section called “建立組態檔”。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-testconn-config.json
   ```

   在終端機視窗中， AWS IoT 裝置用戶端會顯示資訊訊息，以及執行時發生的任何錯誤。

   如果終端機視窗中未顯示任何錯誤，請檢閱 MQTT test client (MQTT 測試用戶端)。

3. 在 MQTT test client (MQTT 測試用戶端) 的 Subscriptions (訂閱) 視窗中，請參閱傳送至 `test/dc/pubtopic` 訊息主題的 Hello World! 訊息。
4. 如果 AWS IoT 裝置用戶端未顯示錯誤，您會看到 Hello World！ 傳送至 MQTT 測試用戶端中的`test/dc/pubtopic`訊息，表示連線成功。
5. 在終端機視窗中，輸入 **^C**(Ctrl-C) 以停止 AWS IoT 裝置用戶端。

示範 AWS IoT 裝置用戶端在 Raspberry Pi 上正確執行並可與 通訊後 AWS IoT，您可以繼續前往 the section called “使用 MQTT 與裝置用戶端通訊”。

教學課程：示範與 AWS IoT 裝置用戶端的 MQTT 訊息通訊

本教學課程示範 AWS IoT 裝置用戶端如何訂閱和發佈 MQTT 訊息，這些訊息通常用於 IoT 解決方案。

若要開始此教學課程：

- 將本機主機電腦和 Raspberry Pi 設定成上一節中的使用方式。

  如果您在安裝 AWS IoT 裝置用戶端後儲存 microSD 卡映像，則可以搭配 Raspberry Pi 使用該映像的 microSD 卡。
- 如果您之前已執行過此示範，請檢閱 ??? 以刪除先前執行中建立的所有 AWS IoT 資源，以避免重複的資源錯誤。

此教學課程約需 45 分鐘方能完成。

完成此主題時：

- 您將示範 IoT 裝置從 訂閱 MQTT 訊息 AWS IoT 以及向其發佈 MQTT 訊息的不同方式 AWS IoT。

必要設備：

- 上一節中的本機開發和測試環境
- 上一節中使用的 Raspberry Pi
- 上一節中使用的 Raspberry Pi microSD 記憶卡

本教學課程中的程序

- 準備 Raspberry Pi 以示範 MQTT 訊息通訊
- 示範使用 AWS IoT 裝置用戶端發佈訊息
- 示範使用 AWS IoT 裝置用戶端訂閱訊息

準備 Raspberry Pi 以示範 MQTT 訊息通訊

此程序會在 Raspberry Pi 中 AWS IoT 和 中建立資源，以使用 AWS IoT 裝置用戶端示範 MQTT 訊息通訊。

本節中的程序：

- 建立憑證檔案來示範 MQTT 通訊
- 佈建裝置來示範 MQTT 通訊
- 設定 AWS IoT Device Client 組態檔案和 MQTT 測試用戶端，以示範 MQTT 通訊

建立憑證檔案來示範 MQTT 通訊

此程序會建立此示範的裝置憑證檔案。

為 Raspberry Pi 建立並下載裝置憑證檔案

1. 在本機主機電腦的終端機視窗中輸入下列命令，為裝置建立裝置憑證檔案。

```
mkdir ~/certs/pubsub
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/pubsub/device.pem.crt" \
--public-key-outfile "~/certs/pubsub/public.pem.key" \
--private-key-outfile "~/certs/pubsub/private.pem.key"
```

此命令會傳回類似以下的回應。儲存 *certificateArn* 值，供之後使用。

```
{
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 輸入下列命令來設定憑證目錄及其檔案的許可。

```
chmod 700 ~/certs/pubsub
chmod 644 ~/certs/pubsub/*
chmod 600 ~/certs/pubsub/private.pem.key
```

3. 執行此命令來檢閱憑證目錄和檔案的許可。

```
ls -l ~/certs/pubsub
```

命令的輸出應該與您在此處看到的相同，但檔案日期和時間會有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

4. 輸入這些命令來建立記錄檔案的目錄。

```
mkdir ~/.aws-iot-device-client
mkdir ~/.aws-iot-device-client/log
chmod 745 ~/.aws-iot-device-client/log
echo " " > ~/.aws-iot-device-client/log/aws-iot-device-client.log
```

```
echo " " > ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
chmod 600 ~/.aws-iot-device-client/log/*
```

佈建裝置來示範 MQTT 通訊

本節會建立在 中佈建 Raspberry Pi AWS IoT 的資源 AWS IoT。

若要在 AWS IoT中佈建裝置：

1. 在本機主機電腦的終端機視窗中輸入下列命令，取得 AWS 帳戶裝置資料端點的地址。

   ```
   aws iot describe-endpoint --endpoint-type IoT:Data-ATS
   ```

   自您依照上一個教學課程執行此命令後，端點值便未發生變更。此處再次執行命令是為了輕鬆找到資料端點值並將其貼入本教學課程中使用的組態檔。

   先前步驟的命令會傳回類似以下的回應：記錄 *endpointAddress* 值，供之後使用。

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 輸入此命令，為您的 Raspberry Pi 建立新的 AWS IoT 物件資源。

   ```
   aws iot create-thing --thing-name "PubSubTestThing"
   ```

   由於 AWS IoT 物件資源是雲端中裝置的虛擬表示法，因此我們可以在 中建立多個物件資源 AWS IoT ，以用於不同的用途。這些資源都可以由同一個實體 IoT 裝置使用，用以代表裝置的不同面向。

   這些教學課程一次只會使用一個物件資源來表示 Raspberry Pi。如此一來，在這些教學課程中，它們代表不同的示範，因此在您為示範建立 AWS IoT 資源後，您可以返回並使用您專門為每個示範建立的資源重複示範。

   如果您的 AWS IoT 物件資源已建立，命令會傳回類似這樣的回應。

   ```
   {
   "thingName": "PubSubTestThing",
   "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/PubSubTestThing",
   "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
   ```

```
}
```

3. 在終端機視窗中：

   a. 開啟文字編輯器，例如 nano。

   b. 複製此 JSON 文件並將其貼入開啟的文字編輯器中。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/PubSubTestThing"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic"
            ]
        }
    ]
}
```

c. 在編輯器中，在政策文件的每個Resource區段中，將 *us-west-2#57EXAMPLE833* 取代為您的 AWS 區域、冒號字元 (:) 和 12 位數字 AWS 帳戶 。

d. 將文字編輯器中的檔案儲存為 **~/policies/pubsub_test_thing_policy.json**。

4. 執行此命令以使用先前步驟的政策文件來建立 AWS IoT 政策。

```
aws iot create-policy \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_test_thing_policy.json"
```

如果建立此政策，此命令會傳回類似以下的回應。

```
{
                                    "policyName": "PubSubTestThingPolicy",
                                    "policyArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
                                    "policyDocument": "{\n\"Version\":
 \"2012-10-17\",       \n\"Statement\": [\n{\n\"Effect\": \"Allow\",\n
\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
                                    "policyVersionId": "1"
                                    }
```

5. 執行此命令，將此政策連接至裝置憑證。使用之前在本節儲存的 certificateArn 值來取代 *certificateArn*

```
aws iot attach-policy \
--policy-name "PubSubTestThingPolicy" \
```

```
--target "certificateArn"
```

若成功，此命令不會傳回任何內容。

6. 執行此命令，將裝置憑證連接至 AWS IoT 物件資源。使用之前在本節儲存的 `certificateArn` 值來取代 *certificateArn*

```
aws iot attach-thing-principal \
--thing-name "PubSubTestThing" \
--principal "certificateArn"
```

若成功，此命令不會傳回任何內容。

成功在 中佈建裝置後 AWS IoT，您就可以繼續 the section called “設定 Device Client 組態檔案和 MQTT 用戶端”。

設定 AWS IoT Device Client 組態檔案和 MQTT 測試用戶端，以示範 MQTT 通訊

此程序會建立組態檔案來測試 AWS IoT 裝置用戶端。

建立組態檔案以測試 AWS IoT 裝置用戶端

1. 在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中：

   a. 開啟文字編輯器，例如 nano。

   b. 複製此 JSON 文件並將其貼入開啟的文字編輯器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/pubsub/device.pem.crt",
  "key": "~/certs/pubsub/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "PubSubTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
```

```
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. 將##值取代為您 AWS 帳戶 在 中找到之 的裝置資料端點the section called “在 中佈建您的裝置 AWS IoT Core”。

d. 將文字編輯器中的檔案儲存為 **~/dc-configs/dc-pubsub-config.json**。

e. 執行此命令來設定新組態檔的許可。

```
chmod 644 ~/dc-configs/dc-pubsub-config.json
```

2. 若要準備 MQTT test client (MQTT 測試用戶端) 來訂閱所有 MQTT 訊息：

a. 在 AWS IoT 主控台的本機主機電腦上，選擇 MQTT test client (MQTT 測試用戶端)。

b. 在 Subscribe to a topic (訂閱主題) 索引標籤的 Topic filter (主題篩選條件) 中，輸入 **#** (單個井字符號)，然後選擇 Subscribe (訂閱)。

c. 在 Subscriptions (訂閱) 標籤下方，請確認看到 **#** (單個井字符號)。

在繼續進行此教學課程時，請讓有 MQTT test client (MQTT 測試用戶端) 的視窗保持開啟。

在儲存檔案並設定 MQTT test client (MQTT 測試用戶端)之後，您就可以繼續進行 the section called "使用 IoT 裝置用戶端發佈訊息"。

示範使用 AWS IoT 裝置用戶端發佈訊息

本節中的程序示範 AWS IoT 裝置用戶端如何傳送預設和自訂 MQTT 訊息。

在上一個步驟為這些練習建立的此政策中的政策陳述式，會授予 Raspberry Pi 執行這些動作的許可：

- **`iot:Connect`**

  提供名為 的用戶端`PubSubTestThing`進行連線，也就是執行 AWS IoT 裝置用戶端的 Raspberry Pi。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Connect"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
  ]
}
```

- **`iot:Publish`**

  授予 Raspberry Pi 許可來發佈擁有 `test/dc/pubtopic` MQTT 主題的訊息。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Publish"
```

```
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
      ]
    }
```

此 `iot:Publish` 動作會授予許可來發佈至資源陣列中列出的 MQTT 主題。這些訊息的內容不受政策陳述式的控制。

使用 AWS IoT 裝置用戶端發佈預設訊息

此程序會執行 AWS IoT 裝置用戶端，以便發佈 MQTT 測試用戶端接收和顯示的單一預設 MQTT 訊息。

從 AWS IoT 裝置用戶端傳送預設 MQTT 訊息

1. 請確保這兩個連接到 Raspberry Pi 的本機主機電腦終端機視窗和有 MQTT test client (MQTT 測試用戶端) 的視窗，會在您執行此程序時顯示出來。
2. 在終端機視窗中，輸入這些命令，以使用在 中建立的組態檔案執行 AWS IoT 裝置用戶端the section called “建立組態檔”。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-config.json
   ```

   在終端機視窗中， AWS IoT 裝置用戶端會顯示資訊訊息，以及執行時發生的任何錯誤。

   如果終端機視窗中未顯示任何錯誤，請檢閱 MQTT test client (MQTT 測試用戶端)。
3. 在 MQTT test client (MQTT 測試用戶端) 的 Subscriptions (訂閱) 視窗中，請參閱傳送至 `test/dc/pubtopic` 訊息主題的 Hello World! 訊息。
4. 如果 AWS IoT 裝置用戶端未顯示錯誤，您會看到 Hello World！ 傳送至 MQTT 測試用戶端中的`test/dc/pubtopic`訊息，表示連線成功。
5. 在終端機視窗中，輸入 **^C**(Ctrl-C) 以停止 AWS IoT 裝置用戶端。

在您示範 AWS IoT Device Client 已發佈預設 MQTT 訊息後，您可以繼續前往 the section called “發佈自訂 MQTT 訊息”。

使用 AWS IoT 裝置用戶端發佈自訂訊息

本節中的程序會建立自訂 MQTT 訊息，然後執行 AWS IoT 裝置用戶端，以便它發佈一次自訂 MQTT 訊息，供 MQTT test client (MQTT 測試用戶端) 接收和顯示。

為 AWS IoT 裝置用戶端建立自訂 MQTT 訊息

在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中執行這些步驟。

為 AWS IoT 要發佈的裝置用戶端建立自訂訊息

1. 在終端機視窗中，開啟文字編輯器，例如 nano。
2. 在文字編輯器中複製並貼入以下 JSON 文件。這將是 AWS IoT 裝置用戶端發佈的 MQTT 訊息承載。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

3. 將文字編輯器的內容儲存為 **~/messages/sample-ws-message.json**。
4. 輸入下列命令，為剛建立的訊息檔案設定許可。

```
chmod 600 ~/messages/*
```

為 AWS IoT 裝置用戶端建立組態檔案，以用於傳送自訂訊息

1. 在終端機視窗中，在 等文字編輯器中nano，開啟現有的 AWS IoT Device Client 組態檔案：**~/dc-configs/dc-pubsub-config.json**。
2. 按如下所示編輯 samples 物件。無需變更此檔案的其他部分。

```
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
```

```
        "publish-file": "~/messages/sample-ws-message.json",
        "subscribe-topic": "test/dc/subtopic",
        "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

3. 將文字編輯器的內容儲存為 **~/dc-configs/dc-pubsub-custom-config.json**。
4. 執行此命令來設定新組態檔的許可。

```
chmod 644 ~/dc-configs/dc-pubsub-custom-config.json
```

使用 AWS IoT 裝置用戶端發佈自訂 MQTT 訊息

此項變更只會影響 MQTT 訊息承載的內容，因此目前的政策會繼續運作。不過，如果 MQTT 主題 (如 ~/dc-configs/dc-pubsub-custom-config.json 中定義的 publish-topic 值) 已變更，iot::Publish 政策陳述式也需要進行修改，以允許 Raspberry Pi 發佈至新的 MQTT 主題。

從 AWS IoT 裝置用戶端傳送 MQTT 訊息

1. 請確保這兩個終端機視窗和有 MQTT test client (MQTT 測試用戶端) 的視窗，會在您執行此程序時顯示出來。此外，請確保 MQTT test client (MQTT 測試用戶端) 仍然訂閱了 # 主題篩選條件。如果未訂閱，請訂閱 # 主題篩選條件。
2. 在終端機視窗中輸入這些命令，使用在 the section called “建立組態檔” 中建立的組態檔來執行 AWS IoT 裝置用戶端。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

   在終端機視窗中， AWS IoT 裝置用戶端會顯示資訊訊息，以及執行時發生的任何錯誤。

   如果終端機視窗中未顯示任何錯誤，請檢閱 MQTT test client (MQTT 測試用戶端)。
3. 在 MQTT test client (MQTT 測試用戶端) 的 Subscriptions (訂閱) 視窗中，參閱傳送至 test/dc/pubtopic 訊息主題的自訂訊息承載。
4. 如果 AWS IoT Device Client 沒有顯示錯誤，而且您在 MQTT 測試用戶端中看到發佈至test/dc/pubtopic訊息的自訂訊息承載，表示您已成功發佈自訂訊息。
5. 在終端機視窗中，輸入 **^C**(Ctrl-C) 以停止 AWS IoT 裝置用戶端。

在您示範 AWS IoT Device Client 已發佈自訂訊息承載之後，您可以繼續 the section called “使用 IoT 裝置用戶端訂閱訊息”。

示範使用 AWS IoT 裝置用戶端訂閱訊息

在本節中，您會示範兩種類型的訊息訂閱：

- 單一主題訂閱
- 萬用字元主題訂閱

為這些練習建立的此政策中的政策陳述式會授予 Raspberry Pi 執行這些動作的許可：

- **`iot:Receive`**

  授予 AWS IoT Device Client 許可，以接收符合`Resource`物件中名為 的 MQTT 主題。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
  ]
}
```

- **`iot:Subscribe`**

  授予 AWS IoT Device Client 許可，以訂閱符合`Resource`物件中名為 的 MQTT 主題篩選條件。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Subscribe"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
  ]
}
```

訂閱單一 MQTT 訊息主題

此程序示範 AWS IoT 裝置用戶端如何訂閱和記錄 MQTT 訊息。

在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中，列出 **~/dc-configs/dc-pubsub-custom-config.json** 的內容或在文字編輯器中開啟檔案來檢閱其內容。找到 samples 物件，它應該如下所示。

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/subtopic",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

請注意 subscribe-topic 值是 MQTT 主題， AWS IoT 裝置用戶端會在執行時訂閱該主題。 AWS IoT Device Client 會將從此訂閱收到的訊息承載寫入 subscribe-file值中名為 的檔案。

從 AWS IoT 裝置用戶端訂閱 MQTT 訊息主題

1. 請確保這兩個終端機視窗和有 MQTT test client (MQTT 測試用戶端) 的視窗，會在您執行此程序時顯示出來。此外，請確保 MQTT test client (MQTT 測試用戶端) 仍然訂閱了 # 主題篩選條件。如果未訂閱，請訂閱 # 主題篩選條件。
2. 在終端機視窗中，輸入這些命令，以使用在 中建立的組態檔案執行 AWS IoT 裝置用戶端the section called “建立組態檔”。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

   在終端機視窗中， AWS IoT 裝置用戶端會顯示資訊訊息，以及執行時發生的任何錯誤。

   如果終端機視窗中未顯示任何錯誤，請在 AWS IoT 主控台中繼續進行。
3. 在 AWS IoT 主控台的 MQTT 測試用戶端中，選擇發佈至主題索引標籤。
4. 在 Topic name (主題名稱) 中輸入 **test/dc/subtopic**。
5. 在 Message payload (訊息承載) 中，檢閱郵件內容。
6. 請選擇 Publish (發佈) 來發佈 MQTT 訊息。
7. 在終端機視窗中，請留意從 AWS IoT Device Client 接收的訊息項目，如下所示。

   ```
   2021-11-10T16:02:20.890Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 45 bytes
   ```

8. 在您看到顯示訊息已接收的訊息項目後，請輸入 **^C**(Ctrl-C) 以停止 AWS IoT 裝置用戶端。
9. 輸入此命令來檢視訊息日誌檔案的結尾，查看您從 MQTT test client (MQTT 測試用戶端) 發佈的訊息。

```
tail ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

藉由檢視日誌檔案中的訊息，您示範了 AWS IoT 裝置用戶端收到您從 MQTT 測試用戶端發佈的訊息。

使用萬用字元訂閱多個 MQTT 訊息主題

這些程序示範 AWS IoT 裝置用戶端如何使用萬用字元訂閱和記錄 MQTT 訊息。若要做到這一點，您必須：

1. 更新 AWS IoT Device Client 用來訂閱 MQTT 主題的主題篩選條件。
2. 更新裝置所使用的政策，以允許新的訂閱。
3. 執行 AWS IoT 裝置用戶端，並從 MQTT 測試主控台發佈訊息。

使用萬用字元 MQTT 主題篩選條件建立用於訂閱多個 MQTT 訊息主題的組態檔

1. 在連接至 Raspberry Pi 本機主機電腦的終端機視窗中，開啟 **~/dc-configs/dc-pubsub-custom-config.json** 進行編輯，並找出 `samples` 物件。
2. 在文字編輯器中，找出 `samples` 物件並更新 `subscribe-topic` 值，如下所示。

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/#",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

新 `subscribe-topic` 值是末尾具有 MQTT 萬用字元的 MQTT 主題篩選條件。本節說明訂閱所有以 `test/dc/` 為開頭的 MQTT 主題。 AWS IoT Device Client 會將從此訂閱收到的訊息承載寫入 中名為 的檔案`subscribe-file`。

3. 將修改的組態檔儲存為 **~/dc-configs/dc-pubsub-wild-config.json**，然後退出編輯器。

修改 Raspberry Pi 使用的政策，以允許訂閱和接收多個 MQTT 消息主題

1. 在連接至 Raspberry Pi 的本機主機電腦上的終端機視窗中，於最喜歡的文字編輯器中開啟 **~/policies/pubsub_test_thing_policy.json** 進行編輯，然後找出檔案中的 `iot::Subscribe` 和 `iot::Receive` 政策陳述式。
2. 在 `iot::Subscribe` 政策陳述式中，更新資源物件中的字串，用 * 取代 `subtopic`，使其如下所示。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Subscribe"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*"
  ]
}
```

> **Note**
>
> 此 MQTT 主題篩選條件萬用字元 是 + (加號) 和 # (井號)。結尾帶有 # 的訂閱請求會訂閱所有以 # 字元前面的字串為開頭的主題 (例如本例中的 `test/dc/`)。
> 不過，政策陳述式中授權此訂閱的資源值必須在主題篩選條件 ARN 中使用 * (星號) 取代 # (井號)。這是因為政策處理器使用的萬用字元與 MQTT 使用的不同。
> 如需有關在政策中為主題和主題篩選條件使用萬用字元的詳細資訊，請參閱 在 MQTT 和 AWS IoT Core 政策中使用萬用字元。

3. 在 `iot::Receive` 政策陳述式中，更新資源物件中的字串，用 * 取代 `subtopic`，使其如下所示。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*"
  ]
}
```

4. 將更新後的政策文件儲存為 **~/policies/pubsub_wild_test_thing_policy.json**，然後退出編輯器。
5. 輸入此指令來更新此教學課程的政策，以使用新的資源定義。

```
aws iot create-policy-version \
--set-as-default \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_wild_test_thing_policy.json"
```

如果命令成功，它會傳回類似這樣的回應。請注意，policyVersionId 現在是 2，表示這是此政策的第二個版本。

如果成功更新了政策，您可以繼續進行下一個程序。

```
{
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
    "policyDocument": "{\n  \"Version\": \"2012-10-17\",       \n  \"Statement
\": [\n    {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n
 \"iot:Connect\"\n      ],\n      \"Resource\": [\n        \"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n      ]\n    },\n    {\n
 \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Publish\"\n      ],
\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/pubtopic\"\n      ]\n    },\n    {\n      \"Effect\": \"Allow\",\n
 \"Action\": [\n        \"iot:Subscribe\"\n      ],\n      \"Resource\": [\n
  \"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*\"\n      ]\n    },\n
   {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Receive\"\n
     ],\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/*\"\n      ]\n    }\n  ]\n}\n",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

如果收到因政策版本過多而無法儲存新政策的錯誤，請輸入此命令來列出政策的目前版本。檢閱此命令傳回的清單，找出可以刪除的政策版本。

```
aws iot list-policy-versions --policy-name "PubSubTestThingPolicy"
```

輸入此命令來刪除不再需要的版本。請注意，您無法刪除預設政策版本。預設政策版本的 isDefaultVersion 值為 true。

```
aws iot delete-policy-version \
--policy-name "PubSubTestThingPolicy" \
--policy-version-id policyId
```

刪除政策版本後，請重試此步驟。

使用更新的組態檔案和政策，您就可以使用 AWS IoT 裝置用戶端示範萬用字元訂閱。

示範 AWS IoT 裝置用戶端如何訂閱和接收多個 MQTT 訊息主題

1. 在 MQTT test client (MQTT 測試用戶端) 中，檢查訂閱。如果 MQTT test client (MQTT 測試用戶端) 訂閱了 **#** 主題篩選條件，請繼續下一個步驟。如果沒有，進入 MQTT test client (MQTT 測試用戶端)，在 Subscribe to a topic (訂閱主題) 索引標籤的 Topic filter (主題篩選條件) 中，輸入 **#** (井號字元)，然後選擇 Subscribe (訂閱) 進行訂閱。
2. 在連接至 Raspberry Pi 的本機主機電腦終端機視窗中，輸入這些命令來啟動 AWS IoT 裝置用戶端。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-wild-config.json
   ```

3. 在本機主機電腦上的終端機視窗中觀看 AWS IoT Device Client 輸出時，請返回 MQTT 測試用戶端。在 Publish to a topic (發佈至主題) 索引標籤中的 Topic name (主題名稱) 輸入 **test/dc/subtopic**，然後選擇 Publish (發佈)。
4. 在終端機視窗中，尋找如下訊息來確認已收到訊息：

   ```
   2021-11-10T16:34:20.101Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 76 bytes
   ```

5. 在本機主機電腦的終端機視窗中觀看 AWS IoT Device Client 輸出時，請返回 MQTT 測試用戶端。在 Publish to a topic (發佈至主題) 索引標籤中的 Topic name (主題名稱) 輸入 **test/dc/subtopic2**，然後選擇 Publish (發佈)。
6. 在終端機視窗中，尋找如下訊息來確認已收到訊息：

   ```
   2021-11-10T16:34:32.078Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 77 bytes
   ```

7. 在您看到確認收到兩個訊息的訊息後，請輸入 **^C**(Ctrl-C) 以停止 AWS IoT 裝置用戶端。

8. 輸入此命令來檢視訊息日誌檔案的結尾，查看您從 MQTT test client (MQTT 測試用戶端) 發佈的訊息。

```
tail -n 20 ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

> **Note**
>
> 記錄檔案僅包含訊息承載。訊息主題不會記錄在接收到的訊息日誌檔案中。
> 您也可以在收到的日誌中看到 AWS IoT Device Client 發佈的訊息。這是因為萬用字元主題篩選條件會包含該訊息主題，有時候，訂閱請求可能由訊息代理程式處理，然後才將發佈的訊息傳送給訂閱者。

日誌檔案中的項目會示範已收到訊息。您可以使用其他主題名稱重複此程序。所有主題名稱以 `test/dc/` 為開頭的訊息應該會被接收並記錄。主題名稱以任何其他文字為開頭的訊息都會被忽略。

在示範 AWS IoT Device Client 如何發佈和訂閱 MQTT 訊息之後，請繼續 教學課程：使用 AWS IoT 裝置用戶端示範遠端動作 （任務）。

教學課程：使用 AWS IoT 裝置用戶端示範遠端動作 （任務）

在這些教學課程中，您要設定並部署任務至 Raspberry Pi，以此示範如何將遠程操作發送至 IoT 裝置。

若要開始此教學課程：

- 將本機主機電腦和 Raspberry Pi 設定成上一節中的使用方式。
- 如果您尚未完成上一節中的教學課程，您可以使用 Raspberry Pi 搭配 microSD卡來嘗試本教學課程，該卡具有您在 中安裝 AWS IoT 裝置用戶端後儲存的影像(選用) 儲存 microSD 卡映像。
- 如果您之前已執行過此示範，請檢閱 ??? 以刪除先前執行中建立的所有 AWS IoT 資源，以避免重複的資源錯誤。

此教學課程約需 45 分鐘方能完成。

完成此主題時：

- 您將示範 IoT 裝置使用 AWS IoT Core 執行受管遠端操作的不同方式 AWS IoT 。

必要設備：

- 上一節中測試的本機開發和測試環境
- 上一節中測試的 Raspberry Pi
- 上一節中測試的 Raspberry Pi microSD 記憶卡

本教學課程中的程序

- 準備 Raspberry Pi 以執行任務
- AWS IoT 使用 AWS IoT 裝置用戶端在 中建立和執行任務

準備 Raspberry Pi 以執行任務

本節中的程序說明如何使用 AWS IoT 裝置用戶端，讓您的 Raspberry Pi 準備好執行任務。

> **Note**
>
> 這些程序是裝置特定的程序。如果想要同時使用多個裝置執行本節中的程序，每個裝置都需要有自己的政策和唯一的裝置特定憑證及物件名稱。若要為每個裝置提供獨特資源，請在按程序所述變更裝置特定元素時，針對每個裝置執行此程序一次。

本教學課程中的程序

- 佈建 Raspberry Pi 來示範任務
- 設定 AWS IoT 裝置用戶端以執行任務代理程式

佈建 Raspberry Pi 來示範任務

本節中的程序 AWS IoT 會透過為 Raspberry Pi 建立 AWS IoT 資源和裝置憑證，在 中佈建您的 Raspberry Pi。

主題

- 建立和下載裝置憑證檔案以示範 AWS IoT 任務
- 建立 AWS IoT 資源以示範 AWS IoT 任務

建立和下載裝置憑證檔案以示範 AWS IoT 任務

此程序會建立此示範的裝置憑證檔案。

如果要準備一個以上的裝置，則必須在每個裝置上執行此程序。

若要為 Raspberry Pi 建立並下載裝置憑證檔案：

在連接至 Raspberry Pi 的本機主機電腦終端機視窗中，輸入這些命令。

1. 請輸入下列命令，為裝置建立裝置憑證檔案。

```
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/jobs/device.pem.crt" \
--public-key-outfile "~/certs/jobs/public.pem.key" \
--private-key-outfile "~/certs/jobs/private.pem.key"
```

此命令會傳回類似以下的回應。儲存 *certificateArn* 值，供之後使用。

```
{
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 輸入下列命令來設定憑證目錄及其檔案的許可。

```
chmod 700 ~/certs/jobs
chmod 644 ~/certs/jobs/*
chmod 600 ~/certs/jobs/private.pem.key
```

3. 執行此命令來檢閱憑證目錄和檔案的許可。

```
ls -l ~/certs/jobs
```

命令的輸出應該與您在此處看到的相同，但檔案日期和時間會有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

在下載裝置憑證檔案至 Raspberry Pi 之後，您就可以繼續進行 the section called “為任務佈建 Raspberry Pi”。

建立 AWS IoT 資源以示範 AWS IoT 任務

建立此裝置 AWS IoT 的資源。

如果要準備一個以上的裝置，則必須為每個裝置上執行此程序。

若要在 AWS IoT中佈建裝置：

在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中：

1. 輸入下列命令，取得 AWS 帳戶裝置資料端點地址。

```
aws iot describe-endpoint --endpoint-type IoT:Data-ATS
```

自您上一次執行此命令後，端點值便未發生變更。此處再次執行命令是為了輕鬆找出資料端點值並貼入本教學課程中使用的組態檔。

describe-endpoint 命令會傳回類似以下的回應。記錄 *endpointAddress* 值，供之後使用。

```
{
"endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. 使用裝置的唯一名稱來取代 *uniqueThingName*。如果想要使用多個裝置執行本教學課程，請賦予每個裝置自己的名稱。例如 **TestDevice01**、**TestDevice02** 等等。

輸入此命令來為您的 Raspberry Pi 建立新的 AWS IoT 物件資源。

```
aws iot create-thing --thing-name "uniqueThingName"
```

由於 AWS IoT 物件資源是雲端中裝置的虛擬表示法，因此我們可以在 中建立多個物件資源 AWS IoT ，以用於不同的用途。這些資源都可以由同一個實體 IoT 裝置使用，用以代表裝置的不同面向。

> **Note**
>
> 當您想要保護多個裝置的政策時，可以使用 ${iot:Thing.*ThingName*} 來取代靜態物件名稱 *uniqueThingName*。

這些教學課程每次只會使用一個物件資源。如此一來，在這些教學課程中，它們代表不同的示範，因此在您為示範建立 AWS IoT 資源後，您可以使用您專門為每個示範建立的資源來返回並重複示範。

如果您的 AWS IoT 物件資源已建立，命令會傳回類似這樣的回應。記錄 *thingArn* 值以供稍後在此裝置上建立要執行的任務時使用。

```
{
"thingName": "uniqueThingName",
"thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/uniqueThingName",
"thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 在終端機視窗中：

   a. 開啟文字編輯器，例如 nano。

   b. 複製此 JSON 文件並將其貼入開啟的文字編輯器中。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
```

```
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeJobExecution",
                "iot:GetPendingJobExecutions",
                "iot:StartNextPendingJobExecution",
                "iot:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

c. 在編輯器的每個政策陳述式 Resource 區段中，使用 AWS 區域來取代 *us-west-2:57EXAMPLE833*、一個冒號字元 (:) 和 12 位數的 AWS 帳戶 編號。

d. 在編輯器的在每個政策陳述式中，使用您賦予此物件資源的物件名稱來取代 *uniqueThingName*。

e. 將文字編輯器中的檔案儲存為 **~/policies/jobs_test_thing_policy.json**。

如果為多個裝置執行此程序，請在每個裝置上將檔案儲存為此檔案名稱。

4. 將 *uniqueThingName* 取代為裝置的物件名稱，然後執行此命令來建立為該裝置量身打造 AWS IoT 的政策。

```
aws iot create-policy \
--policy-name "JobTestPolicyFor uniqueThingName" \
--policy-document "file://~/policies/jobs_test_thing_policy.json"
```

如果建立此政策，此命令會傳回類似以下的回應。

```
{
    "policyName": "JobTestPolicyFor uniqueThingName",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
JobTestPolicyFor uniqueThingName",
    "policyDocument": "{\n\"Version\": \"2012-10-17\",\n\"Statement\": [\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\":
```

```
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
    "policyVersionId": "1"
}
```

5. 使用裝置的物件名稱來取代 *uniqueThingName*，再使用稍早您在本節為此裝置儲存的 certificateArn 值來取代 *certificateArn*，然後執行此命令來將政策連接至裝置憑證。

```
aws iot attach-policy \
--policy-name "JobTestPolicyFor<uniqueThingName>" \
--target "<certificateArn>"
```

若成功，此命令不會傳回任何內容。

6. 將 *uniqueThingName* 取代為裝置的物件名稱，將 取代*certificateArn*為您稍早在本節中儲存certificateArn的值，然後執行此命令以將裝置憑證連接至 AWS IoT 物件資源。

```
aws iot attach-thing-principal \
--thing-name "<uniqueThingName>" \
--principal "<certificateArn>"
```

若成功，此命令不會傳回任何內容。

在成功佈建 Raspberry Pi 之後，您就可以在測試中為另一 Raspberry Pi 重複本節程序；或者，如果所有裝置都已佈建，則繼續進行 the section called “設定 Device Client 以執行任務”。

設定 AWS IoT 裝置用戶端以執行任務代理程式

此程序會為 AWS IoT 裝置用戶端建立組態檔案，以執行任務代理程式：。

注意：如果要準備一個以上的裝置，則必須在每個裝置上執行此程序。

若要建立組態檔案以測試 AWS IoT 裝置用戶端：

1. 在連接至 Raspberry Pi 本機主機電腦上的終端機視窗中：

   a. 開啟文字編輯器，例如 nano。

   b. 複製此 JSON 文件並將其貼入開啟的文字編輯器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/jobs/device.pem.crt",
  "key": "~/certs/jobs/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "uniqueThingName",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": true,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": false,
      "publish-topic": "",
      "publish-file": "",
      "subscribe-topic": "",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
```

```
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. 將##值取代為您 AWS 帳戶 在 中找到之 的裝置資料端點值the section called “在 中佈建您的裝置 AWS IoT Core”。

d. 用此裝置使用的物件名稱取代 *uniqueThingName*。

e. 將文字編輯器中的檔案儲存為 **~/dc-configs/dc-jobs-config.json**。

2. 執行此命令來設定新組態檔的檔案許可。

```
chmod 644 ~/dc-configs/dc-jobs-config.json
```

您不必使用 MQTT test client (MQTT 測試用戶端) 進行此測試。雖然裝置將與 交換任務相關的 MQTT 訊息 AWS IoT，但任務進度訊息只會與執行任務的裝置交換。由於任務進度訊息只會與執行任務的裝置交換，因此您無法從其他裝置訂閱它們，例如 AWS IoT 主控台。

儲存組態檔之後，您就可以繼續進行 the section called “建立和執行 IoT 任務”。

AWS IoT 使用 AWS IoT 裝置用戶端在 中建立和執行任務

本節中的程序會建立任務文件和 AWS IoT 任務資源。建立任務資源後， 會將任務文件 AWS IoT 傳送至任務代理程式將任務文件套用至裝置或用戶端的指定任務目標。

本節中的程序

- 建立和存放 IoT 任務的任務文件
- AWS IoT 在一個 IoT 裝置的 中執行任務

建立和存放 IoT 任務的任務文件

此程序會建立簡單的任務文件，以包含在 AWS IoT 任務資源中。此任務文件會在任務目標上 顯示 "Hello world!"。

若要建立和儲存任務文件：

1. 選取要儲存任務文件的 Amazon S3 儲存貯體。如果沒有可使用的現有 Amazon S3 儲存貯體，就需要建立一個。如需如何建立 Amazon S3 儲存貯體的詳細資訊，請參閱 Amazon S3 入門中的主題。
2. 建立並儲存此任務的任務文件

   a. 在本機主機電腦上，開啟文字編輯器。

   b. 將此文字複製並貼入編輯器中。

   ```
   {
       "operation": "echo",
       "args": ["Hello world!"]
   }
   ```

   c. 在本機主機電腦上，將編輯器的內容儲存至名為 **hello-world-job.json** 的檔案。

   d. 請確認檔案已正確儲存。某些文字編輯器會在儲存文字檔案時自動附加 `.txt` 至檔案名稱。如果編輯器附加了 `.txt` 至檔案名稱，請更正檔案名稱後再繼續。

3. 使用 **hello-world-job.json** 路徑來取代 *path_to_file*；如果該路徑不在當前目錄中，則使用所選儲存貯體的 Amazon S3 儲存貯體路徑取代 *s3_bucket_name*，然後執行此命令將任務文件放入 Amazon S3 儲存貯體中。

   ```
   aws s3api put-object \
   --key hello-world-job.json \
   --body path_to_file/hello-world-job.json --bucket s3_bucket_name
   ```

   識別儲存在 Amazon S3 中的任務文件的任務文件 URL 是透過取代以下 URL 中 *s3_bucket_name* 和 *AWS_region* 來決定。記錄產生的 URL 以便稍後使用為 *job_document_path*

   ```
   https://s3_bucket_name.s3.AWS_Region.amazonaws.com/hello-world-job.json
   ```

   > **Note**
   >
   > AWS 安全性可防止您在 外部開啟此 URL AWS 帳戶，例如使用瀏覽器。根據預設，具有檔案存取權 AWS IoT 的任務引擎會使用 URL。在生產環境中，您需要確保 AWS IoT 服務具有存取儲存在 Amazon S3 中任務文件的許可。

在儲存任務文件的 URL 之後，請繼續進行 the section called “執行單一裝置的任務”。

AWS IoT 在一個 IoT 裝置的 中執行任務

本節中的程序會啟動 Raspberry Pi 上的 AWS IoT Device Client，以在裝置上執行任務代理程式，以等待任務執行。它也會在 中建立任務資源 AWS IoT，將任務傳送至 IoT 裝置並在其上執行。

> **Note**
>
> 此程序只會在單一裝置上執行任務。

若要啟動 Raspberry Pi 任務代理程式：

1. 在連線至 Raspberry Pi 的本機主機電腦上的終端機視窗中，執行此命令來啟動 AWS IoT 裝置用戶端。

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-jobs-config.json
```

2. 在終端機視窗中，確認 AWS IoT 裝置用戶端並顯示這些訊息

```
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Jobs is enabled
                        .
                        .
                        .
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Client base has been notified that
 Jobs has started
2021-11-15T18:45:56.708Z [INFO]  {JobsFeature.cpp}: Running Jobs!
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 startNextPendingJobExecution accepted and rejected
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 nextJobChanged events
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusAccepted for jobId +
2021-11-15T18:45:56.738Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionAccepted with code {0}
2021-11-15T18:45:56.739Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusRejected for jobId +
2021-11-15T18:45:56.753Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToNextJobChanged with code {0}
```

```
2021-11-15T18:45:56.760Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobRejected with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobAccepted with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionRejected with code {0}
2021-11-15T18:45:56.777Z [DEBUG] {JobsFeature.cpp}: Publishing
 startNextPendingJobExecutionRequest
2021-11-15T18:45:56.785Z [DEBUG] {JobsFeature.cpp}: Ack received for
 StartNextPendingJobPub with code {0}
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. 在終端機視窗中看到此訊息之後，請繼續執行下一個程序並建立任務資源。請注意，它可能不是清單中的最後一個項目。

```
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

建立 AWS IoT 任務資源

1. 在本機主機電腦上：

   a. 使用來自 the section called “建立和存放任務文件” 的任務文件 URL 來取代 *job_document_url*。

   b. 使用為裝置建立的物件資源 ARN 來取代 *thing_arn*，然後執行此命令。

   ```
   aws iot create-job \
   --job-id hello-world-job-1 \
   --document-source "job_document_url" \
   --targets "thing_arn" \
   --target-selection SNAPSHOT
   ```

   如果成功，命令會傳回類似以下的結果。

   ```
   {
     "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-job-1",
     "jobId": "hello-world-job-1"
   }
   ```

2. 在終端機視窗中，您應該會看到來自 AWS IoT Device Client 的輸出，如下所示。

```
2021-11-15T18:02:26.688Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Job ids differ
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: Executing job: hello-world-
job-1
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Assuming executable is in PATH
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: About to execute: echo Hello
 world!
2021-11-15T18:10:24.890Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken 3TEWba9Xj6 in the updateJobExecution promises map
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process now running
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process about to call
 execvp
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Parent process now running, child
 PID is 16737
2021-11-15T18:10:24.891Z [DEBUG] {16737}: Hello world!
2021-11-15T18:10:24.891Z [DEBUG] {JobEngine.cpp}: JobEngine finished waiting for
 child process, returning 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job exited with status: 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job executed successfully!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.892Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.892Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken GmQ0HTzWGg in the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
```

```
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken 3TEWba9Xj6
 from the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:24.917Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken GmQ0HTzWGg
 from the updateJobExecution promises map
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:25.861Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. AWS IoT 裝置用戶端正在執行並等待任務時，您可以透過變更 `job-id`值，並從create-job步驟 1 重新執行 來提交另一個任務。

當您完成執行任務時，請在終端機視窗中輸入 ^C(control-C) 以停止 AWS IoT 裝置用戶端。

教學課程：執行 AWS IoT Device Client 教學課程後清除

本教學課程中的程序會在完成此學習路徑中的教學課程時，逐步引導您移除所建立的檔案和資源。

本教學課程中的程序

- 步驟 1：使用 裝置用戶端建置示範後清除 AWS IoT 裝置
- 步驟 2：使用 AWS IoT 裝置用戶端建置示範 AWS 帳戶 後清除您的

步驟 1：使用 裝置用戶端建置示範後清除 AWS IoT 裝置

本教學課程說明在此學習路徑中建置示範後，如何清除 microSD 卡的兩個選項。選擇提供所需安全等級的選項。

請注意，清除裝置的 microSD 卡不會移除您建立的任何 AWS IoT 資源。若要在清理裝置的 microSD 卡後清除 AWS IoT 資源，您應該檢閱 上的教學課程the section called “使用 AWS IoT 裝置用戶端建置示範後清除”。

選項 1：透過重新寫入 microSD 記憶卡進行清除

在完成此學習路徑中的教學課程後，清除 microSD 卡最簡單且最徹底的方法是，使用您在第一次準備裝置時所建立的儲存映像檔案覆寫 microSD 卡。

此程序會使用本機主機電腦，將儲存的 microSD 卡映像寫入 microSD 卡。

> **Note**
>
> 如果裝置未針對其作業系統使用卸除式儲存媒體，請參閱該裝置的程序。

若要將新映像寫入 microSD 卡

1. 在本機主機電腦上，找出要寫入至 microSD 卡的儲存 microSD 卡映像。
2. 請將 microSD 卡插入主機電腦。
3. 使用 SD 卡映像工具，將選取的影像檔案寫入至 microSD 卡。
4. 寫入 Raspberry Pi 作業系統映像至 microSD 卡後，請退出 microSD 卡並安全地將其從本機主機電腦中移除。

您的 microSD 卡已可供使用。

選項 2：透過刪除使用者目錄進行清除

若要在完成教學課程後清理 microSD 卡而不重寫 microSD 卡映像，您可以個別刪除使用者目錄。這並不像從儲存映像重寫 microSD 卡那麼徹底，因為此操作不會移除任何可能已安裝的系統檔案。

如果移除使用者目錄足以滿足需求，您可以遵循此程序。

從裝置刪除此學習路徑的使用者目錄

1. 在連接至裝置的終端機視窗中，執行這些命令來刪除在此學習路徑中建立的使用者目錄、子目錄及其所有檔案。

   > **Note**
   >
   > 在刪除這些目錄和檔案之後，您必須再次完成教學課程，才能執行示範。

   ```
   rm -Rf ~/dc-configs
   rm -Rf ~/policies
   rm -Rf ~/messages
   rm -Rf ~/certs
   rm -Rf ~/.aws-iot-device-client
   ```

2. 在連接至裝置的終端機視窗中，執行這些命令來刪除應用程式來源目錄和檔案。

> **Note**
>
> 這些命令不會解除安裝任何程式。它們只會刪除用於建置和安裝它們的來源檔案。刪除這些檔案後，AWS CLI 和 AWS IoT 裝置用戶端可能無法運作。

```
rm -Rf ~/aws-cli
rm -Rf ~/aws
rm -Rf ~/aws-iot-device-client
```

步驟 2：使用 AWS IoT 裝置用戶端建置示範 AWS 帳戶 後清除您的

這些程序可協助您識別和移除您在完成此學習路徑中的教學課程時建立 AWS 的資源。

清除 AWS IoT 資源

此程序可協助您識別和移除您在完成此學習路徑中的教學課程時建立 AWS IoT 的資源。

AWS IoT 在此學習路徑中建立的資源

教學課程	物件資源	政策資源
the section called “安裝和設定 IoT 裝置用戶端”	DevCliTestThing	DevCliTestThingPolicy
the section called “使用 MQTT 與裝置用戶端通訊”	PubSubTestThing	PubSubTestThingPolicy
the section called “使用 裝置用戶端執行 IoT 任務”	使用者定義 (可能不止一個)	使用者定義 (可能不止一個)

若要刪除 AWS IoT 資源，請遵循您建立的每個物件資源的此程序

1. 使用要刪除的物件資源名稱來取代 *thing_name*，然後執行此命令從本機主機電腦列出連接至物件資源的憑證。

```
aws iot list-thing-principals --thing-name thing_name
```

此命令會傳回像這樣的回應，其中會列出連接至 *thing_name* 的憑證。在大多數情況下，清單中只會有一個憑證。

```
{
    "principals": [
        "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/23853eea3cf0edc7f8a69c74abeafa27b2b52823cab5b3e156295e94b26ae8ac"
    ]
}
```

2. 針對上一個命令列出的每個憑證：

   a. 使用上一個命令的憑證 ID 來取代 *certificate_ID*。憑證 ID 是上一個命令傳回的 ARN 中 `cert/` 後面的英數字元。然後執行此命令來停用憑證。

   ```
   aws iot update-certificate --new-status INACTIVE --certificate-
   id certificate_ID
   ```

   如果成功，此命令不會傳回任何內容。

   b. 使用之前傳回的憑證清單中的憑證 ARN 來取代 *certificate_ARN*，然後執行此命令來列出連接至此憑證的政策。

   ```
   aws iot list-attached-policies --target certificate_ARN
   ```

   此命令會傳回像這樣的回應，其中會列出連接至憑證的政策。在大多數情況下，清單中只會有一個政策。

   ```
   {
       "policies": [
           {
               "policyName": "DevCliTestThingPolicy",
               "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
   DevCliTestThingPolicy"
           }
       ]
   }
   ```

   c. 每個連接至憑證的政策：

i. 使用上一個命令的 policyName 值來取代 *policy_name*，使用憑證的 ARN 來取代 *certificate_ARN*，然後執行此命令從憑證分離政策。

```
aws iot detach-policy --policy-name policy_name --target certificate_ARN
```

如果成功，此命令不會傳回任何內容。

ii. 使用 policyName 值來取代 *policy_name*，然後執行此命令，查看政策是否已連接至任何憑證。

```
aws iot list-targets-for-policy --policy-name policy_name
```

如果命令傳回類似這樣的空白清單，代表政策未連接至任何憑證，您可以繼續列出政策版本。如果仍有連接至政策的憑證，請繼續進行 detach-thing-principal 步驟。

```
{
    "targets": []
}
```

iii. 使用 policyName 值來取代 *policy_name*，然後執行此命令來檢查政策版本。若要刪除政策，該政策必須只有一個版本。

```
aws iot list-policy-versions --policy-name policy_name
```

如果政策只有一個版本 (例如此範例)，您可以跳至 delete-policy 步驟，然後立即刪除政策。

```
{
    "policyVersions": [
        {
            "versionId": "1",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:02:46.778000+00:00"
        }
    ]
}
```

如果政策具有多個版本 (例如此範例)，就必須先刪除具有 `isDefaultVersion` 值的 `false`，才能刪除政策。

```
{
    "policyVersions": [
        {
            "versionId": "2",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:52:04.423000+00:00"
        },
        {
            "versionId": "1",
            "isDefaultVersion": false,
            "createDate": "2021-11-18T01:30:18.083000+00:00"
        }
    ]
}
```

如果需要刪除政策版本，請使用 policyName 值來取代 *policy_name*，使用上一個命令的 versionId 值取代 *version_ID*，然後執行此命令來刪除政策版本。

```
aws iot delete-policy-version --policy-name policy_name --policy-version-
id version_ID
```

如果成功，此命令不會傳回任何內容。

刪除政策版本後，請重複此步驟，直到政策只有一個政策版本為止。

iv. 使用 policyName 值來取代 *policy_name*，然後執行此命令來刪除政策。

```
aws iot delete-policy --policy-name policy_name
```

d. 使用物件的名稱來取代 *thing_name*，再使用憑證的 ARN 來取代 *certificate_ARN*，然後執行此命令來從物件資源分離憑證。

```
aws iot detach-thing-principal --thing-name thing_name --
principal certificate_ARN
```

如果成功，此命令不會傳回任何內容。

e. 使用上一個命令的憑證 ID 來取代 *certificate_ID*。憑證 ID 是上一個命令傳回的 ARN 中 cert/ 後面的英數字元。然後執行此命令來刪除憑證資源。

```
aws iot delete-certificate --certificate-id certificate_ID
```

如果成功，此命令不會傳回任何內容。

3. 使用物件的名稱來取代 *thing_name*，然後執行此命令來刪除該物件。

```
aws iot delete-thing --thing-name thing_name
```

如果成功，此命令不會傳回任何內容。

清除 AWS 資源

此程序可協助您識別和移除您在完成此學習路徑中的教學課程時建立的其他 AWS 資源。

在此學習路徑中建立的其他 AWS 資源

教學課程	Resource Type (資源類型)	資源名稱或 ID
the section called “使用 裝置用戶端執行 IoT 任務”	Amazon S3 物件	hello-world-job.json
the section called “使用 裝置用戶端執行 IoT 任務”	AWS IoT 任務資源	使用者定義

刪除在此學習路徑中建立 AWS 的資源

1. 刪除在此學習路徑中建立的任務

   a. 執行此命令以列出 中的任務 AWS 帳戶。

```
aws iot list-jobs
```

命令會傳回 AWS 帳戶 和 中 AWS 區域 看起來像這樣的 AWS IoT 任務清單。

```
{
    "jobs": [
        {
            "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
job-2",
```

```
            "jobId": "hello-world-job-2",
            "targetSelection": "SNAPSHOT",
            "status": "COMPLETED",
            "createdAt": "2021-11-16T23:40:36.825000+00:00",
            "lastUpdatedAt": "2021-11-16T23:40:41.375000+00:00",
            "completedAt": "2021-11-16T23:40:41.375000+00:00"
        },
        {
            "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
job-1",
            "jobId": "hello-world-job-1",
            "targetSelection": "SNAPSHOT",
            "status": "COMPLETED",
            "createdAt": "2021-11-16T23:35:26.381000+00:00",
            "lastUpdatedAt": "2021-11-16T23:35:29.239000+00:00",
            "completedAt": "2021-11-16T23:35:29.239000+00:00"
        }
    ]
}
```

b. 對於您從清單中辨識為在此學習路徑中建立任務的每個任務，請將 *jobId*取代為要刪除的任務jobId值，然後執行此命令來刪除 AWS IoT 任務。

```
aws iot delete-job --job-id jobId
```

如果成功，此命令不會傳回任何內容。

2. 刪除儲存在此學習路徑 Amazon S3 儲存貯體中的任務文件

a. 使用您採用的儲存貯體名稱來取代 *bucket*，然後執行此命令來列出您使用的 Amazon S3 儲存貯體中的物件。

```
aws s3api list-objects --bucket bucket
```

此命令會傳會儲存貯體中的 Amazon S3 物件清單，如下所示。

```
{
    "Contents": [
        {
            "Key": "hello-world-job.json",
            "LastModified": "2021-11-18T03:02:12+00:00",
            "ETag": "\"868c8bc3f56b5787964764d4b18ed5ef\"",
```

```
            "Size": 54,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
"e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "iot_job_firmware_update.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"7c68c591949391791ecf625253658c61\"",
            "Size": 66,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
"e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "order66.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"bca60d5380b88e1a70cc27d321caba72\"",
            "Size": 29,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
"e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        }
    ]
}
```

b. 對於您從清單中辨識為在此學習路徑中建立的物件的每個物件，請使用儲存貯體名稱來取代 *bucket*，使用要刪除物件的鍵值來取代 *key*，然後執行此命令來刪除 Amazon S3 物件。

```
aws s3api delete-object --bucket bucket --key key
```

如果成功，此命令不會傳回任何內容。

刪除完成此學習路徑時建立的所有 AWS 資源和物件後，您可以重新開始並重複教學課程。

使用 AWS IoT 裝置SDKs建置解決方案

本節中的教學課程將逐步引導您開發可使用 部署到生產環境的 IoT 解決方案 AWS IoT。

這些教學課程可能需要比 章節中更多的時間來完成，the section called “使用 AWS IoT 裝置用戶端建置示範”因為它們使用 AWS IoT 裝置SDKs並更詳細地解釋正在套用的概念，以協助您建立安全可靠的解決方案。

使用 AWS IoT 裝置SDKs開始建置解決方案

這些教學課程會逐步解說不同的 AWS IoT 案例。在適當的情況下，教學課程會使用 AWS IoT 裝置 SDKs。

主題

- 教學課程： AWS IoT Core 使用 裝置 SDK 將 AWS IoT 裝置連線至
- 建立 AWS IoT 規則將裝置資料路由到其他 服務
- 在裝置離線時保留裝置狀態
- 教學課程：為 建立自訂授權方 AWS IoT Core
- 教學課程：使用 AWS IoT 和 Raspberry Pi 監控土壤濕度

教學課程： AWS IoT Core 使用 裝置 SDK 將 AWS IoT 裝置連線至

本教學課程示範如何將裝置連線至 ， AWS IoT Core 以便傳送和接收進出的資料 AWS IoT。完成本教學課程後，您的裝置將設定為連線至 ， AWS IoT Core 而且您將了解裝置與 的通訊方式 AWS IoT。

主題

- 先決條件
- 為 準備您的裝置 AWS IoT
- 檢閱 MQTT 通訊協定
- 檢閱 pubsub.py 裝置 SDK 範例應用程式
- 連接您的裝置並與 通訊 AWS IoT Core
- 檢視結果
- 教學課程：使用 適用於 Embedded C 的 AWS IoT Device SDK

先決條件

開始本教學課程之前，請確定您有：

- 已完成 AWS IoT Core 教學課程入門

  在該教學課程中您必須進行 the section called “設定您的裝置” 的部分中，請為您的裝置選取 the section called “連接 Raspberry Pi 或其他裝置” 選項，並使用 Python 語言選項來配置您的裝置。

  > **Note**
  >
  > 讓該教學課程中的終端機視窗處於開啟狀態，因為您還會在本教學課程中用到它。

- 可執行適用於 Python 的 AWS IoT Device SDK v2 的裝置。

  本教學課程說明如何使用 Python AWS IoT Core 程式碼範例將裝置連接至 ，這需要相對強大的裝置。若您使用是資源受限的裝置，這些程式碼範例可能無法在這些裝置上運作。在這種情況下，您可能會在the section called “使用 適用於 Embedded C 的 AWS IoT Device SDK”教學課程中獲得更多成功。

- 取得連線至裝置所需的資訊

  若要將裝置連線至 AWS IoT，您必須擁有物件名稱、主機名稱和連接埠號碼的相關資訊。

  > **Note**
  >
  > 您也可以使用自訂身分驗證來連接裝置 AWS IoT Core。您傳遞給授權方 Lambda 函數的連線資料取決於您使用的通訊協定。

  - 物件名稱：您要連線的 AWS IoT 物件名稱。您必須已將 註冊為裝置做為 AWS IoT 物件。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。
  - 主機名稱：帳戶特定 IoT 端點的主機名稱。
  - 連接埠號碼：要連線的連接埠號碼。

  您可以使用 AWS IoT Python SDK 中的 `configureEndpoint`方法來設定主機名稱和連接埠號碼。

```
myAWSIoTMQTTClient.configureEndpoint("random.iot.region.amazonaws.com", 8883)
```

為 準備您的裝置 AWS IoT

在 AWS IoT Core 教學課程入門 中，您準備好裝置和 AWS 帳戶，使其可進行通訊。本節會檢閱適用於任何裝置連線之準備工作的各個層面 AWS IoT Core。

若為要連線至 AWS IoT Core的裝置：

1. 您必須具有 AWS 帳戶。

   AWS 帳戶 如果您還沒有 ， 中的程序會設定 AWS 帳戶說明如何建立 。
2. 在該帳戶中，您必須為 AWS 帳戶 和 區域中的裝置定義下列AWS IoT 資源。

   程序建立 AWS IoT 資源 中的程序說明如何為您在 AWS 帳戶 和區域中的裝置建立這些資源。
   - 以 AWS IoT 註冊並啟用以驗證裝置的裝置憑證。

     憑證通常是使用 AWS IoT 物件建立並加以連接。雖然裝置不需要物件才能連線 AWS IoT，但它可讓裝置使用其他 AWS IoT 功能。
   - 連接到裝置憑證的政策，授權它連接到 AWS IoT Core 並執行您想要它的所有動作。
3. 可存取您 AWS 帳戶之裝置端點的網際網路連線。

   裝置端點會在 中說明AWS IoT 裝置資料和服務端點，並可在 AWS IoT 主控台的設定頁面中查看。
4. 通訊軟體，例如 AWS IoT 裝置 SDK 提供。本教學課程使用適用於 Python 的AWS IoT 裝置 SDK v2。

檢閱 MQTT 通訊協定

在我們談論範例應用程式之前，其有助於了解 MQTT 通訊協定。MQTT 通訊協定提供了一些優於其他網路通訊協定 (如 HTTP) 的優勢，這使其成為 IoT 裝置的熱門選擇。本節將會檢視 MQTT 適用於本教學課程的主要層面。如需 MQTT 與 HTTP 比較的資訊，請參閱 選擇裝置通訊的應用程式通訊協定。

MQTT 使用一個發佈/訂閱通訊模型。

MQTT 通訊協定會與其主機一起使用發佈/訂閱通訊模型。此模型與 HTTP 使用的請求/回應模型不同。使用 MQTT，裝置會與由唯一用戶端 ID 識別的主機建立工作階段。如要傳送資料，裝置會將主題識別的訊息發佈至主機中的訊息代理程式。為了收到訊息代理程式的訊息，裝置會在訂閱請求中傳送主題篩選條件給訊息代理程式，以訂閱其會收到的主題。

MQTT 支援持久性工作階段

訊息代理程式會收到來自裝置的訊息，並將訊息發佈至已加以訂閱的裝置。利用持久性工作階段 (即使啟動裝置以中斷連線，工作階段仍保持活動狀態)，裝置可擷取中斷連線時所發佈的訊息。在裝置端，MQTT 支援服務品質層級 (QoS)，以確保主機接收裝置所傳送的訊息。

檢閱 pubsub.py 裝置 SDK 範例應用程式

本節會從用於本教學課程中之適用於 Python 的AWS IoT 裝置 SDK v2 檢閱 pubsub.py 範例應用程式。在這裡，我們將檢閱其如何連線至 AWS IoT Core 以發佈和訂閱 MQTT 訊息。下一節提供一些練習，協助您探索裝置連線和通訊的方式 AWS IoT Core。

pubsub.py 範例應用程式示範 MQTT 連線的這些層面 AWS IoT Core：

- 通訊協定
- 持久性工作階段
- 服務品質
- 訊息發佈
- 訊息訂閱
- 裝置中斷連線及重新連線

通訊協定

pubsub.py 範例會示範使用 MQTT 和透過 WSS 的 MQTT 通訊協定的 MQTT 連線。AWS 通用執行時間 (AWS CRT) 程式庫提供低階通訊協定支援，並包含在適用於 Python 的 AWS IoT Device SDK v2 中。

MQTT

中使用 AWS IoT Core MQTT 通訊協定mqtt_connection_builder與 建立連線pubsub.py的範例呼叫 mtls_from_path（如下所示）。 mtls_from_path使用 X.509 憑證和 TLS 1.2 版來驗證裝置。 AWS CRT 程式庫會處理該連線的較低層級詳細資訊。

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.ca_file,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
```

```
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

endpoint

AWS 帳戶您 IoT 裝置端點

於範例應用程式中，此值會從命令列傳入。

cert_filepath

裝置憑證檔案的路徑。

於範例應用程式中，此值會從命令列傳入。

pri_key_filepath

使用其憑證檔案建立的裝置私密金鑰檔案路徑

於範例應用程式中，此值會從命令列傳入。

ca_filepath

根 CA 檔案的路徑。僅當 MQTT 伺服器使用不在您信任庫中的憑證時才需要。

於範例應用程式中，此值會從命令列傳入。

client_bootstrap

處理通訊端通訊活動的通用執行時間物件

在範例應用程式中，此物件就在呼叫 mqtt_connection_builder.mtls_from_path 之前進行實例化。

on_connection_interrupted, on_connection_resumed

裝置連線遭到中斷並回復時呼叫的回呼函數。

client_id

唯一在 AWS 區域中識別此裝置的 ID。

於範例應用程式中，此值會從命令列傳入。

`clean_session`

是否啟動新的持續性工作階段，或者，若存在，則重新連接到現有的工作階段

`keep_alive_secs`

在 CONNECT 請求中傳送的保持活動值 (以秒為單位)。Ping 會在此時間間隔自動傳送。若在此值的 1.5 倍之後未收到 ping，則伺服器會假設連線中斷。

透過 WSS 的 MQTT

pubsub.py 範例呼叫 mqtt_connection_builder 中的 websockets_with_default_aws_signing (如圖所示)，使用透過 WSS 的 MQTT 通訊協定來建立與 AWS IoT Core 的連線。websockets_with_default_aws_signing 會使用 Signature 第 4 版透過 WSS 建立 MQTT 連線，來驗證裝置。

```
mqtt_connection = mqtt_connection_builder.websockets_with_default_aws_signing(
    endpoint=args.endpoint,
    client_bootstrap=client_bootstrap,
    region=args.signing_region,
    credentials_provider=credentials_provider,
    websocket_proxy_options=proxy_options,
    ca_filepath=args.ca_file,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

AWS 帳戶您 IoT 裝置端點

於範例應用程式中，此值會從命令列傳入。

`client_bootstrap`

處理通訊端通訊活動的通用執行時間物件

在範例應用程式中，此物件就在呼叫 mqtt_connection_builder.websockets_with_default_aws_signing 之前進行實例化。

`region`

Signature V4 身分驗證使用的 AWS 簽署區域。於 pubsub.py 中，其會傳遞在命令列中輸入的參數。

於範例應用程式中，此值會從命令列傳入。

`credentials_provider`

提供用於身分驗證的 AWS 登入資料

在範例應用程式中，此物件就在呼叫 `mqtt_connection_builder.websockets_with_default_aws_signing` 之前進行實例化。

`websocket_proxy_options`

HTTP 代理選項 (若使用代理主機)

在範例應用程式中，此值就在呼叫 `mqtt_connection_builder.websockets_with_default_aws_signing` 之前進行初始化。

`ca_filepath`

根 CA 檔案的路徑。僅當 MQTT 伺服器使用不在您信任庫中的憑證時才需要。

於範例應用程式中，此值會從命令列傳入。

`on_connection_interrupted`, `on_connection_resumed`

裝置連線遭到中斷並回復時呼叫的回呼函數。

`client_id`

在 AWS 區域中唯一識別此裝置的 ID。

於範例應用程式中，此值會從命令列傳入。

`clean_session`

是否啟動新的持續性工作階段，或者，若存在，則重新連接到現有的工作階段

`keep_alive_secs`

在 CONNECT 請求中傳送的保持活動值 (以秒為單位)。Ping 會在此時間間隔自動傳送。若在此值的 1.5 倍之後未收到 ping，則伺服器會假設連線中斷。

HTTPS

HTTPS 呢？ AWS IoT Core 支援發佈 HTTPS 請求的裝置。從程式設計的角度來看，裝置會將 HTTPS 請求傳送至 AWS IoT Core ，就像任何其他應用程式一樣。有關從裝置傳送 HTTP 訊息的 Python 程式範例，請參閱使用 Python `requests` 程式庫的 HTTPS 程式碼範例。此範例 AWS IoT Core 使用 HTTPS 傳送訊息至 ， AWS IoT Core 讓 將其解譯為 MQTT 訊息。

雖然 AWS IoT Core 支援來自裝置的 HTTPS 請求，請務必檢閱有關 的資訊，選擇裝置通訊的應用程式通訊協定以便您可以針對裝置通訊所使用的通訊協定做出明智的決策。

持久性工作階段

於範例應用程式中，將 `clean_session` 參數設定為 `False` 表示連線應該是持續的。於實踐中，這意味著透過此呼叫開啟的連線重新連接至現有的持久性工作階段 (如若存在)。否則，其會建立並連線至新的持續工作階段。

使用持續性工作階段，在裝置未連線時，訊息代理程式會儲存傳送至裝置的訊息。當裝置重新連線至持續性工作階段時，訊息代理程式會將所有已訂閱的儲存訊息傳送至裝置。

若無持續性工作階段，裝置將不會收到裝置未連線時傳送的訊息。要使用哪個選項取決於您的應用程式，及是否必須對裝置未連接時產生訊息進行通訊。如需詳細資訊，請參閱MQTT 持久性工作階段。

服務品質

當裝置發佈和訂閱訊息時，可以設定偏好的服務品質 (QoS)。 AWS IoT 支援 QoS 層級 0 和 1 以進行發佈和訂閱操作。如需 中 QoS 層級的詳細資訊 AWS IoT，請參閱 MQTT 服務品質 (QoS) 選項。

Python 的 AWS CRT 執行期會為其支援的 QoS 層級定義這些常數：

Python 服務品質層級

MQTT QoS 層級	SDK 所使用的 Python 符號值	說明
QoS 層級 0	`mqtt.QoS.AT_MOST_ONCE`	無論是否收到訊息，都只會嘗試傳送一次訊息。該訊息可能根本不會傳送，例如，若裝置未連線或網路發生錯誤。
QoS 層級 1	`mqtt.QoS.AT_LEAST_ONCE`	訊息會重複傳送，直到收到 PUBACK 確認為止。

於範例應用程式中，發佈和訂閱請求的 QoS 層級為 1 (mqtt.QoS.AT_LEAST_ONCE)。

- 發佈時的 QoS

  當裝置發佈 QoS 層級 1 的訊息時，其會重複傳送訊息，直到收到來自訊息代理程式的 PUBACK 回應為止。若裝置未連線，訊息會在重新連線後佇列等待傳送。

- 訂閱時的 QoS

  當裝置訂閱 QoS 層級 1 的訊息時，訊息代理程式會儲存裝置訂閱的訊息，直至其可傳送至裝置為止。訊息代理程式會重新傳送訊息，直至收到來自裝置的 PUBACK 回應。

訊息發佈

成功建立與 的連線後 AWS IoT Core，裝置可以發佈訊息。pubsub.py 範例經由呼叫 mqtt_connection 物件的 publish 操作來完成此作業。

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

topic

識別訊息的訊息主題名稱

於範例應用程式中，這會從命令列傳入。

payload

將訊息承載格式化為字串 (例如，JSON 文件)

於範例應用程式中，這會從命令列傳入。

JSON 文件是一種常見的承載格式，而其他 AWS IoT 服務可辨識該文件；不過，訊息承載的資料格式可以是發佈者和訂閱者同意的任何內容。但在某些狀況下，對大多數的作業而言，其他 AWS IoT 服務僅識別 JSON 和 CBOR。

qos

此訊息的 QoS 層級

訊息訂閱

若要接收來自 AWS IoT 和其他 服務和裝置的訊息，裝置會依其主題名稱訂閱這些訊息。裝置可透過指定主題名稱訂閱個別訊息，及透過指定可包含萬用字元字的主題篩選條件訂閱一組訊息。pubsub.py 範例會使用顯示於此處的程式碼來訂閱訊息，並註冊回呼函數，以於收到訊息之後進行處理。

```
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=args.topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received
)
subscribe_result = subscribe_future.result()
```

`topic`

要訂閱的主題。此可為主題名稱或主題篩選條件。

於範例應用程式中，這會從命令列傳入。

`qos`

訊息代理程式是否應該在裝置中斷連線斷時儲存這些訊息。

`mqtt.QoS.AT_LEAST_ONCE` 的值 (QoS 層級 1) 需要在建立連線時指定持久性工作階段 (`clean_session=False`)。

`callback`

呼叫來處理訂閱訊息的函數。

此 `mqtt_connection.subscribe` 函式會傳回一個未來和一個封包 ID。若訂閱請求已順利啟動，傳回的封包 ID 大於 0。如要確定訊息代理程式收到訂閱並已註冊，您必須等待非同步作業的結果傳回，如程式碼範例中所示。

回呼函數

pubsub.py 範例中的回呼會在裝置收到訂閱訊息時進行處理。

```
def on_message_received(topic, payload, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == args.count:
```

```
        received_all_event.set()
```

`topic`

訊息的主題

即使您已訂閱了主題篩選條件，此為所接收訊息的特定主題名稱。

`payload`

訊息承載

此格式為應用程式專用。

`kwargs`

可能的其他引數，如 `mqtt.Connection.subscribe` 中所述。

在 `pubsub.py` 範例中，`on_message_received` 只會顯示主題及其承載。其還會計算在達到限制後，所收到結束程式的訊息。

您的應用程式會評估主題和承載，以決定要執行的動作。

裝置中斷連線及重新連線

`pubsub.py` 範例包含在裝置中斷連線及重新建立連線時呼叫的回呼函數。您的裝置對這些事件採取的動作為應用程式專用。

裝置首次連線時，必須訂閱要接收的主題。若裝置的工作階段在重新連線時存在，則會還原其訂閱，並在重新連線後將來自這些訂閱的任何儲存訊息傳送至裝置。

若裝置的工作階段在重新連線時不復存在，則必須重新訂閱其訂閱。持續性工作階段具有限的存留期，當裝置中斷連線過久時，可能會過期。

連接您的裝置並與 通訊 AWS IoT Core

本節提供一些練習，協助您探索將裝置連線至 AWS IoT Core的不同層面。於這些練習中，您將會使用 AWS IoT 主控台中的 MQTT 測試用戶端，查看裝置發佈的內容，及將訊息發佈至裝置。這些練習使用來自適用於 Python 的AWS IoT 裝置 SDK v2 的 `pubsub.py` 範例，並以您 入門教學課程 教學課程經驗為基礎。

在本節中，您會：

- 訂閱萬用字元主題篩選條件

- 處理主題篩選條件訂閱
- 從您的裝置發佈訊息

對於這些練習，您會從 pubsub.py 範例程式開始。

> **Note**
>
> 這些練習假設您已完成 入門教學課程 教學課程，並使用該教學課程中的裝置終端機視窗。

訂閱萬用字元主題篩選條件

在本練習中，您會修改用來呼叫 pubsub.py 以訂閱萬用字元主題篩選條件並根據訊息主題處理所接收訊息的命令列。

演練程序

在此練習中，想像您的裝置包含溫度控制和光源控制。其會使用這些主題名稱來識別與其相關的訊息。

1. 開始練習之前，請試著從裝置上的 入門教學課程 教學課程執行此命令，確保所有項目都準備好進行練習。

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

   您應該會看到與在教學課程入門中看到的相同輸出。
2. 在此練習中，請變更這些命令列參數。

Action	命令列參數	Effect
add	--message ""	僅限配置 pubsub.py 加以聆聽
add	--count 2	收到兩則訊息後結束程式
change	--topic device/+/details	定義要訂閱的主題篩選條件

對初始命令列進行這些變更會產生此命令列。在裝置的終端機視窗中輸入此命令。

```
python3 pubsub.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

程式應該會顯示如下所示的內容：

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-24d7cdcc-cc01-458c-8488-2d05849691e1'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
```

若您在終端機上看到類似的內容，表示您的裝置已準備就緒，且會收聽主題名稱開頭為 `device`，結尾為 `/detail` 的訊息。所以，讓我們來進行測試。

3. 以下為您的裝置可能會收到的幾個訊息。

主題名稱	訊息承載
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

4. 在 AWS IoT 主控台中使用 MQTT 測試用戶端，將上一個步驟中所述的訊息傳送到您的裝置。

   a. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。

   b. 於 Subscribe to a topic (訂閱主題) 的 Subscription topic (訂閱主題) 欄位中，輸入主題篩選條件：**`device/+/details`**，再選擇 Subscribe to topic (訂閱主題)。

   c. 在 MQTT 測試用戶端的 Subscriptions (訂閱) 欄中，選擇 device/+/details。

   d. 若為上表中的每個主題，請於 MQTT 測試用戶端中執行下列動作：

      1. 於 Publish (發佈) 中，輸入來自表格中 Topic name (主題名稱) 欄的值。

2. 在主題名稱下方的訊息承載欄位中，輸入來自表格中 Message payload (訊息承載) 欄的值。
3. 觀看 pubsub.py 正在執行的終端機視窗，並在 MQTT 測試用戶端中，選擇 Publish to topic (發佈至主題)。

您應該看到該訊息是由終端機視窗中的 pubsub.py 所收到。

演練結果

有了此 pubsub.py，使用萬用字元主題篩選條件訂閱訊息，加以訊息，並將其顯示於終端機視窗中。請注意您如何訂閱單一主題篩選條件，並呼叫回呼函數來處理具有兩個不同主題的訊息。

處理主題篩選條件訂閱

建置於上一個練習上，修改 pubsub.py 範例應用程式來評估訊息主題，並根據主題處理訂閱的訊息。

演練程序

如要評估訊息主題

1. 將 pubsub.py 複製至 pubsub2.py。
2. 以您最愛的文字編輯器或 IDE 中開啟 pubsub2.py 檔案。
3. 在 pubsub2.py 中，尋找 on_message_received 函數。
4. 在 on_message_received 中，請在以 print("Received message 開頭的行之後及以 global received_count 開頭的行之前插入下列程式碼。

```
    topic_parsed = False
    if "/" in topic:
        parsed_topic = topic.split("/")
        if len(parsed_topic) == 3:
            # this topic has the correct format
            if (parsed_topic[0] == 'device') and (parsed_topic[2] == 'details'):
                # this is a topic we care about, so check the 2nd element
                if (parsed_topic[1] == 'temp'):
                    print("Received temperature request: {}".format(payload))
                    topic_parsed = True
                if (parsed_topic[1] == 'light'):
                    print("Received light request: {}".format(payload))
                    topic_parsed = True
```

```
    if not topic_parsed:
        print("Unrecognized message topic.")
```

5. 儲存您的變更，並使用此命令列執行修改後的程式。

```
python3 pubsub2.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

6. 在 AWS IoT 主控台中，開啟 MQTT 測試用戶端。
7. 於 Subscribe to a topic (訂閱主題) 的 Subscription topic (訂閱主題) 欄位中，輸入主題篩選條件：**device/+/details**，再選擇 Subscribe to topic (訂閱主題)。
8. 在 MQTT 測試用戶端的 Subscriptions (訂閱) 欄中，選擇 device/+/details。
9. 若為此表格中的每個主題，請在 MQTT 測試用戶端中執行下列動作：

主題名稱	訊息承載
device/temp/details	{ "desiredTemp": 20, "currentTemp": 15 }
device/light/details	{ "desiredLight": 100, "currentLight": 50 }

   1. 於 Publish (發佈) 中，輸入來自表格中 Topic name (主題名稱) 欄的值。
   2. 在主題名稱下方的訊息承載欄位中，輸入來自表格中 Message payload (訊息承載) 欄的值。
   3. 觀看 pubsub.py 正在執行的終端機視窗，並在 MQTT 測試用戶端中，選擇 Publish to topic (發佈至主題)。

   您應該看到該訊息是由終端機視窗中的 pubsub.py 所收到。

您應該會在終端機視窗中看到類似的內容。

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID 'test-
af794be0-7542-45a0-b0af-0b0ea7474517'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
```

```
Waiting for all messages to be received...
Received message from topic 'device/light/details': b'{ "desiredLight": 100,
 "currentLight": 50 }'
Received light request: b'{ "desiredLight": 100, "currentLight": 50 }'
Received message from topic 'device/temp/details': b'{ "desiredTemp": 20,
 "currentTemp": 15 }'
Received temperature request: b'{ "desiredTemp": 20, "currentTemp": 15 }'
2 message(s) received.
Disconnecting...
Disconnected!
```

演練結果

在本練習中，您已新增程式碼，讓範例應用程式能辨識並處理回呼函數中的多個訊息。有了這個，您的裝置可以接收訊息並對其採取行動。

裝置接收和處理多則訊息的另一種方式是分別訂閱不同訊息，並將每個訂閱分配給各自的回呼函數。

從您的裝置發佈訊息

您可使用 pubsub.py 範例應用程式，以發佈來自您裝置的訊息。在其會發佈訊息時，這些訊息無法作為 JSON 文件加以讀取。本練習會修改範例應用程式，以便在可讀取的訊息承載中發佈 JSON 文件 AWS IoT Core。

演練程序

在本練習中，下列訊息將會與 `device/data` 主題一起傳送。

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
    ]
}
```

如要準備 MQTT 測試用戶端監控本練習的訊息

1. 於 Subscribe to a topic (訂閱主題) 的 Subscription topic (訂閱主題) 欄位中，輸入主題篩選條件：**`device/data`**，再選擇 Subscribe to topic (訂閱主題)。

2. 在 MQTT 測試用戶端的 Subscriptions (訂閱) 欄中，選擇 device/data。
3. 讓 MQTT 測試用戶端視窗保持開啟狀態，以等待來自您裝置的訊息。

如要使用 pubsub.py 範例應用程式傳送 JSON 文件

1. 在您的裝置上，將 pubsub.py 複製至 pubsub3.py。
2. Edit (編輯) pubsub3.py，變更其格式化所發佈訊息的方式。
   a. 在文字編輯器中開啟 pubsub3.py。
   b. 找出這行程式碼：

      ```
      message = "{} [{}]".format(message_string, publish_count)
      ```

   c. 將其變更為：

      ```
      message = "{}".format(message_string)
      ```

   d. 找出這行程式碼：

      ```
      message_json = json.dumps(message)
      ```

   e. 將其變更為：

      ```
      message = "{}".json.dumps(json.loads(message))
      ```

   f. 儲存您的變更。
3. 在您的裝置上，執行此命令以傳送訊息兩次。

```
python3 pubsub3.py  --ca_file ~/certs/Amazon-root-CA-1.pem  --cert ~/certs/
device.pem.crt  --key ~/certs/private.pem.key  --topic device/data  --count 2 --
message '{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'  --endpoint your-iot-endpoint
```

4. 於 MQTT 測試用戶端中，檢查是否已解譯並格式化訊息承載中的 JSON 文件，例如：



依預設，pubsub3.py 也會訂閱其傳送的訊息。您您應會看到其在應用程式輸出中收到訊息。終端機視窗看起來與下列內容相似。

```
Connecting to a3qEXAMPLEsffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-5cff18ae-1e92-4c38-a9d4-7b9771afc52f'...
Connected!
Subscribing to topic 'device/data'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 2 message(s)
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
2 message(s) received.
Disconnecting...
Disconnected!
```

演練結果

如此一來，您的裝置可以產生要傳送至 的訊息 AWS IoT Core ，以測試基本連線，並提供裝置訊息 AWS IoT Core 供 處理。例如，您可以使用此應用程式從您的裝置傳送測試資料，以測試 AWS IoT 規則動作。

檢視結果

本教學中的範例為您提供了有關裝置如何與 通訊的基本知識的實作體驗 AWS IoT Core，這是您 AWS IoT 解決方案的基本部分。當您的裝置能夠與 通訊時 AWS IoT Core，他們可以將訊息傳遞給 AWS 服務和其他可以採取行動的裝置。同樣地， AWS 服務和其他裝置可以處理導致訊息傳回裝置的資訊。

當您準備好 AWS IoT Core 進一步探索時，請嘗試這些教學課程：

- the section called “正在傳送 Amazon SNS 通知”
- the section called “將裝置資料儲存於 DynamoDB 表格中”
- the section called “使用 AWS Lambda 函數格式化通知”

教學課程：使用 適用於 Embedded C 的 AWS IoT Device SDK

本節說明如何執行 適用於 Embedded C 的 AWS IoT Device SDK。

本節中的程序

- Step1：安裝 適用於 Embedded C 的 AWS IoT Device SDK
- 步驟 2：設定範例應用程式
- 步驟 3：建置並執行範例應用程式

Step1：安裝 適用於 Embedded C 的 AWS IoT Device SDK

通常以需要最佳化 C 語言執行時間的資源限制裝置 適用於 Embedded C 的 AWS IoT Device SDK 為目標。您可以在任何作業系統上使用軟體開發套件，並將其裝載在任何處理器類型 (例如 MCU 和 MPU) 上。如果您有更多可用的記憶體和處理資源，建議您使用較高階 AWS IoT 的裝置和行動 SDKs（ 例如 C++、Java、JavaScript 和 Python)。

一般而言， 適用於 Embedded C 的 AWS IoT Device SDK 適用於使用 MCUs的系統，或執行內嵌作業系統的低階 MPUs。在本節中的程式設計範例，我們假設您的裝置使用 Linux。

Example

1. 從 GitHub 將 下載 適用於 Embedded C 的 AWS IoT Device SDK 至您的裝置。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-submodules
   ```

   這會在目前目錄中建立名為 `aws-iot-device-sdk-embedded-c` 的目錄。

2. 導航至該目錄並查看最新版本。請參閱 github.com/aws/aws-iot-device-sdk-embedded-C/tags，取得最新版本標籤。

   ```
   cd aws-iot-device-sdk-embedded-c
   git checkout latest-release-tag
   ```

3. 安裝 OpenSSL 1.1.0 版或更新版本。當透過套件管理工具進行安裝時，OpenSSL 開發程式庫通常會稱為 "libssl-dev" 或 "openssl-devel"。

   ```
   sudo apt-get install libssl-dev
   ```

步驟 2：設定範例應用程式

適用於 Embedded C 的 AWS IoT Device SDK 包含供您嘗試的範例應用程式。為了簡化，本教學課程使用 `mqtt_demo_mutual_auth` 應用程式，說明如何連線至 AWS IoT Core 訊息中介裝置，以及訂閱和發佈至 MQTT 主題。

1. 將您建立於 AWS IoT Core 教學課程入門 中的憑證和私有金鑰複製至 `build/bin/certificates` 目錄中。

   > **Note**
   >
   > 裝置和根憑證授權機構憑證會有過期或遭撤銷的可能。若這些憑證過期或遭到撤銷，您必須將新的憑證授權機構憑證或私有金鑰和裝置憑證複製到裝置上。

2. 您必須使用個人 AWS IoT Core 端點、私有金鑰、憑證和根 CA 憑證來設定範例。導覽至 `aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth` 目錄。

   如果您 AWS CLI 已安裝 ，您可以使用此命令來尋找帳戶的端點 URL。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果您沒有 AWS CLI 安裝 ，請開啟您的 AWS IoT 主控台。在導覽窗格中，選擇 Manage (管理)，然後選擇 Things (物件)。為您的裝置選擇 IoT 物件，然後選擇 Interact (互動)。您的端點會顯示在物件詳細資料頁面的 HTTPS 區段中。

3. 開啟 demo_config.h 檔案，並更新下列內容的值：

   AWS_IOT_ENDPOINT

   您的個人端點。

   CLIENT_CERT_PATH

   您的憑證檔案路徑，例如 certificates/device.pem.crt"。

   CLIENT_PRIVATE_KEY_PATH

   您的私有金鑰檔案名稱，例如certificates/private.pem.key。

   例如：

```
// Get from demo_config.h
// ================================================
#define AWS_IOT_ENDPOINT                "my-endpoint-ats.iot.us-
east-1.amazonaws.com"
#define AWS_MQTT_PORT                   8883
#define CLIENT_IDENTIFIER               "testclient"
#define ROOT_CA_CERT_PATH               "certificates/AmazonRootCA1.crt"
#define CLIENT_CERT_PATH                "certificates/my-device-cert.pem.crt"
#define CLIENT_PRIVATE_KEY_PATH         "certificates/my-device-private-key.pem.key"
// ================================================
```

4. 使用此命令來檢查您的裝置是否已安裝 CMake。

```
cmake --version
```

若您看見編譯器的版本資訊，則可繼續到下一節。

若發生錯誤，或者沒看見資任何訊，您將需要使用此命令來安裝 cmake 套件。

```
sudo apt-get install cmake
```

再次執行 cmake --version 命令，確認已安裝 CMake，備妥繼續。

5. 使用此命令來檢查您的裝置是否已安裝開發工具。

```
gcc --version
```

若您看見編譯器的版本資訊，則可繼續到下一節。

如果發生錯誤，或者沒看見編譯器資訊，您將需要使用此命令來安裝 build-essential 套件。

```
sudo apt-get install build-essential
```

再次執行 gcc --version 命令，確認已安裝建置工具，準備好繼續。

步驟 3：建置並執行範例應用程式

此程序說明如何在裝置上產生mqtt_demo_mutual_auth應用程式，並使用 將其連接至AWS IoT 主控台 適用於 Embedded C 的 AWS IoT Device SDK。

執行 適用於 Embedded C 的 AWS IoT Device SDK 範例應用程式

1. 導覽至 aws-iot-device-sdk-embedded-c，建立一個建置目錄。

```
mkdir build && cd build
```

2. 輸入下列 CMake 命令，產生需要建置的 Makefile。

```
cmake ..
```

3. 輸入下列命令來建置可執行的應用程式檔案。

```
make
```

4. 使用此命令執行 mqtt_demo_mutual_auth 應用程式。

```
cd bin
./mqtt_demo_mutual_auth
```

您應該會看到類似下列的輸出：

```
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:584] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8883.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1264] Creating an MQTT connection to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com.
[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt_serializer.c:970] CONNACK session present bit not set.
[INFO] [MQTT] [core_mqtt_serializer.c:912] Connection accepted.
[INFO] [MQTT] [core_mqtt.c:1526] Received MQTT CONNACK successfully from broker.
[INFO] [MQTT] [core_mqtt.c:1792] MQTT connection established with the broker.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1033] MQTT connection successfully established with broker.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1296] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1314] Subscribing to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1097] SUBSCRIBE sent for topic testclient/example/topic to broker.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:921] Subscribed to the topic testclient/example/topic. with maximum QoS 1.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1358] Sending Publish to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1195] PUBLISH sent for topic testclient/example/topic to broker with packet ID 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt.c:1126] Ack packet deserialized with result: MQTTSuccess.
[INFO] [MQTT] [core_mqtt.c:1139] State record updated. New state=MQTTPublishDone.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:946] PUBACK received for packet id 2.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:672] Cleaned up outgoing publish packet with packet id 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=40.
[INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
```

您的裝置現在已 AWS IoT 使用 連線至 適用於 Embedded C 的 AWS IoT Device SDK。

您也可以使用 AWS IoT 主控台來檢視範例應用程式正在發佈的 MQTT 訊息。如需關於如何在 AWS IoT 主控台中使用 MQTT 用戶端的資訊，請參閱 the section called “使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息”。

建立 AWS IoT 規則將裝置資料路由到其他 服務

這些教學課程說明如何使用一些較常見的 AWS IoT 規則動作來建立和測試規則。

AWS IoT 規則會將資料從您的裝置傳送至其他 AWS 服務。其會聆聽特定的 MQTT 訊息、格式化訊息承載中的資料，並將結果傳送至其他 AWS 服務。

即使目標是要建立使用 Lambda 函數或更複雜的規則，我們仍建議您依照此處顯示的順序進行嘗試。教學課程會依照基本到複雜的順序顯示。其會逐步提供新概念，協助您學習可用來建立沒有特定教學課程的規則動作概念。

> **Note**
>
> AWS IoT 規則可協助您將資料從 IoT 裝置傳送至其他 AWS 服務。不過，如要成功地達到這一點，您需具備要傳送資料的其他服務的相關工作知識。雖然這些教學課程提供了完成任務所需的資訊，但您可能會發現，在解決方案中使用這些服務之前，進一步了解將資料傳送至服務的相關資訊可能更有幫助。其他服務的詳細說明 AWS 超出這些教學課程的範圍。

教學課程方案概觀

這些教學課程的方案為定期發佈其資料的天氣感應器裝置。在此虛構的系統中有很多這樣的感應器裝置。不過，本節中的教學課程會聚焦在單一裝置上，同時展現容納多個感應器的方法。

本節中的教學課程說明如何使用 AWS IoT 規則，透過這個假想的天氣感應器裝置系統來執行下列任務。

- 教學課程：重新發佈 MQTT 訊息

  本教學課程將會顯示如何將氣象感應器所接收的 MQTT 訊息重新發佈為僅包含感應器 ID 和溫度值的訊息。其僅使用 AWS IoT Core 服務，並展示簡單的 SQL 查詢，及如何使用 MQTT 用戶端來測試您的規則。
- 教學課程：傳送 Amazon SNS 通知

  本教學課程會顯示天氣感應器裝置的值超出特定值時，如何傳送 SNS 訊息。它以先前教學課程中呈現的概念為基礎，並新增了如何使用其他服務 AWS ：Amazon Simple Notification Service (Amazon SNS)。

  若您初次使用 Amazon SNS，請檢閱其入門練習，然後再開始本教學課程。
- 教學課程：將裝置資料儲存在 DynamoDB 表格中

  本教學課程會顯示如何從天氣感應器裝置存放資料庫表中的資料。其會使用規則查詢陳述式和替代範本來格式化目的地服務Amazon DynamoDB 的訊息資料。

  若您初次使用 DynamoDB，請檢閱其入門練習，然後再開始本教學課程。
- 教學課程：使用 AWS Lambda 函數格式化通知

  本教學課程顯示如何呼叫 Lambda 函數來重新格式化裝置資料，然後以文字訊息進行傳送。它在AWS Lambda函數中新增 Python 指令碼和 AWS SDK 函數，以格式化來自天氣感應器裝置的訊息承載資料，並傳送文字訊息。

若您初次使用 Lambda，請檢閱其入門練習，然後再開始本教學課程。

AWS IoT 規則概觀

所有這些教學課程都會建立 AWS IoT 規則。

若要讓 AWS IoT 規則將資料從裝置傳送至其他服務 AWS ，它會使用：

- 包含下列項目的規則查詢陳述式：
  - SQL SELECT 子句，用於選取並格式化訊息承載中的資料
  - 主題篩選條件 (規則查詢陳述式中的 FROM 物件)，用來識別要使用的訊息
  - 選用性的條件陳述式 (SQL WHERE 子句)，指定要採取行動的特定條件
- 至少一個規則動作

裝置會發佈訊息至 MQTT 主題。SQL SELECT 陳述式中的主題篩選條件會識別要套用規則的 MQTT 主題。SQL SELECT 陳述式中指定的欄位會格式化傳入 MQTT 訊息承載的資料，以供規則動作使用。如需規則動作的完整清單，請參閱 AWS IoT 規則動作。

本節中的教學課程

- 教學課程：重新發佈 MQTT 訊息
- 教學課程：傳送 Amazon SNS 通知
- 教學課程：將裝置資料儲存在 DynamoDB 表格中
- 教學課程：使用 AWS Lambda 函數格式化通知

教學課程：重新發佈 MQTT 訊息

本教學課程示範如何建立 AWS IoT 規則，在接收到指定的 MQTT 訊息時發佈 MQTT 訊息。傳入的訊息承載可在發佈前由規則進行修改。這樣就可以建立專為特定應用程式量身訂製的訊息，而不需要更改裝置或其韌體。您還可使用規則的篩選條件面向，僅在符合特定條件時才發佈訊息。

規則重新發佈的訊息就像由任何其他 AWS IoT 裝置或用戶端傳送的訊息。裝置可訂閱重新發佈的訊息，就像裝置可訂閱任何其他 MQTT 訊息主題一樣。

您會在本教學課程中學到什麼：

- 如何在規則查詢陳述式中使用簡單的 SQL 查詢和函數

- 如何使用 MQTT 用戶端測試 AWS IoT 規則

此教學課程約需 30 分鐘方能完成。

於本教學課程中，您將會：

- 檢閱 MQTT 主題和 AWS IoT 規則
- 步驟 1：建立 AWS IoT 規則以重新發佈 MQTT 訊息
- 步驟 2：測試新規則
- 步驟 3：檢閱結果及後續步驟

開始本教學課程之前，請確定您有：

- 設定 AWS 帳戶

  您需要 AWS 帳戶 和 AWS IoT 主控台才能完成本教學課程。
- 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

  請確定您可使用 MQTT 用戶端來訂閱並發佈至主題。您會使用 MQTT 用戶端，在此程序中測試您的新規則。

檢閱 MQTT 主題和 AWS IoT 規則

在討論 AWS IoT 規則之前，了解 MQTT 通訊協定會有幫助。在 IoT 解決方案中，MQTT 通訊協定提供了一些優於其他網路通訊協定 (如 HTTP) 的優勢，這使其成為 IoT 裝置使用的熱門選擇。本節將會檢視 MQTT 適用於本教學課程的主要層面。如需 MQTT 與 HTTP 比較的資訊，請參閱 選擇裝置通訊的應用程式通訊協定。

MQTT 通訊協定

MQTT 通訊協定會與其主機一起使用發佈/訂閱通訊模型。若要傳送資料，裝置會將主題識別的訊息發佈至 AWS IoT 訊息中介裝置。若要收到訊息代理程式的訊息，裝置會在訂閱請求中傳送主題篩選條件給訊息代理程式，以訂閱其會收到的主題。 AWS IoT 規則引擎會收到來自訊息代理程式的 MQTT 訊息。

AWS IoT 規則

AWS IoT 規則包含規則查詢陳述式和一或多個規則動作。當 AWS IoT 規則引擎收到 MQTT 訊息時，這些元素對訊息產生的作用如下。

- 規則查詢陳述式

  規則的查詢陳述式說明要使用的 MQTT 主題、解譯來自訊息承載的資料，及依據 SQL 陳述式所描述的 SQL 陳述式來格式化資料，這些陳述式類似於常用 SQL 資料庫所使用的陳述式。查詢陳述式的結果為傳送至規則動作的資料。

- 規則動作

  規則中的每個規則動作都會作用於規則查詢陳述式所產生的資料。 AWS IoT 支援許多規則動作。不過，在本教學課程中，您會專注於 重新發佈 規則動作，其會將查詢陳述式的結果發佈為具有特定主題的 MQTT 訊息。

步驟 1：建立 AWS IoT 規則以重新發佈 MQTT 訊息

您將在本教學課程中建立的 AWS IoT 規則會訂閱 device/*device_id*/data MQTT 主題，其中 *device_id* 是傳送訊息的裝置 ID。主題篩選條件會將這些主題描述為 device/+/data，其中+ 為與兩個正斜線字元間之任何字串相符的萬用字元。

規則收到來自相符主題的訊息時，其會重新發佈 device_id 和 temperature 值，作為具 device/data/temp 主題的新 MQTT 訊息。

例如，具 device/22/data 主題的 MQTT 訊息承載如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

該規則採用來自訊息承載的 temperature 值及來自主題的 device_id，並將其重新發佈為具 device/data/temp 主題的 MQTT 訊息，及如下所示的訊息承載：

```
{
  "device_id": "22",
  "temperature": 28
}
```

使用此規則，則只需要裝置 ID 和溫度資料的裝置會訂閱 `device/data/temp` 主題，以僅接收該資訊。

如要建立一個重新發佈 MQTT 訊息的規則

1. 開啟 AWS IoT 主控台的規則中樞。
2. 在 Rules (規則) 中，選擇Create(建立)，然後開始建立新規則。
3. 在 Create a rule (建立規則) 的頂部中：
   a. 在 Name (名稱) 中，輸入規則的名稱。在本教學課程中，請將其命名為 **`republish_temp`**。

      請記住，規則名稱在您的帳戶和區域內必須是唯一的，且不能有任何空格。我們在此名稱中使用底線字元來分隔規則名稱中的兩個單字。
   b. 在 Description (說明) 中，說明規則。

      有意義的說明可幫助您記住此規則的作用及建立規則的原因。說明可依所需而定，因此請盡可能詳細說明。
4. 在 Create a rule (建立規則) 的 Rule query statement (規則查詢陳述式) 中：
   a. 在使用 SQL 版本中，選取 **`2016-03-23`**。
   b. 在 Rule query statement (規則查詢陳述式) 編輯方塊中輸入陳述式：

      ```
      SELECT topic(2) as device_id, temperature FROM 'device/+/data'
      ```

      本陳述式：

      - 聆聽具與 `device/+/data` 主題篩選條件相符之主題的 MQTT 訊息。
      - 從主題字串中選取第二個元素，並將其指定給 `device_id` 欄位。
      - 從訊息承載選取值 `temperature` 欄位，並將其指派給 `temperature` 欄位。
5. 在 Set one or more actions (設定一個或多個動作) 中：
   a. 若要開啟此規則的規則動作清單，請選擇 Add action (新增動作)。
   b. 在選取動作中，選擇重新發佈訊息至 AWS IoT 主題。
   c. 在動作清單底部，選擇設定動作以開啟所選動作的組態頁面。
6. 在 Configure action (設定動作)：
   a. 在 Topic (主題) 中，輸入 **`device/data/temp`**。這是此規則將發佈訊息的 MQTT 主題。

b. 在Quality of Service (服務品質) 中，選擇 0 - The message is delivered zero or more times (0 - 訊息傳遞零次或多次)。

c. 在選擇或建立角色以授予執行此動作的 AWS IoT 存取權：

i. 選擇 Create Role (建立角色)。Create a new role (建立新角色) 對話方塊隨即開啟。

ii. 輸入可說明新角色的名稱。在本教學課程中，使用 **`republish_role`**。

當您建立新角色時，會建立執行規則動作的正確政策，並將其連接至新角色。如果您變更此規則動作的主題，或在其他規則動作中使用此角色，則必須更新該角色的政策，以授權新的主題或動作。若要更新現有角色，請選擇本節中的 Update role (更新角色)。

iii. 選擇Create Role (建立角色)，以建立角色並關閉對話方塊。

d. 選擇Add action (新增動作)，將動作新增至規則，並返回Create a rule (建立規則) 頁面。

7. 將訊息重新發佈至 AWS IoT 主題動作現在列在設定一或多個動作中。

在新動作的圖標中，Republish a message to an AWS IoT topic (將訊息重新發佈至 IoT 主題) 之下，您可看到重新發佈動作將發佈的主題。

這是您將新增至此規則的唯一規則動作。

8. 在 Create a rule (建立規則) 中，向下捲動至底部，然後選擇 Create rule (建立規則)，建立規則並完成此步驟。

步驟 2：測試新規則

若要測試新規則，您將使用 MQTT 用戶端來發佈和訂閱此規則所使用的 MQTT 訊息。

在新視窗的 AWS IoT 主控台中開啟 MQTT 用戶端。這可讓您編輯規則，而不會遺失 MQTT 用戶端的組態。如果您讓它轉到主控台中的另一個頁面，MQTT 用戶端不會保留任何訂閱或訊息記錄。

如要使用 MQTT 用戶端來測試您的規則。

1. 在 AWS IoT 主控台的 MQTT 用戶端 中，訂閱輸入主題，在此案例中為 `device/+/data`。

a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Subscribe to a topic (訂閱主題)。

b. 在 Subscription topic (訂閱主題) 中，輸入輸入主題篩選條件 **`device/+/data`** 的主題 。

c. 將剩下的欄位保留為其預設設定。

d. 請選擇 Subscribe to topic (訂閱主題)。

在 Subscriptions (訂閱) 欄中，Publish to a topic (發佈到主題) 之下，**device/+/data** 隨即顯示。

2. 訂閱規則將發佈的主題：device/data/temp。

   a. 在 Subscriptions (訂閱) 之下，選擇 Subscribe to a topic (訂閱主題)，並在 Subscription topic (訂閱主題) 中，輸入重新發佈訊息 **device/data/temp** 的主題。

   b. 將剩下的欄位保留為其預設設定。

   c. 請選擇 Subscribe to topic (訂閱主題)。

      在 Subscriptions (訂閱) 欄中，device/+/data 之下，**device/data/temp** 隨即顯示。

3. 使用特定裝置 ID **device/22/data**，將訊息發佈至輸入主題。您無法發佈至包含萬用字元的 MQTT 主題。

   a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Publish to topic (發佈至主題)。

   b. 在 Publish (發佈) 欄位中輸入輸入主題名稱 **device/22/data**。

   c. 複製此處顯示的範例資料，並在主題名稱下方的編輯方塊中貼上範例資料。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. 若要傳送 MQTT 訊息，請選擇 Publish to topic (發佈至主題)。

4. 檢閱傳送的訊息。

   a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下，您先前訂閱的兩個主題旁會有一個綠點。

      該綠點表示自上次查看後，已收到一個或多個新訊息。

   b. 於 Subscriptions (訂閱) 下，選擇 device/+/data，來檢查訊息承載是否與您剛剛發佈的內容相符，如下所示：

```
{
  "temperature": 28,
```

```
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. 於 Subscriptions (訂閱) 下，選擇 device/data/temp，來檢查您重新發佈的訊息承載，如下所示：

```
{
  "device_id": "22",
  "temperature": 28
}
```

請注意，device_id 值是個帶有引號的字串，而 temperature 值是個數字。這是因為 topic() 函數從輸入訊息的主題名稱中提取字串，而 temperature 值會使用輸入訊息承載的數值。

若要將 device_id 值設為數值，請將規則查詢陳述式中的 topic(2) 替換為：

```
cast(topic(2) AS DECIMAL)
```

請注意，將 topic(2) 值轉換為數值，僅適用於主題的該部分僅包含數字字元時。

5. 若您看到正確的訊息已發佈至 device/data/temp 主題，則您的規則有效。請參閱下一節，了解更多 Republish (重新發佈) 規則動作的相關資訊。

   若您看不到正確的訊息已發佈至 device/+/data 或 device/data/temp 主題中，請查看疑難排解提示。

疑難排解重新發佈訊息規則

若您並未看到預期的結果，請查看以下事項。

- 您收到錯誤的橫幅

  若在您發佈輸入訊息時出現錯誤，請先更正該錯誤。下列步驟可協助您修正該錯誤。

- 您並未在 MQTT 用戶端中看到輸入訊息

每次您將輸入訊息發佈至 device/22/data 主題時，若您依程序中所述訂閱了 device/+/data 主題篩選條件，則該訊息應會顯示於 MQTT 用戶端中。

要檢查的事項

- 檢查您訂閱的主題篩選條件

  若您依程序中所述訂閱了輸入訊息主題，則每次發佈輸入訊息時都應該會看到其複本。

  若您並未訊息，請檢查您訂閱的主題名稱，並將其與所發佈的主題進行比較。主題名稱區分大小寫，且您訂閱的主題必須與所發佈訊息承載的主題相同。

- 檢查訊息發佈功能

  在 MQTT 用戶端中的 Subscriptions (訂閱) 下，選擇 device/+/data，檢查發佈訊息的主題，然後選擇 Publish to topic (發佈至主題)。您應該會在訊息清單中出現主題下方的編輯方塊中看到訊息承載。

- 您在 MQTT 用戶端中並未看到已重新發佈的訊息

  若要讓您的規則運作，其必須具有授權其接收和重新發佈訊息的正確政策，且必須接收訊息。

  要檢查的事項

  - 檢查 MQTT 用戶端 AWS 區域 的 和您建立的規則

    您正在執行 MQTT 用戶端的主控台必須與您建立的規則處於相同的 AWS 區域。

  - 檢查規則查詢陳述式中的輸入訊息主題

    若要讓規則運作，其必須收到一則訊息，其主題名稱與規則查詢陳述式之 FROM 子句中的主題篩選條件相符。

    檢查規則查詢陳述式中主題篩選條件的拼字與 MQTT 用戶端中主題的拼字。主題名稱區分大小寫，且郵件的主題必須與規則查詢陳述式中的主題篩選條件相符。

  - 檢查輸入訊息承載的內容

    若要讓規則運作，其必須在 SELECT 陳述式中宣告的訊息承載中尋找資料欄位。

    檢查規則查詢陳述式中 temperature 欄位的拼字與 MQTT 用戶端中訊息承載的拼字。欄位名稱區分大小寫，規則查詢陳述式中的 temperature 欄位必須與訊息承載中的 temperature 欄位相符。

請確定訊息承載中的 JSON 文件格式正確。若 JSON 有任何錯誤，例如缺少逗號，則規則將無法進行讀取。

- 檢查規則動作中重新發佈的訊息主題

  Republish (重新發佈) 規則動作發佈新訊息的主題必須與您在 MQTT 用戶端中訂閱的主題相符。

  開啟您建立於主控台中的規則，並檢查規則動作重新發佈訊息的主題。

- 檢查規則所使用的角色

  規則動作必須具有接收原始主題及發佈新主題的權限。

  授權規則接收訊息資料的政策並加以重新發佈為所使用的主題所特定的。若變更用於重新發佈訊息資料的主題，則必須更新規則動作的角色，來更新其政策以與目前主題相符。

  若您懷疑這會是問題，請編輯 Republish (重新發佈) 規則動作並建立新角色。規則動作建立的新角色會收到執行這些動作所需的授權。

步驟 3：檢閱結果及後續步驟

於本教學課程中

- 您使用了簡單的 SQL 查詢和規則查詢陳述式中的幾個函數來產生一個新的 MQTT 訊息。
- 您已建立一個重新發佈該新訊息的規則。
- 您使用 MQTT 用戶端來測試 AWS IoT 規則。

後續步驟

使用此規則重新發佈一些訊息之後，請嘗試使用其來查看教學課程的某些層面如何影響重新發佈的訊息。此處有幾種簡單的入門方式。

- 變更輸入訊息主題中的 *device_id*，並觀察重新發佈之訊息承載中的影響。
- 變更規則查詢陳述式中所選取的欄位，並觀察重新發佈之訊息承載中的影響。
- 請嘗試本系列中的下一個教學課程，並了解如何進行 教學課程：傳送 Amazon SNS 通知。

此教學課程中所使用的 Republish (重新發佈) 規則動作亦可協助您對規則查詢陳述式進行偵錯。例如，您可將此動作新增至規則，以查看其規則查詢陳述式如何格式化用於其規則動作的資料。

教學課程：傳送 Amazon SNS 通知

本教學課程示範如何建立 AWS IoT 規則，將 MQTT 訊息資料傳送至 Amazon SNS 主題，以做為 SMS 文字訊息傳送。

在本教學課程中，您會建立一個規則，在溫度超過規則中所設定的值時，將訊息資料從天氣感應器傳送至 Amazon SNS 主題的所有訂閱者。該規則會在回報的溫度超過規則設定值時進行偵測，然後建立新的訊息承載資料，其中僅包含裝置 ID、回報的溫度及超過的溫度限制。該規則會將新訊息承載作為 JSON 文件傳送至 SNS 主題，其會通知 SNS 主題的所有訂閱者。

您會在本教學課程中學到什麼：

- 如何建立及測試 Amazon SNS 通知
- 如何從 AWS IoT 規則呼叫 Amazon SNS 通知
- 如何在規則查詢陳述式中使用簡單的 SQL 查詢和函數
- 如何使用 MQTT 用戶端測試 AWS IoT 規則

此教學課程約需 30 分鐘方能完成。

於本教學課程中，您將會：

- 步驟 1：建立傳送簡訊的 Amazon SNS 主題
- 步驟 2：建立 AWS IoT 規則以傳送文字訊息
- 步驟 3：測試 AWS IoT 規則和 Amazon SNS 通知
- 步驟 4：檢閱結果及後續步驟

開始本教學課程之前，請確定您有：

- 設定 AWS 帳戶

  您需要 AWS 帳戶 和 AWS IoT 主控台才能完成本教學課程。
- 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

  請確定您可使用 MQTT 用戶端來訂閱並發佈至主題。您會使用 MQTT 用戶端，在此程序中測試您的新規則。
- 檢閱 Amazon Simple Notification Service

若您未曾使用過 Amazon SNS，請查閱設定 Amazon SNS 的存取。如果您已完成其他 AWS IoT 教學課程，則應該已正確設定您的 AWS 帳戶 。

步驟 1：建立傳送簡訊的 Amazon SNS 主題

此程序說明如何建立天氣感應器可以傳送訊息資料的 Amazon SNS 主題。然後，Amazon SNS 主題會透過簡訊通知所有訂閱者超過溫度限制。

若要建立一個傳送 SMS 簡訊的 Amazon SNS 主題

1. 建立一個 Amazon SNS 主題。
   a. 登入 Amazon SNS 主控台。
   b. 在左側導覽窗格中，選擇 Topics (主題)。
   c. 在 Topics (主題) 頁面上，選擇 Create new topic (建立新主題)。
   d. 於 Details (詳細資訊) 中，選擇 Standard (標準) 類型。依預設，主控台會建立一個 FIFO 主題。
   e. 於 Name (名稱) 中，輸入 SNS 主題名稱。針對本教學，輸入 **high_temp_notice**。
   f. 向下捲動到頁面底部，並選擇 Create topic (建立主題)。

      主控台會開啟新主題的 Details (詳細資訊) 頁面。
2. 建立 Amazon SNS 訂閱。

   > **Note**
   > 您在此訂閱中使用的電話號碼可能會因您將在本教學課程中傳送訊息而產生簡訊費用。

   a. 於 high_temp_notice 主題詳細資訊頁面中，選擇 Create subscription (建立訂閱)。
   b. 於Create subscription (建立訂閱) 的 Details (詳細資訊) 區段下，在 Protocol (通訊協定) 清單中選擇 SMS。
   c. 於 Endpoint (端點) 中，輸入可接收簡訊的電話號碼。請務必將其輸入，使其以 + 開頭，包含國碼和地區碼，且不包含任何其他標點符號字元。
   d. 選擇建立訂閱。
3. 測試 Amazon SNS 通知。

a. 於 Amazon SNS 主控台的左側導覽窗格中，選擇 Topics (主題)。

b. 若要開啟主題的詳細資料頁面，請於 Topics (主題) 的主題清單中，選擇 high_temp_notice。

c. 如要開啟 Publish message to topic (將訊息發佈至主題) 頁面，請於 high_temp_notice 詳細資訊頁面中，選擇 Publish message (發佈訊息)。

d. 在 Publish message to topic (將訊息發佈至主題) 的 Message body (訊息內文) 區段下，在 Message body to send to the endpoint (要傳送至端點的訊息內文) 中，輸入簡短訊息。

e. 捲動到頁面底部，並選擇 Publish message (發佈訊息)。

f. 在您先前建立訂閱時所使用的手機號碼上，確認已收到訊息。

若您並未收到測試訊息，請再次檢查電話號碼及手機設定。

請確保您可從 Amazon SNS 主控台 發佈測試訊息，然後再繼續教學課程。

步驟 2：建立 AWS IoT 規則以傳送文字訊息

您將在本教學課程中建立的 AWS IoT 規則會訂閱 `device/`*`device_id`*`/data` MQTT 主題，其中 *`device_id`* 是傳送訊息的裝置 ID。主題篩選條件會將這些主題描述為 `device/+/data`，其中 `+` 為與兩個正斜線字元間之任何字串相符的萬用字元。此規則也會測試訊息裝載中的 `temperature` 欄位值。

當規則收到來自相符主題的訊息時，其會採用來自主題名稱的 *`device_id`*、來自訊息承載的 `temperature` 值、及為其測試的限制新增一個常數值，並將這些值做為 JSON 文件傳送至 Amazon SNS 通知主題。

例如，來自天氣感應器裝置編號 32 的 MQTT 訊息會使用 `device/32/data` 主題，並具有看起來如下的訊息承載：

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

規則的規則查詢陳述式採用訊息承載的 `temperature` 值、主題名稱的 *`device_id`*，並新增常數 `max_temperature` 值，以將看起來像這樣的訊息承載傳送至 Amazon SNS 主題：

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

建立 AWS IoT 規則以偵測超額溫度值，並建立要傳送至 Amazon SNS 主題的資料

1. 開啟 AWS IoT 主控台的規則中樞。
2. 若此為您的第一個規則，請選擇 Create (建立) 或 Create a rule (建立規則)。
3. 在 Create a rule (建立規則) 中：

   a. 在 Name (名稱) 中，輸入 **`temp_limit_notify`**。

      請記住，規則名稱在您的 AWS 帳戶 和 區域中必須是唯一的，而且不能有任何空格。我們在此名稱中使用底線字元來分隔規則名稱中的字詞。

   b. 在 Description (說明) 中，說明規則。

      有意義的說明可讓您更容易記住此規則的作用及您建立規則的原因。說明可依所需而定，因此請盡可能詳細說明。

4. 在 Create a rule (建立規則) 的 Rule query statement (規則查詢陳述式) 中：

   a. 在使用 SQL 版本中，選取 2016-03-23。

   b. 在 Rule query statement (規則查詢陳述式) 編輯方塊中輸入陳述式：

      ```
      SELECT topic(2) as device_id,
          temperature as reported_temperature,
          30 as max_temperature
        FROM 'device/+/data'
        WHERE temperature > 30
      ```

      本陳述式：

      - 聆聽主題與 `device/+/data` 主題篩選條件相符及 `temperature` 值大於 30 的 MQTT 訊息。
      - 從主題字串中選取第二個元素，並將其指定給 `device_id` 欄位。

- 從訊息承載選取值 `temperature` 欄位，並將其指派給 `reported_temperature` 欄位。
- 建立常數值 `30` 來表示限制值，並將其指定給 `max_temperature` 欄位。

5. 若要開啟此規則的規則動作清單，請於 Set one or more actions (設定一個或多個動作) 中，選擇 Add action (新增動作)。
6. 在 Select an action (選取動作) 中，選擇 Send a message as an SNS push notification (傳送 SNS 推送通知形式的訊息)。
7. 若要開啟所選取動作的組態頁面，請在動作清單底部選擇 Configure action (設定動作)。
8. 於 Configure action (設定動作)：

   a. 於 SNS target (SNS 目標) 中，選擇 Select (選取)，找出您名為 high_temp_notice 的 SNS 主題，然後選擇 Select (選取)。

   b. 在 Message format (訊息格式) 中，選擇 RAW。

   c. 在選擇或建立角色以授予執行此動作的 AWS IoT 存取權中，選擇建立角色。

   d. 在 Create a new role (建立新角色) 的 Name (名稱) 中，輸入新角色的唯一名稱。在本教學課程中，使用 **`sns_rule_role`**。

   e. 選擇 Create Role (建立角色)。

   若您要重複此教學課程或重複使用現有的角色，請先選擇 Update role (更新角色)，再繼續進行。此會更新角色的政策文件，以使用 SNS 目標。

9. 選擇 Add action (新增動作)，並返回 Create a rule (建立規則) 頁面。

   在新動作的圖標中，Send a message as an SNS push notification (以 SNS 推送通知形式傳送訊息)，您可看到規則呼叫的 SNS 主題。

   這是您將新增至此規則的唯一規則動作。

10. 如要建立並完成此步驟，請於 Create a rule (建立規則) 中，向下捲動至底部，然後選擇 Create rule (建立規則)。

步驟 3：測試 AWS IoT 規則和 Amazon SNS 通知

若要測試新規則，您將使用 MQTT 用戶端來發佈和訂閱此規則所使用的 MQTT 訊息。

在新視窗的 AWS IoT 主控台中開啟 MQTT 用戶端。這可讓您編輯規則，而不會遺失 MQTT 用戶端的組態。如果您讓 MQTT 用戶端前往主控台中的另一個頁面，其不會保留任何訂閱或訊息記錄。

如要使用 MQTT 用戶端來測試您的規則。

1. 在 AWS IoT 主控台的 MQTT 用戶端 中，訂閱輸入主題，在此案例中為 `device/+/data`。

   a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Subscribe to a topic (訂閱主題)。

   b. 在 Subscription topic (訂閱主題) 中，輸入輸入主題篩選條件 **`device/+/data`** 的主題 。

   c. 將剩下的欄位保留為其預設設定。

   d. 請選擇 Subscribe to topic (訂閱主題)。

      在 Subscriptions (訂閱) 欄中，Publish to a topic (發佈到主題) 之下，**`device/+/data`** 隨即顯示。

2. 使用特定裝置 ID **`device/32/data`**，將訊息發佈至輸入主題。您無法發佈至包含萬用字元的 MQTT 主題。

   a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Publish to topic (發佈至主題)。

   b. 在 Publish (發佈) 欄位中輸入輸入主題名稱 **`device/32/data`**。

   c. 複製此處顯示的範例資料，並在主題名稱下方的編輯方塊中貼上範例資料。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. 選擇 Publish to topic (發佈至主題)，發佈您的 MQTT 訊息。

3. 確認簡訊已傳送。

   a. 在 MQTT 用戶端中 Subscriptions (訂閱) 下，您先前訂閱的主題旁會有一個綠點。

      該綠點表示自上次查看後，已收到一個或多個新訊息。

   b. 於 Subscriptions (訂閱) 下，選擇 device/+/data，來檢查訊息承載是否與您剛剛發佈的內容相符，如下所示：

```
{
  "temperature": 38,
```

```
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. 檢查您用來訂閱 SNS 主題的手機，並確認訊息承載內容如下所示：

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

請注意，device_id 值是個帶有引號的字串，而 temperature 值是個數字。這是因為 topic() 函數從輸入訊息的主題名稱中提取字串，而 temperature 值會使用輸入訊息承載的數值。

若要將 device_id 值設為數值，請將規則查詢陳述式中的 topic(2) 替換為：

```
cast(topic(2) AS DECIMAL)
```

請注意，將 topic(2) 值轉換為數值，DECIMAL 值僅適用於該部分主題僅包含數字字元時。

4. 請嘗試傳送溫度不超過限制的 MQTT 訊息。

a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Publish to topic (發佈至主題)。

b. 在 Publish (發佈) 欄位中輸入輸入主題名稱 **device/33/data**。

c. 複製此處顯示的範例資料，並在主題名稱下方的編輯方塊中貼上範例資料。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 若要傳送 MQTT 訊息，請選擇 Publish to topic (發佈至主題)。

您應該能看到您在 **device/+/data** 訂閱中傳送的訊息。不過，因溫度值低於規則查詢陳述式中的最高溫度，所以您不應收到簡訊。

若您並未看到正確的行為，請查看疑難排解提示。

疑難排解 SNS 訊息規則

若您並未看到預期的結果，請查看以下事項。

- 您收到錯誤的橫幅

  若在您發佈輸入訊息時出現錯誤，請先更正該錯誤。下列步驟可協助您修正該錯誤。

- 您並未在 MQTT 用戶端中看到輸入訊息

  每次您將輸入訊息發佈至 `device/22/data` 主題中，若您依程序中所述訂閱了 `device/+/data` 主題篩選條件，則該訊息應會顯示於 MQTT 用戶端中。

  要檢查的事項

  - 檢查您訂閱的主題篩選條件

    若您依程序中所述訂閱了輸入訊息主題，則每次發佈輸入訊息時都應該會看到其複本。

    若您並未訊息，請檢查您訂閱的主題名稱，並將其與所發佈的主題進行比較。主題名稱區分大小寫，且您訂閱的主題必須與所發佈訊息承載的主題相同。

  - 檢查訊息發佈功能

    在 MQTT 用戶端中的 Subscriptions (訂閱) 下，選擇 device/+/data，檢查發佈訊息的主題，然後選擇 Publish to topic (發佈至主題)。您應該會在訊息清單中出現主題下方的編輯方塊中看到訊息承載。

- 您並未收到 SMS 訊息：

  若要讓您的規則運作，其必須具有授權其接收訊息和傳送 SNS 通知的正確政策，且必須接收訊息。

  要檢查的事項

  - 檢查 MQTT 用戶端 AWS 區域 的 和您建立的規則

    您正在執行 MQTT 用戶端的主控台必須與您建立的規則處於相同的 AWS 區域。

- 檢查訊息承載中的溫度值是否超過測試閾值

  若溫度值小於或等於 30 (如規則查詢陳述式中所定義)，則規則將不會執行其任何動作。
- 檢查規則查詢陳述式中的輸入訊息主題

  若要讓規則運作，其必須收到一則訊息，其主題名稱與規則查詢陳述式之 FROM 子句中的主題篩選條件相符。

  檢查規則查詢陳述式中主題篩選條件的拼字與 MQTT 用戶端中主題的拼字。主題名稱區分大小寫，且郵件的主題必須與規則查詢陳述式中的主題篩選條件相符。
- 檢查輸入訊息承載的內容

  若要讓規則運作，其必須在 SELECT 陳述式中宣告的訊息承載中尋找資料欄位。

  檢查規則查詢陳述式中 `temperature` 欄位的拼字與 MQTT 用戶端中訊息承載的拼字。欄位名稱區分大小寫，規則查詢陳述式中的 `temperature` 欄位必須與訊息承載中的 `temperature` 欄位相符。

  請確定訊息承載中的 JSON 文件格式正確。若 JSON 有任何錯誤，例如缺少逗號，則規則將無法進行讀取。
- 檢查規則動作中重新發佈的訊息主題

  Republish (重新發佈) 規則動作發佈新訊息的主題必須與您在 MQTT 用戶端中訂閱的主題相符。

  開啟您建立於主控台中的規則，並檢查規則動作重新發佈訊息的主題。
- 檢查規則所使用的角色

  規則動作必須具有接收原始主題及發佈新主題的權限。

  授權規則接收訊息資料的政策並加以重新發佈為所使用的主題所特定的。若變更用於重新發佈訊息資料的主題，則必須更新規則動作的角色，來更新其政策以與目前主題相符。

  若您懷疑這會是問題，請編輯 Republish (重新發佈) 規則動作並建立新角色。規則動作建立的新角色會收到執行這些動作所需的授權。

步驟 4：檢閱結果及後續步驟

於本教學課程中：

- 您已建立並測試了 Amazon SNS 通知主題和訂閱。

- 您在規則查詢陳述式中使用了一個簡單的 SQL 查詢和函數，可為您的通知建立一個新訊息。
- 您已建立 AWS IoT 規則來傳送使用自訂訊息承載的 Amazon SNS 通知。
- 您使用 MQTT 用戶端來測試 AWS IoT 規則。

後續步驟

使用此規則傳送一些簡訊之後，請嘗試使用其來查看教學課程的某些層面如何影響訊息，及訊息傳送的時間。此處有幾種簡單的入門方式。

- 變更輸入訊息主題中的 *device_id*，並觀察簡訊內容中的影響。
- 變更規則查詢陳述式中所選取的欄位，並觀察簡訊內容中的影響。
- 變更規則查詢陳述式中的測試，以測試最低溫度，而非最高溫度。請記得變更 `max_temperature` 的名稱！
- 新增重新發佈規則動作，以於傳送 SNS 通知時傳送 MQTT 訊息。
- 請嘗試本系列中的下一個教學課程，並了解如何進行 教學課程：將裝置資料儲存在 DynamoDB 表格中。

教學課程：將裝置資料儲存在 DynamoDB 表格中

本教學課程示範如何建立將訊息資料傳送至 DynamoDB 資料表的 AWS IoT 規則。

在此教學課程中，您會建立一個規則，將訊息資料從虛構天氣感應器裝置傳送至 DynamoDB 表格。該規則會格式化來自多個天氣感應器的資料，由此可將其新增至單個資料庫表格中。

您會在本教學課程中學到什麼

- 如何建立 DynamoDB 表格
- 如何從 AWS IoT 規則傳送訊息資料至 DynamoDB 資料表
- 如何在 AWS IoT 規則中使用替代範本
- 如何在規則查詢陳述式中使用簡單的 SQL 查詢和函數
- 如何使用 MQTT 用戶端測試 AWS IoT 規則

此教學課程約需 30 分鐘方能完成。

於本教學課程中，您將會：

- 步驟 1：為本教學課程建立 DynamoDB 表格

- 步驟 2：建立 AWS IoT 規則以將資料傳送至 DynamoDB 資料表
- 步驟 3：測試 AWS IoT 規則和 DynamoDB 資料表
- 步驟 4：檢閱結果及後續步驟

開始本教學課程之前，請確定您有：

- 設定 AWS 帳戶

  您需要 AWS 帳戶 和 AWS IoT 主控台才能完成本教學課程。
- 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

  請確定您可使用 MQTT 用戶端來訂閱並發佈至主題。您會使用 MQTT 用戶端，在此程序中測試您的新規則。
- 檢閱了 Amazon DynamoDB 概觀

  若您未曾使用過 DynamoDB，請查閱 DynamoDB 入門，以熟悉 DynamoDB 的基本概念和作業。

步驟 1：為本教學課程建立 DynamoDB 表格

於本教學課程中，您會建立一個具下列屬性的 DynamoDB 表格，以記錄來自虛構氣候感應器裝置的資料：

- `sample_time` 為主索引鍵，並說明記錄範例的時間。
- `device_id` 為排序索引鍵，並說明提供範例的裝置
- `device_data` 為從裝置接收並由規則查詢陳述式格式化的資料

如要建立本教學課程的 DynamoDB 表格

1. 開啟 DynamoDB 主控台，然後選擇 Create table (建立表格)。
2. 在 Create table (建立資料表) 中：
   a. 於 Table name (表格名稱) 中，輸入表格名稱：**`wx_data`**。
   b. 在 Primary key (主索引鍵) 中，輸入 **`sample_time`**，然後在欄位旁的選項清單中，選擇 **`Number`**。
   c. 在 Sort key (排序索引鍵) 中，輸入 **`device_id`**，然後在欄位旁的選項清單中，選擇 **`Number`**。

d. 請在頁面底部，選擇 Create (建立)。

您稍後將在設定 DynamoDB 規則動作時定義 `device_data`。

步驟 2：建立 AWS IoT 規則以將資料傳送至 DynamoDB 資料表

於此步驟中，您會使用規則查詢陳述式，格式化虛構天氣感應器裝置的資料，以寫入資料庫表格。

從天氣感應器裝置接收的訊息承載範例如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

對於資料庫項目，您會使用規則查詢陳述式將訊息承載的結構平面化，如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

於此規則中，您還會使用一些 替代範本。替代範本是可讓您從函數和訊息資料插入動態值的運算式。

建立 AWS IoT 規則以將資料傳送至 DynamoDB 資料表

1. 開啟AWS IoT 主控台的 Rules (規則) 中樞。或者，您可以在 AWS 管理主控台 中開啟 AWS IoT 首頁，並導覽至訊息路由 > 規則。
2. 如要在 Rules (規則) 中開始建立新規則，請選擇 Create rule (建立規則)。
3. 在 Rule properties (規則屬性) 中：

   a. 在 Rule name (規則名稱) 中，輸入 **`wx_data_ddb`**。

請記住，規則名稱在您的 AWS 帳戶 和 區域中必須是唯一的，而且不能有任何空格。我們在此名稱中使用底線字元來分隔規則名稱中的兩個單字。

b. 在 Rule description (規則說明) 中，說明規則。

有意義的說明可讓您更容易記住此規則的作用及您建立規則的原因。說明可依所需而定，因此請盡可能詳細說明。

4. 選擇 Next (下一步) 繼續。
5. 在 SQL statement (SQL 陳述式) 中：

a. 在 SQL version (SQL 版本) 中，選取 **2016-03-23**。

b. 在 SQL statement (SQL 陳述式) 編輯方塊中，輸入陳述式：

```
SELECT temperature, humidity, barometer,
  wind.velocity as wind_velocity,
  wind.bearing as wind_bearing,
FROM 'device/+/data'
```

本陳述式：

- 聆聽具與 `device/+/data` 主題篩選條件相符之主題的 MQTT 訊息。
- 將 `wind` 屬性的元素格式化為個別屬性。
- 傳遞 `temperature`、`humidity` 和 `barometer` 屬性不變。

6. 選擇 Next (下一步) 繼續。
7. 在 Rule actions (規則動作) 中：

a. 若要開啟此規則的規則動作清單，請於 Action 1 (動作 1) 中，選擇 **DynamoDB**。

> **Note**
>
> 請務必選擇 DynamoDB 而非 DynamoDBv2 作為規則動作。

b. 於 Table name (表格名稱) 中，選擇您在先前步驟中建立的 DynamoDB 表格名稱：**wx_data**。

Partition key type (分割區索引鍵類型) 和 Sort key type (排序索引鍵類型) 欄位，都會填入 DynamoDB 表格中的值。

c. 在 Partition key (分區索引鍵)，輸入 **sample_time**。

d. 在 Partition key value (分割區索引鍵值) 中，輸入 **${timestamp()}**。

這是您將用於此規則中 替代範本 的首項。而非使用訊息承載中的值，其會使用 timestamp 函數傳回的值。如需進一步了解，請參閱AWS IoT Core 開發人員指南中的時間戳記。

e. 在 Sort key (排序索引鍵) 中，輸入 **device_id**。

f. 在 Sort key value (排序索引鍵值) 中，輸入 **${cast(topic(2) AS DECIMAL)}**。

這是您將用於此規則中 替代範本 第二項。其會在主題名稱中插入第二個元素值，即裝置的 ID，然後將其轉換為 DECIMAL 值，以與索引鍵的數值格式相符。若要進一步了解主題，請參閱AWS IoT Core 開發人員指南中的主題。或者，若要了解將值轉換為數值的詳細資訊，請參閱AWS IoT Core 開發人員指南中的將值轉換為數值。

g. 在 Write message data to this column (寫入訊息資料至此欄) 中輸入 **device_data**。

這會建立 DynamoDB 表格中的 `device_data` 欄。

h. 將 Operation (操作) 保持空白。

i. 在 IAM Role (IAM 角色) 中，選擇 Create a new Role (建立新角色)。

j. 在 Create role (建立角色) 對話方塊中，請為 Role name (角色名稱) 輸入 wx_ddb_role。這個新角色會自動包含以 "aws-iot-rule" 為字首的政策，其允許 **wx_data_ddb** 規則將資料傳送到您建立的 **wx_data** DynamoDB 資料表。

k. 在 IAM role (IAM 角色) 中，選擇 **wx_ddb_role**。

l. 請選擇頁面最下方的 Next (下一頁)。

8. 請在 Review and create (檢閱和建立) 頁面底部，選擇 Create (建立) 來建立規則。

步驟 3：測試 AWS IoT 規則和 DynamoDB 資料表

若要測試新規則，您會使用 MQTT 用戶端來發佈和訂閱用於此測試的 MQTT 訊息。

在新視窗的 AWS IoT 主控台中開啟 MQTT 用戶端。這可讓您編輯規則，而不會遺失 MQTT 用戶端的組態。如果您讓它轉到主控台中的另一個頁面，MQTT 用戶端不會保留任何訂閱或訊息記錄。您也需要在主控台中開啟單獨的AWS IoT 主控台視窗至 DynamoDB Tables 中樞，以檢視規則傳送的新項目。

如要使用 MQTT 用戶端來測試您的規則。

1. 於 AWS IoT 主控台的 MQTT 用戶端 中，訂閱輸入主題 `device/+/data`。

a. 於 MQTT 用戶端中，選擇 Subscribe to a topic (訂閱主題)。

b. 若為 Topic filter (主題篩選條件)，請輸入輸入主題篩選條件的主題 **device/+/data**。

c. 選擇 Subscribe (訂閱)。

2. 現在，使用特定裝置 ID **device/22/data** 將訊息發佈至輸入主題。您無法發佈至包含萬用字元的 MQTT 主題。

a. 於 MQTT 用戶端中，請選擇 Publish to a topic (發佈至主題)。

b. 若為 Topic name (主題名稱)，請輸入輸入主題名稱 **device/22/data**。

c. 若為 Message payload (訊息承載)，請輸入下列範例資料。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 如要發佈 MQTT 訊息，請選擇 Publish (發佈)。

e. 現在，於 MQTT 用戶端中，選擇 Subscribe to a topic (訂閱主題)。於 Subscribe (訂閱) 欄中，選擇 **device/+/data** 訂閱。確認上一步的範例資料會顯示於此處。

3. 查看您規則所建立 DynamoDB 表格中的列。

a. 在AWS IoT 主控台的 DynamoDB Tables 中樞中，選擇 wx_data，然後選擇項目索引標籤。

若您已在 Items (項目) 索引標籤上，您可能需要選擇表格標題右上角的重新整理圖示來重新整理該顯示。

b. 請注意，表格中的 sample_time 值為連結並開啟一個。若您剛傳送了第一則訊息，則其將是清單中唯一的訊息。

此連結會顯示表格該列中的所有資料。

c. 展開 device_data 項目，查看規則查詢陳述式所產生的資料。

d. 探索此顯示中可用資料的不同表示法。您亦可於此顯示中編輯資料。

e. 完成查閱此資料列之後，如要儲存所做的任何變更，請選擇 Save (儲存)，或者，如要結束而不儲存任何變更，請選擇 Cancel (取消)。

若您並未看到正確的行為，請查看疑難排解提示。

對您的 DynamoDB 規則進行疑難排解

若您並未看到預期的結果，請查看以下事項。

- 您收到錯誤的橫幅

  若在您發佈輸入訊息時出現錯誤，請先更正該錯誤。下列步驟可協助您修正該錯誤。

- 您並未在 MQTT 用戶端中看到輸入訊息

  每次您將輸入訊息發佈至 `device/22/data` 主題時，若您依程序中所述訂閱了 `device/+/data` 主題篩選條件，則該訊息應會顯示於 MQTT 用戶端中。

  要檢查的事項

  - 檢查您訂閱的主題篩選條件

    若您依程序中所述訂閱了輸入訊息主題，則每次發佈輸入訊息時都應該會看到其複本。

    若您並未訊息，請檢查您訂閱的主題名稱，並將其與所發佈的主題進行比較。主題名稱區分大小寫，且您訂閱的主題必須與所發佈訊息承載的主題相同。

  - 檢查訊息發佈功能

    在 MQTT 用戶端中的 Subscriptions (訂閱) 下，選擇 device/+/data，檢查發佈訊息的主題，然後選擇 Publish to topic (發佈至主題)。您應該會在訊息清單中出現主題下方的編輯方塊中看到訊息承載。

- 您在 DynamoDB 表格中看不到您的資料

  首先要做的是選擇表格標題右上角的重新整理圖示，以重新整理顯示。若未顯示您正在尋找的資料，請檢查下列內容。

  要檢查的事項

  - 檢查 MQTT 用戶端 AWS 區域 的 和您建立的規則

    您正在執行 MQTT 用戶端的主控台必須與您建立的規則處於相同的 AWS 區域。

  - 檢查規則查詢陳述式中的輸入訊息主題

    若要讓規則運作，其必須收到一則訊息，其主題名稱與規則查詢陳述式之 FROM 子句中的主題篩選條件相符。

檢查規則查詢陳述式中主題篩選條件的拼字與 MQTT 用戶端中主題的拼字。主題名稱區分大小寫，且郵件的主題必須與規則查詢陳述式中的主題篩選條件相符。

- 檢查輸入訊息承載的內容

  若要讓規則運作，其必須在 SELECT 陳述式中宣告的訊息承載中尋找資料欄位。

  檢查規則查詢陳述式中 `temperature` 欄位的拼字與 MQTT 用戶端中訊息承載的拼字。欄位名稱區分大小寫，規則查詢陳述式中的 `temperature` 欄位必須與訊息承載中的 `temperature` 欄位相符。

  請確定訊息承載中的 JSON 文件格式正確。若 JSON 有任何錯誤，例如缺少逗號，則規則將無法進行讀取。

- 檢查用於規則動作中的索引鍵和欄位名稱

  用於主題規則中的欄位名稱必須與已發佈訊息之 JSON 訊息承載中找到的欄位名稱相符。

  開啟您建立於主控台中的規則，並檢查規則動作組態中的欄位名稱與用於 MQTT 用戶端中的欄位名稱。

- 檢查規則所使用的角色

  規則動作必須具有接收原始主題及發佈新主題的權限。

  授權規則以接收訊息資料及更新 DynamoDB 表格的政策是所使用主題特有的。若您變更規則使用的主題或 DynamoDB 表格名稱，則必須更新規則動作的角色，以更新其相符的政策。

  若您懷疑這會是問題，請編輯規則動作並建立新角色。規則動作建立的新角色會收到執行這些動作所需的授權。

步驟 4：檢閱結果及後續步驟

在您使用此規則將數則訊息傳送至 DynamoDB 表格後，請試著對其進行試驗，以查看變更教學課程中的某些層面如何影響寫入表格的資料。此處有幾種簡單的入門方式。

- 變更輸入訊息主題中的 *`device_id`*，並觀察對資料的影響。您可使用此來模擬從多個天氣感應器接收資料。
- 變更規則查詢陳述式中所選取的欄位，並觀察對資料的影響。您可以此來篩選儲存於表格中的資料。
- 新增重新發佈規則動作，為新增至表格中的每一列傳送 MQTT 訊息。您可以此來進行除錯。

完成本教學課程後，請查看 the section called “使用 AWS Lambda 函數格式化通知”。

教學課程：使用 AWS Lambda 函數格式化通知

本教學課程示範如何將 MQTT 訊息資料傳送至 AWS Lambda 動作，以格式化和傳送至其他 AWS 服務。在本教學課程中， AWS Lambda 動作會使用 AWS 開發套件，將格式化的訊息傳送至您在教學課程中建立的 Amazon SNS 主題，了解如何 the section called “正在傳送 Amazon SNS 通知”。

於有關如何進行 the section called “正在傳送 Amazon SNS 通知” 的教學課程中，由規則查詢陳述式產生的 JSON 文件 作為簡訊的主文傳送。結果為一則看似此範例的簡訊：

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

在本教學課程中，您將使用 AWS Lambda 規則動作來呼叫 AWS Lambda 函數，將規則查詢陳述式中的資料格式化為較友善的格式，例如此範例：

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

您將在本教學課程中建立的 AWS Lambda 函數會使用規則查詢陳述式中的資料來格式化訊息字串，並呼叫 AWS SDK 的 SNS 發佈函數來建立通知。

您會在本教學課程中學到什麼

- 如何建立和測試 AWS Lambda 函數
- 如何在 AWS Lambda 函數中使用 AWS SDK 發佈 Amazon SNS 通知
- 如何在規則查詢陳述式中使用簡單的 SQL 查詢和函數
- 如何使用 MQTT 用戶端測試 AWS IoT 規則

此教學課程約需 45 分鐘方能完成。

於本教學課程中，您將會：

- 步驟 1：建立傳送文字訊息的 AWS Lambda 函數
- 步驟 2：使用 AWS IoT 規則動作建立 AWS Lambda 規則
- 步驟 3：測試 AWS IoT 規則和 AWS Lambda 規則動作
- 步驟 4：檢閱結果及後續步驟

開始本教學課程之前，請確定您有：

- 設定 AWS 帳戶

  您需要 AWS 帳戶 和 AWS IoT 主控台才能完成本教學課程。

- 檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息

  請確定您可使用 MQTT 用戶端來訂閱並發佈至主題。您會使用 MQTT 用戶端，在此程序中測試您的新規則。

- 已完成本節中的其他規則教學課程

  本教學課程需要您於教學課程中建立有關如何進行 the section called “正在傳送 Amazon SNS 通知” 的 SNS 通知主題。其亦假設您已完成本節中其他與規則相關的教學課程。

- 檢閱了 AWS Lambda 概觀

  如果您 AWS Lambda 未曾使用過 ，請檢閱AWS Lambda和開始使用 Lambda 以了解其術語和概念。

步驟 1：建立傳送文字訊息的 AWS Lambda 函數

本教學課程中的 AWS Lambda 函數會收到規則查詢陳述式的結果、將元素插入文字字串，並將產生的字串作為通知中的訊息傳送至 Amazon SNS。

與the section called “正在傳送 Amazon SNS 通知”如何使用規則 AWS IoT 動作傳送通知的 教學課程不同，本教學課程會使用 AWS SDK 的 函數，從 Lambda 函數傳送通知。然而，用於本教學課程中的實際 Amazon SNS 通知主題與您在教學課程中有關如何進行 the section called “正在傳送 Amazon SNS 通知” 相同。

建立傳送文字訊息的 AWS Lambda 函數

1. 建立新的 AWS Lambda 函數。

   a. 於 AWS Lambda 主控台中，選擇 Create function (建立函數)。

   b. 於 Create function (建立函數) 中，選取 Use a blueprint (使用藍圖)。

      搜尋並選取 **`hello-world-python`** 藍圖，然後選擇 Configure (設定)。

   c. 於基本資訊中：

i. 在 Function name (函數名稱) 中，輸入此函數的名稱 **format-high-temp-notification**。

ii. 在執行角色中，選擇從 AWS 政策範本建立新角色。

iii. 於 Role name (角色名稱) 中，輸入新角色 **format-high-temp-notification-role** 的名稱。

iv. 在 Policy templates - optional (政策範本：選用) 中，搜尋並選取 Amazon SNS publish policy (Amazon SNS 發佈政策)。

v. 選擇建立函數。

2. 修改藍圖程式碼以進行格式化並傳送 Amazon SNS 通知。

a. 在您建立函數之後，您應該會看到 format-high-temp-notification 詳細資訊頁面。若您未看到，請從 Lambda 函數 頁面上加以開啟。

b. 在format-high-temp-notification 詳細資訊頁面中，選擇Configuration (組態) 索引標籤，然後捲動至 Function code (函數程式碼) 面板。

c. 於 Function code (函數程式碼) 視窗的 Environment (環境) 窗格中，選擇 Python 檔案 `lambda_function.py`。

d. 於 Function code (函數程式碼) 視窗中，刪除藍圖中的所有原始程式程式碼，並以此程式碼進行取代。

```
import boto3
#
#   expects event parameter to contain:
#   {
#       "device_id": "32",
#       "reported_temperature": 38,
#       "max_temperature": 30,
#       "notify_topic_arn": "arn:aws:sns:us-
east-1:57EXAMPLE833:high_temp_notice"
#   }
#
#   sends a plain text string to be used in a text message
#
#      "Device {0} reports a temperature of {1}, which exceeds the limit of
 {2}."
#
#   where:
#       {0} is the device_id value
#       {1} is the reported_temperature value
```

```
#       {2} is the max_temperature value
#
def lambda_handler(event, context):

    # Create an SNS client to send notification
    sns = boto3.client('sns')

    # Format text message from data
    message_text = "Device {0} reports a temperature of {1}, which exceeds the
 limit of {2}.".format(
            str(event['device_id']),
            str(event['reported_temperature']),
            str(event['max_temperature'])
        )

    # Publish the formatted message
    response = sns.publish(
            TopicArn = event['notify_topic_arn'],
            Message = message_text
        )

    return response
```

e. 選擇部署。

3. 在新視窗中，從教學課程有關如何進行 the section called “正在傳送 Amazon SNS 通知” 中查詢您 Amazon SNS 主題的 Amazon 資源名稱 (ARN)。

   a. 於新視窗中，開啟 Amazon SNS 主控台的主題頁面。

   b. 於 Topics (主題) 頁面中，於 Amazon SNS 主題清單中尋找 high_temp_notice 通知主題。

   c. 尋找 high_temp_notice 通知主題的 ARN，以用於下一個步驟中。

4. 建立您 Lambda 函數的測試案例。

   a. 於主控台的 Lambda 函數頁面的 format-high-temp-notification 詳細資訊頁面上，選擇該頁面右上角中的 Select a test event (選擇測試事件) (即使看似已停用)，然後選擇 Configure test events (設定測試事件)。

   b. 於 Configure test event (設定測試事件) 中，選擇 Create new test event (建立新的測試事件)。

   c. 於 Event name (事件名稱) 中，輸入 **SampleRuleOutput**。

   d. 在 Event name (事件名稱) 下的 JSON 編輯器，請貼上此範例 JSON 文件。這是您的 AWS IoT 規則將傳送到 Lambda 函數的範例。

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

e. 請參閱具有 high_temp_notice 通知主題之 ARN 的視窗，然後複製 ARN 值。

f. 以您通知主題的 ARN 取代 JSON 編輯器中的 `notify_topic_arn` 值。

保持此視窗的開啟狀態，您可於建立 AWS IoT 規則時再次使用此 ARN 值。

g. 選擇建立。

5. 以範例資料測試函數。

a. 於 format-high-temp-notification 詳細資料頁面的右上角中，確認 SampleRuleOutput 會顯示於 Test (測試) 按鈕旁。若未顯示，請從可用的測試事件清單中進行選擇。

b. 如要將範例規則輸出訊息傳送至您的函數，請選擇 Test (測試)。

若函數和通知皆有效，您會在訂閱通知的手機上收到一則簡訊。

若手機上並未收到簡訊，請檢查作業結果。於 Function code (函數程式碼) 面板中的 Execution result (執行結果) 索引標籤中，查閱回應以找出發生的任何錯誤。在您的函數可將通知傳送至手機之前，請勿繼續進行下一個步驟。

步驟 2：使用 AWS IoT 規則動作建立 AWS Lambda 規則

於此步驟中，您會使用規則查詢陳述式，格式化虛構天氣感應器裝置的資料，以傳送至 Lambda 函數，該函數將會格式化並傳送簡訊。

從天氣裝置接收到的訊息承載裝置的範例如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
```

```
}
```

於此規則中，您會使用規則查詢陳述式來建立 Lambda 函數的訊息承載，如下所示：

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

此包含 Lambda 函數格式化及傳送正確簡訊所需的所有資訊。

建立 AWS IoT 規則以呼叫 Lambda 函數

1. 開啟 AWS IoT 主控台的規則中樞。
2. 如要於 Rules (規則) 中開始建立新規則，請選擇 Create (建立)。
3. 在 Create a rule (建立規則) 的頂部中：

   a. 於 Name (名稱) 中，輸入規則名稱 **wx_friendly_text**。

      請記住，規則名稱在您的 AWS 帳戶 和 區域中必須是唯一的，而且不能有任何空格。我們在此名稱中使用底線字元來分隔規則名稱中的兩個單字。

   b. 在 Description (說明) 中，說明規則。

      有意義的說明可讓您更容易記住此規則的作用及您建立規則的原因。說明可依所需而定，因此請盡可能詳細說明。
4. 在 Create a rule (建立規則) 的 Rule query statement (規則查詢陳述式) 中：

   a. 在使用 SQL 版本中，選取 **2016-03-23**。

   b. 在 Rule query statement (規則查詢陳述式) 編輯方塊中輸入陳述式：

      ```
      SELECT
        cast(topic(2) AS DECIMAL) as device_id,
        temperature as reported_temperature,
        30 as max_temperature,
        'arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice' as notify_topic_arn
      FROM 'device/+/data' WHERE temperature > 30
      ```

      本陳述式：

- 聆聽主題與 device/+/data 主題篩選條件相符及 temperature 值大於 30 的 MQTT 訊息。
- 從主題字串選取第二個元素，將其轉換為十進位數字，進而指派給 device_id 欄位。
- 從訊息承載選取值 temperature 欄位的值，然後將其指派給 reported_temperature 欄位。
- 建立常數值 30 來表示限制值，並將其指定給 max_temperature 欄位。
- 建立 notify_topic_arn 欄位的常數值。

c. 請參閱具有 high_temp_notice 通知主題之 ARN 的視窗，然後複製 ARN 值。

d. 以您通知主題的 ARN 取代規則查詢陳述式編輯器中的 ARN 值 (*arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice*)。

5. 於 Set one or more actions (設定一個或多個動作) 中：

a. 若要開啟此規則的規則動作清單，請選擇 Add action (新增動作)。

b. 於 Select an action (選取動作) 中，選擇 Send a message to a Lambda function (將訊息傳送至 Lambda 函數)。

c. 若要開啟所選取動作的組態頁面，請在動作清單底部選擇 Configure action (設定動作)。

6. 於 Configure action (設定動作)：

a. 於 Function name (函數名稱) 中，選擇 Select (選取)。

b. 選擇 format-high-temp-notification。

c. 在 Configure action (設定動作) 的底部，選擇 Add action (新增動作)。

d. 如要建立規則，請於 Create a rule (建立規則) 中，選擇 Create rule (建立規則)。

步驟 3：測試 AWS IoT 規則和 AWS Lambda 規則動作

若要測試新規則，您將使用 MQTT 用戶端來發佈和訂閱此規則所使用的 MQTT 訊息。

在新視窗的 AWS IoT 主控台中開啟 MQTT 用戶端。您現在可以編輯規則，而不會遺失 MQTT 用戶端的設定。如果您讓 MQTT 用戶端前往主控台中的另一個頁面，您將會遺失您的訂閱或訊息記錄。

如要使用 MQTT 用戶端來測試您的規則。

1. 在 AWS IoT 主控台的 MQTT 用戶端 中，訂閱輸入主題，在此案例中為 device/+/data。

a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Subscribe to a topic (訂閱主題)。

b. 在 Subscription topic (訂閱主題) 中，輸入輸入主題篩選條件 **device/+/data** 的主題 。

c. 將剩下的欄位保留為其預設設定。

d. 請選擇 Subscribe to topic (訂閱主題)。

在 Subscriptions (訂閱) 欄中，Publish to a topic (發佈到主題) 之下，**device/+/data** 隨即顯示。

2. 使用特定裝置 ID **device/32/data**，將訊息發佈至輸入主題。您無法發佈至包含萬用字元的 MQTT 主題。

a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Publish to topic (發佈至主題)。

b. 在 Publish (發佈) 欄位中輸入輸入主題名稱 **device/32/data**。

c. 複製此處顯示的範例資料，並在主題名稱下方的編輯方塊中貼上範例資料。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 如要發佈您的 MQTT 訊息，請選擇 Publish to topic (發佈至主題)。

3. 確認簡訊已傳送。

a. 在 MQTT 用戶端中 Subscriptions (訂閱) 下，您先前訂閱的主題旁會有一個綠點。

該綠點表示自上次查看後，已收到一個或多個新訊息。

b. 於 Subscriptions (訂閱) 下，選擇 device/+/data，來檢查訊息承載是否與您剛剛發佈的內容相符，如下所示：

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
```

```
  }
}
```

c. 檢查您用來訂閱 SNS 主題的手機，並確認訊息承載內容如下所示：

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

若您變更訊息主題中的主題 ID 元素，請記得，僅當訊息主題中的該元素僅包含數字字元時，將 `topic(2)` 值轉換為數值才會有效。

4. 請嘗試傳送溫度不超過限制的 MQTT 訊息。

a. 在 MQTT 用戶端中的 Subscriptions (訂閱) 下選擇 Publish to topic (發佈至主題)。

b. 在 Publish (發佈) 欄位中輸入輸入主題名稱 **`device/33/data`**。

c. 複製此處顯示的範例資料，並在主題名稱下方的編輯方塊中貼上範例資料。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 若要傳送 MQTT 訊息，請選擇 Publish to topic (發佈至主題)。

您應該會看到您在 **`device/+/data`** 訂閱中傳送的訊息；但因溫度值低於規則查詢陳述式中的最高溫度，您不應收到簡訊。

若您並未看到正確的行為，請查看疑難排解提示。

對 AWS Lambda 規則和通知進行故障診斷

若您並未看到預期的結果，請查看以下事項。

- 您收到錯誤的橫幅

若在您發佈輸入訊息時出現錯誤，請先更正該錯誤。下列步驟可協助您修正該錯誤。

- 您並未在 MQTT 用戶端中看到輸入訊息

  每次您將輸入訊息發佈至 device/32/data 主題中，若您依程序中所述訂閱了 device/+/data 主題篩選條件，則該訊息應會顯示於 MQTT 用戶端中。

  要檢查的事項

  - 檢查您訂閱的主題篩選條件

    若您依程序中所述訂閱了輸入訊息主題，則每次發佈輸入訊息時都應該會看到其複本。

    若您並未訊息，請檢查您訂閱的主題名稱，並將其與所發佈的主題進行比較。主題名稱區分大小寫，且您訂閱的主題必須與所發佈訊息承載的主題相同。

  - 檢查訊息發佈功能

    在 MQTT 用戶端中的 Subscriptions (訂閱) 下，選擇 device/+/data，檢查發佈訊息的主題，然後選擇 Publish to topic (發佈至主題)。您應該會在訊息清單中出現主題下方的編輯方塊中看到訊息承載。

- 您並未收到 SMS 訊息：

  若要讓您的規則運作，其必須具有授權其接收訊息和傳送 SNS 通知的正確政策，且必須接收訊息。

  要檢查的事項

  - 檢查 MQTT 用戶端 AWS 區域 的 和您建立的規則

    您正在執行 MQTT 用戶端的主控台必須與您建立的規則處於相同的 AWS 區域。

  - 檢查訊息承載中的溫度值是否超過測試閾值

    若溫度值小於或等於 30 (如規則查詢陳述式中所定義)，則規則將不會執行其任何動作。

  - 檢查規則查詢陳述式中的輸入訊息主題

    若要讓規則運作，其必須收到一則訊息，其主題名稱與規則查詢陳述式之 FROM 子句中的主題篩選條件相符。

    檢查規則查詢陳述式中主題篩選條件的拼字與 MQTT 用戶端中主題的拼字。主題名稱區分大小寫，且郵件的主題必須與規則查詢陳述式中的主題篩選條件相符。

  - 檢查輸入訊息承載的內容

    若要讓規則運作，其必須在 SELECT 陳述式中宣告的訊息承載中尋找資料欄位。

檢查規則查詢陳述式中 `temperature` 欄位的拼字與 MQTT 用戶端中訊息承載的拼字。欄位名稱區分大小寫，規則查詢陳述式中的 `temperature` 欄位必須與訊息承載中的 `temperature` 欄位相符。

請確定訊息承載中的 JSON 文件格式正確。若 JSON 有任何錯誤，例如缺少逗號，則規則將無法進行讀取。

- 檢查 Amazon SNS 通知

  於 步驟 1：建立傳送簡訊的 Amazon SNS 主題 中，請參閱步驟 3，說明如何測試 Amazon SNS 通知並測試通知以確保通知可正常運作。

- 檢查 Lambda 函數

  於 步驟 1：建立傳送文字訊息的 AWS Lambda 函數 中，請參閱步驟 5，說明如何使用測試資料來測試 Lambda 函數，並測試 Lambda 函數。

- 檢查規則所使用的角色

  規則動作必須具有接收原始主題及發佈新主題的權限。

  授權規則接收訊息資料的政策並加以重新發佈為所使用的主題所特定的。若變更用於重新發佈訊息資料的主題，則必須更新規則動作的角色，來更新其政策以與目前主題相符。

  若您懷疑這會是問題，請編輯 Republish (重新發佈) 規則動作並建立新角色。規則動作建立的新角色會收到執行這些動作所需的授權。

步驟 4：檢閱結果及後續步驟

於本教學課程中：

- 您已建立 AWS IoT 規則來呼叫傳送使用自訂訊息承載之 Amazon SNS 通知的 Lambda 函數。
- 您於規則查詢陳述式中使用了一個簡單的 SQL 查詢和函數，可為您的 Lambda 函數建立一個新訊息承載。
- 您使用 MQTT 用戶端來測試 AWS IoT 規則。

後續步驟

使用此規則傳送一些簡訊之後，請嘗試使用其來查看教學課程的某些層面如何影響訊息，及訊息傳送的時間。此處有幾種簡單的入門方式。

- 變更輸入訊息主題中的 *device_id*，並觀察簡訊內容中的影響。
- 變更規則查詢陳述式中所選取的欄位、更新 Lambda 函數以於新訊息中加以使用，並觀察簡訊內容中的影響。
- 變更規則查詢陳述式中的測試，以測試最低溫度，而非最高溫度。更新 Lambda 函數以格式化新的訊息，並記得變更 `max_temperature` 的名稱。
- 若要進一步了解如何尋找在您開發和使用 AWS IoT 規則時可能發生的錯誤，請參閱 監控 AWS IoT。

在裝置離線時保留裝置狀態

這些教學課程說明如何使用 AWS IoT Device Shadow 服務來存放和更新裝置的狀態資訊。Shadow 文件是一個 JSON 文件，其根據裝置、本機應用程式或服務發佈的訊息，顯示裝置狀態的變化。於本教學課程中，Shadow 文件會顯示燈泡顏色的變化。這些教學課程還會顯示影子如何在裝置與網際網路中斷連線時，儲存此資訊，並在裝置回復連線並請求此資訊時，將最新狀態資訊傳回裝置。

建議您依此處顯示的順序試試這些教學課程，從您需要建立的 AWS IoT 資源和必要的硬體設定開始，這亦可協助您逐步學習概念。這些教學課程示範如何設定和連接 Raspberry Pi 裝置以搭配 使用 AWS IoT。若您並無所需的硬體，您可依照這些教學課程進行調整，以適應您選擇的裝置或使用 Amazon EC2 建立虛擬裝置。

教學課程案例概觀

這些教學課程的案例為本機應用程式或服務，可變更燈泡的顏色，及將其資料發佈置預留的影子主題。這些教學課程類似於互動式入門教學課程中說明的 Device Shadow 功能，並在 Raspberry Pi 裝置實作。本節中的教學課程側重於單一經典影子裝置，同時展現如何容納已命名影子或多個裝置的方法。

下列教學課程將協助您了解如何使用 AWS IoT Device Shadow 服務。

- 教學課程：準備好 Raspberry Pi 來執行影子應用程式

  本教學課程說明如何設定 Raspberry Pi 裝置以進行連線 AWS IoT。您也將建立 AWS IoT 政策文件和物件資源、下載憑證，然後將政策連接到該物件資源。此教學課程約需 30 分鐘方能完成。
- 教學課程：安裝裝置 SDK 並執行 Device Shadows 的範例應用程式

  本教學課程說明如何安裝必要的工具、軟體和適用於 Python 的 AWS IoT Device SDK，然後執行範例陰影應用程式。本教學課程以 連接 Raspberry Pi 或其他裝置 中提出的概念為基礎，需 20 分鐘才能完成。
- 教學課程：使用範例應用程式和 MQTT 測試用戶端，與 Device Shadow 互動

本教學課程示範如何使用shadow.py範例應用程式和AWS IoT 主控台來觀察 AWS IoT Device Shadows 與燈泡狀態變更之間的互動。本教學課程也會展示如何將 MQTT 訊息傳送至 Device Shadow 的預留主題。此教學課程約需 45 分鐘方能完成。

AWS IoT Device Shadow 概觀

Device Shadow 是由您在 AWS IoT 登錄檔中建立的物件資源所管理之裝置的持久性虛擬表示法。Shadow 文件是個 JSON 或 JavaScript 標記法文件，用來存儲和檢索裝置的目前狀態資訊。您可透過 MQTT 或 HTTP REST API，使用影子來取得及設定裝置的狀態 (無論該裝置是否連線至網際網路)。

Shadow 的文件包含 `state` 屬性，說明裝置狀態的下列層面：

- `desired`：應用程式會透過更新 `desired` 物件來指定裝置屬性的所需狀態。
- `reported`：裝置會報告其在 `reported` 物件中的目前狀態。
- `delta`： AWS IoT 報告 `delta` 物件中所需狀態與報告狀態之間的差異。

以下為 Shadow 狀態文件的範例：

```
{
  "state": {
    "desired": {
      "color": "green"
      },
    "reported": {
      "color": "blue"
      },
    "delta": {
      "color": "green"
      }
   }
}
```

如要更新裝置的 Shadow 文件，您可使用預留的 MQTT 主題、以 HTTP 支援 GET、UPDATE 和 DELETE 操作的 Device Shadow REST API，以及 AWS IoT CLI。

在上一個範例中，假設您想將 `desired` 顏色變更為 `yellow`。如要執行此動作，請傳送請求至 UpdateThingShadow API 或將訊息發佈至更新主題 `$aws/things/THING_NAME/shadow/update`。

```
{
  "state": {
    "desired": {
      "color": yellow
    }
  }
}
```

更新只會影響請求中所指定的欄位。成功更新 Device Shadow 後， 會將新`desired`狀態 AWS IoT 發佈至`delta`主題 `$aws/things/THING_NAME/shadow/delta`。於此狀況下，Shadow 文件看起來像這樣：

```
{
  "state": {
    "desired": {
      "color": yellow
    },
    "reported": {
      "color": green
    },
    "delta": {
      "color": yellow
      }
  }
}
```

接著會使用`$aws/things/THING_NAME/shadow/update`具有下列 JSON 訊息`Update`的主題，向 AWS IoT Device Shadow 報告新狀態：

```
{
  "state": {
    "reported": {
      "color": yellow
    }
  }
}
```

若您想要取得目前的狀態資訊，請將請求傳送至 GetThingShadow API 或將 MQTT 訊息發佈至取得主題 `$aws/things/THING_NAME/shadow/get`。

如需使用 Device Shadow 服務的相關資訊，請參閱 AWS IoT Device Shadow 服務。

如需在裝置、應用程式和服務中使用 Device Shadows 的詳細資訊，請參閱 在裝置中使用影子 和 在應用程式和服務中使用影子。

如需與 AWS IoT 陰影互動的資訊，請參閱 與影子互動。

如需 MQTT 預留主題和 HTTP REST API 的相關資訊，請參閱 Device Shadow MQTT 主題 和 Device Shadow REST API。

教學課程：準備好 Raspberry Pi 來執行影子應用程式

本教學課程示範如何設定 Raspberry Pi 裝置，以及建立裝置連線和交換 MQTT 訊息所需的 AWS IoT 資源。

> **Note**
>
> 若您打算 the section called “使用 Amazon EC2 建立虛擬裝置”，則可跳過本頁，並繼續 the section called “設定您的裝置”。當您建立虛擬物件時，您將會建立這些資源。若您想使用不同的裝置，而非 Raspberry Pi，您可嘗試依照這些教學課程進行調整，使其適應您選擇的裝置。

於本教學課程中，您會了解如何：

- 設定 Raspberry Pi 裝置並將其設定為搭配 使用 AWS IoT。
- 建立 AWS IoT 政策文件，授權您的裝置與服務 AWS IoT 互動。
- AWS IoT 在 X.509 裝置憑證中建立物件資源，然後連接政策文件。

  問題是您的裝置在 AWS IoT 登錄檔中的虛擬表示。憑證會將您的裝置驗證為 AWS IoT Core，而政策文件會授權您的裝置與之互動 AWS IoT。

如何執行本教學課程

如要執行 Device Shadows 的 `shadow.py` 範例應用程式，您需要一個連接至 AWS IoT的 Raspberry Pi 裝置。我們建議您依照此處顯示的順序學習本教學課程，從設定 Raspberry Pi 及其配件開始，然後建立政策，並將政策連接至您建立的物件資源。然後，您可以使用 Raspberry Pi 支援的圖形使用者介面 (GUI) 來遵循本教學課程，在裝置的 Web 瀏覽器上開啟 AWS IoT 主控台，這也可以讓您更輕鬆地直接將憑證下載到 Raspberry Pi 以進行連線 AWS IoT。

開始本教學課程之前，請確定您有：

- AWS 帳戶。若您沒有帳戶，請完成 設定 AWS 帳戶 所述的步驟，然後再繼續。您需要 AWS 帳戶和 AWS IoT 主控台才能完成本教學課程。
- Raspberry Pi 及其必要的配件。您會需要：
  - Raspberry Pi 3 代 B 型或更新的型號。本教學課程可能適用於較早版本的 Raspberry Pi，但我們尚未對其進行測試。
  - Raspberry Pi OS (32 位元) 或更新版本。我們建議您使用最新版本的 Raspberry Pi 作業系統。較早版本的作業系統可能適用，但我們尚未對其進行測試。
  - 乙太網路或 Wi-Fi 連線。
  - 鍵盤、滑鼠、顯示器、纜線和電源供應器。

此教學課程約需 30 分鐘方能完成。

步驟 1：設定及配置 Raspberry Pi 裝置

在本節中，我們將設定 Raspberry Pi 裝置以搭配 使用 AWS IoT。

> **Important**
>
> 調整這些指示以適用其他裝置和作業系統可能是一項艱鉅的挑戰。您必須充分了解您的裝置，才能解譯這些指示並將其套用至您的裝置。若遇到困難，您可嘗試使用其他裝置選項的一項作為替代選項，例如 使用 Amazon EC2 建立虛擬裝置 或 使用您的 Windows 或 Linux PC 或 Mac 做為 AWS IoT 裝置。

您需要配置 Raspberry Pi，使其可以啟動作業系統 (OS)，連接至網際網路，並可讓您在命令列介面與其互動。您也可以使用 Raspberry Pi 支援的圖形使用者介面 (GUI) 來開啟 AWS IoT 主控台並執行本教學課程的其餘部分。

設定 Raspberry Pi

1. 將 SD 卡插入 Raspberry Pi 上的 MicroSD 記憶卡插槽。有些 SD 卡會預先載入安裝管理員，在啟動主機板後提示您安裝作業系統的選單。您也可使用 Raspberry Pi 成像器，在卡上安裝作業系統。
2. 將 HDMI 電視或顯示器連接至連接到 Raspberry Pi 之 HDMI 連接埠的 HDMI 纜線。
3. 將鍵盤和滑鼠連接至 Raspberry Pi 的 USB 連接埠，然後插入電源整流器以啟動機板。

Raspberry Pi 啟動後，若 SD 卡預先載入安裝管理員，會出現一個安裝作業系統的選單。若於安裝作業系統時發生問題，您可試試下列步驟。如需有關 Raspberry Pi 的設定資訊，請參閱設定 Raspberry Pi。

若您在設定 Raspberry Pi 時發生問題：

- 在啟動機板之前，請檢查是否已插入 SD 卡。若您在啟動機板後插入 SD 卡，則安裝選單可能不會顯示。
- 確認電視或顯示器已開啟，且已選取正確的輸入。
- 確保您正在使用 Raspberry Pi 相容的軟體。

安裝並設定 Raspberry Pi 作業系統之後，請開啟 Raspberry Pi 的 Web 瀏覽器，然後導覽至 AWS IoT Core 主控台以繼續本教學課程的其餘步驟。

如果您可以開啟 AWS IoT Core 主控台，表示 Raspberry Pi 已準備就緒，您可以繼續 the section called “在 中佈建您的裝置 AWS IoT ”。

若您仍然無法解決問題或需要其他協助，請參閱獲取有關 Raspberry Pi 的協助。

教學課程：在 中佈建您的裝置 AWS IoT

本節會建立教學課程將使用 AWS IoT Core 的資源。

在 中佈建裝置的步驟 AWS IoT

- 步驟 1：建立 Device Shadow AWS IoT 的政策
- 步驟 2：建立物件資源並將政策連接至該物件
- 步驟 3：檢閱結果及後續步驟

步驟 1：建立 Device Shadow AWS IoT 的政策

X.509 憑證會以 . AWS IoT policies 驗證您的裝置 AWS IoT Core。此憑證會連接至允許裝置執行 AWS IoT 操作的憑證，例如訂閱或發佈至 Device Shadow 服務所使用的 MQTT 預留主題。您的裝置會在連線並傳送訊息時顯示其憑證 AWS IoT Core。

在此過程中，您將會建立一個政策，可讓您的裝置執行 AWS IoT 執行範例程式所需的作業。建議您建立政策，該政策僅授予執行任務所需的許可權。首先建立 AWS IoT 政策，然後將其連接到稍後建立的裝置憑證。

建立 AWS IoT 政策

1. 請在左側選單上選擇 Secure (安全)，然後選擇 Policies (政策)。若您的帳戶具有現有政策，請選擇 Create (建立)，否則，在 You don't have a policy yet (您尚未設定政策) 頁面上，選擇 Create a policy (建立政策)。
2. 在 Create policy (建立政策) 頁面上：

   a. 在 Name (名稱) 欄位中，輸入政策的名稱 (例如，**My_Device_Shadow_policy**)。請勿在政策名稱中使用個人識別資訊。

   b. 在政策文件中，您說明了連線、訂閱、接收和發佈動作，這些動作授予裝置發佈和訂閱 MQTT 預留主題的許可權限。

      複製下列範例政策，並將其貼入您的政策文件中：`thingname` 將 取代為您建立的物件名稱 （ 例如 `My_light_bulb`)、`region`將 取代為您使用 服務的 AWS IoT 區域，並將 `account`取代為您的 AWS 帳戶 號碼。如需 AWS IoT 政策的詳細資訊，請參閱 AWS IoT Core 政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/accepted",
```

```
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/test-*"
        }
    ]
}
```

步驟 2：建立物件資源並將政策連接至該物件

連線至 的裝置 AWS IoT 可由 AWS IoT 登錄檔中的物件資源表示。物件資源意指特定裝置或邏輯實體，例如本教學課程中的燈泡。

若要了解如何在 中建立物件 AWS IoT，請遵循中所述的步驟建立物件。當您依照該教學課程中的步驟時，請注意以下一些關鍵事項：

1. 選擇 Create a single thing (建立單一物件)，在 Name (名稱) 欄位中，輸入與您先前建立政策時指定之 `thingname` 相同的物件名稱 (例如，`My_light_bulb`)。

   物件類型建立之後，您就無法變更其名稱。若您給它一個不同於 `thingname` 的名字，請建立名為 `thingname` 的新物件並刪除舊物件。

   > **Note**
   >
   > 請勿在物件名稱中使用個人識別資訊。物件名稱可以出現在未加密的通訊和報告中。

2. 我們建議您將憑證已建立！頁面上的每個憑證檔案下載至您可輕鬆找到的位置。您必須安裝這些檔案，才能執行範例應用程式。

   建議您將檔案下載至 Raspberry Pi home 目錄中的 `certs` 子目錄上，並使用如下表中所建議更簡單名稱對其進行命名。

   憑證檔案名稱

檔案	檔案路徑
根 CA 憑證	`~/certs/Amazon-root-CA-1.pem`
裝置憑證	`~/certs/device.pem.crt`
私有金鑰	`~/certs/private.pem.key`

3. 啟用憑證以啟用連線後 AWS IoT，請選擇連接政策，並確保您將先前建立的政策 （ 例如 **`My_Device_Shadow_policy`**) 連接到物件。

   建立物件之後，您可以在 AWS IoT 主控台的物件清單中看到您的物件資源。

步驟 3：檢閱結果及後續步驟

在本教學課程中，您會了解如何：

- 設定及配置 Raspberry Pi 裝置。
- 建立 AWS IoT 政策文件，授權您的裝置與 AWS IoT 服務互動。

- 建立物件資源和相關聯的 X.509 裝置憑證，並將政策文件與其進行連接。

後續步驟

您現在可以安裝適用於 Python AWS IoT 的裝置 SDK、執行`shadow.py`範例應用程式，並使用 Device Shadows 控制狀態。如需如何執行本教學課程的詳細資訊，請參閱 教學課程：安裝裝置 SDK 並執行 Device Shadows 的範例應用程式。

教學課程：安裝裝置 SDK 並執行 Device Shadows 的範例應用程式

本節說明如何安裝必要的軟體和適用於 Python 的 AWS IoT Device SDK，並執行`shadow.py`範例應用程式來編輯 Shadow 文件並控制影子的狀態。

於本教學課程中，您會了解如何：

- 使用已安裝的軟體和適用於 Python 的 AWS IoT Device SDK 來執行範例應用程式。
- 了解如何使用範例應用程式輸入值，以於 AWS IoT 主控台中發佈所需的值。
- 檢閱 `shadow.py` 範例應用程式，及其如何使用 MQTT 通訊協定來更新影子的狀態。

在您執行此教學課程之前：

您必須已設定 AWS 帳戶、設定 Raspberry Pi 裝置，並建立 AWS IoT 物件和政策，以授予裝置發佈和訂閱 Device Shadow 服務 MQTT 預留主題的許可。如需詳細資訊，請參閱教學課程：準備好 Raspberry Pi 來執行影子應用程式。

您還必須安裝 Git、Python 和適用於 Python 的 AWS IoT 裝置 SDK。本教學課程是以教學課程 連接 Raspberry Pi 或其他裝置 中提出的概念為基礎。若您尚未嘗試該教學課程，建議您依照該教學課程中說明的步驟，安裝憑證檔案和裝置 SDK，然後回到本教學課程，執行 `shadow.py` 範例應用程式。

於本教學課程中，您將會：

- 步驟 1：執行 shadow.py 範例應用程式
- 步驟 2：檢閱 shadow.py 裝置 SDK 範例應用程式
- 步驟 3：使用 shadow.py 範例應用程式進行疑難排解
- 步驟 4：檢閱結果及後續步驟

此教學課程約需 20 分鐘方能完成。

步驟 1：執行 shadow.py 範例應用程式

執行 shadow.py 範例應用程式之前，除了您所安裝之憑證檔案的名稱和位置之外，您還需要下列資訊。

應用程式參數值

參數	可在哪裡找到值
your-iot-thing-name	您先前在 中建立的 AWS IoT 物件名稱the section called “步驟 2：建立物件資源並將政策連接至該物件”。 如要尋找此值，請於 AWS IoT 主控台中，依序選擇 Manage (管理) 和 Things (物件)。
your-iot-endpoint	*your-iot-endpoint* 值的格式為：*endpoint_id* -ats.iot. *region*.amazonaws.com ，例如 a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com 。如要尋找此值： 1. 於 AWS IoT 主控台中，依序選擇 Manage (管理) 和 Things (物件)。 2. 選擇您先前使用為裝置 My_light_bulb 建立的 IoT，然後選擇 Interact (互動)。在物件詳細資料頁面上，您的端點會顯示於 HTTPS 區段中。

安裝並執行範例應用程式

1. 導覽至範例應用程式目錄。

```
cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
```

2. 在命令列視窗中，按照指示替換 *your-iot-endpoint* 和 *your-iot-thing-name* 並執行此命令。

```
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

3. 觀察範例應用程式：

   1. 連線至您帳戶的 AWS IoT 服務。
   2. 訂閱 `Delta` 事件和 `Update` 與 `Get` 回應。
   3. 提示您在終端機終輸入所需的值。
   4. 顯示類似下列內容的輸出：

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow contains reported value 'off'.
Enter desired value:
```

> **Note**
>
> 若在執行 `shadow.py` 範例應用程式時發生問題，請檢閱 the section called “步驟 3：使用 shadow.py 範例應用程式進行疑難排解”。如要取得可協助您修正問題的其他資訊，請將 `--verbosity debug` 參數新增至命令列，以便範例應用程式會顯示有關其正在執行之動作的詳細訊息。

輸入值並觀察 Shadow 文件中的更新

您可於終端機中輸入值，以指定 `desired` 值，其還會更新 `reported` 值。假設您在終端機中輸入顏色 `yellow`。`reported` 值也會更新為顏色 `yellow`。下列展示顯示於終端機中的訊息：

```
Enter desired value:
```

```
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

當您發佈此更新請求時， 會為物件資源 AWS IoT 建立預設、傳統陰影。您可以查看您建立之物件資源的 Shadow 文件 （ 例如，)，以觀察您發佈至 AWS IoT 主控台中 `reported`和 `desired`值的更新請求`My_light_bulb`。如要在 Shadow 文件中查看更新：

1. 在 AWS IoT 主控台中，選擇管理，然後選擇實物。
2. 在顯示的物件清單中，選取您所建立的物件，然後依序選擇 Shadows (影子) 和 Classic Shadow (經典影子)。

Shadow 文件應看似如下列內容，顯示設定為顏色 `yellow` 的 `reported` 和 `desired` 值。您會在文件的 Shadow state (影子狀態) 區段中看到這些值。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "yellow"
},
"reported": {
  "welcome": "aws-iot",
  "color": "yellow"
}
}
```

您還會看到 Metadata (中繼資料) 區段，其包含請求之時間戳記資訊和版本號碼。

您可以使用狀態文件版本，確認您正在更新的裝置影子文件為最新版本。若您傳送另一個更新請求，版本編號會遞增 1。當您為更新請求提供版本，若狀態文件的目前版本與提供的版本不符，則服務會拒絕請求並顯示 HTTP 409 衝突回應代碼。

```
{
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
```

```
    "color": {
      "timestamp": 1620156893
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  }
},
"version": 10
}
```

如要了解 Shadow 文件的更多資訊並觀察狀態資訊的變更，請繼續參閱說明於本教學課程 步驟 4：檢閱結果及後續步驟 部分中的下一個教學課程 教學課程：使用範例應用程式和 MQTT 測試用戶端，與 Device Shadow 互動。或者，您還可於下一節中了解 `shadow.py` 範本程式碼及其如何使用 MQTT 通訊協定。

步驟 2：檢閱 shadow.py 裝置 SDK 範例應用程式

本節會從用於本教學課程中之適用於 Python 的AWS IoT 裝置 SDK v2 檢閱 `shadow.py` 範例應用程式。在這裡，我們將 AWS IoT Core 使用 MQTT 和透過 WSS 通訊協定的 MQTT 來檢閱它如何連接到 。AWS 通用執行時間 (AWS-CRT) 程式庫提供低階通訊協定支援，並包含在適用於 Python 的 AWS IoT Device SDK v2 中。

雖然本教學課程透過 WSS 使用 MQTT 和 MQTT， AWS IoT 支援發佈 HTTPS 請求的裝置。有關從裝置傳送 HTTP 訊息的 Python 程式範例，請參閱使用 Python `requests` 程式庫的 HTTPS 程式碼範例。

如需有關如何決定裝置通訊所使用之通訊協定的詳細資訊，請參閱 選擇裝置通訊的應用程式通訊協定。

MQTT

`shadow.py` 範例呼叫 `mqtt_connection_builder` 中的 `mtls_from_path` (如圖所示)，使用 MQTT 通訊協定來建立與 AWS IoT Core 的連線。`mtls_from_path` 會使用 X.509 憑證和 TLS v1.2 來驗證裝置。 AWS CRT 程式庫會處理該連線的較低層級詳細資訊。

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
```

```
  endpoint=args.endpoint,
  cert_filepath=args.cert,
  pri_key_filepath=args.key,
  ca_filepath=args.ca_file,
  client_bootstrap=client_bootstrap,
  on_connection_interrupted=on_connection_interrupted,
  on_connection_resumed=on_connection_resumed,
  client_id=args.client_id,
  clean_session=False,
  keep_alive_secs=6
)
```

- endpoint 是您從命令列傳入的 AWS IoT 端點，client_id也是 中唯一識別此裝置的 ID AWS 區域。
- cert_filepath、pri_key_filepath 和 ca_filepath 是裝置憑證和私密金鑰檔案及根 CA 檔案的路徑。
- client_bootstrap 是處理通訊端通訊活動的通用執行時間物件，並在呼叫 mqtt_connection_builder.mtls_from_path 之前實例化。
- on_connection_interrupted 和 on_connection_resumed 是在裝置連線遭到中斷並回復時呼叫的回呼函數。
- clean_session 為是否啟動一個新的、持續的工作階段，或若存在，則進行重新連線。keep_alive_secs 為 CONNECT 請求中傳送的保持活動值，以秒為單位。Ping 會在此時間間隔自動傳送。若在此值的 1.5 倍之後未收到 ping，則伺服器會假設連線中斷。

shadow.py 範例還會呼叫 mqtt_connection_builder 中的 websockets_with_default_aws_signing，使用透過 WSS 的 MQTT 通訊協定建立與 AWS IoT Core 的連線。透過 WSS 的 MQTT 還會使用與 MQTT 相同的參數，並採用下列其他參數：

- region 是 Signature V4 身分驗證所使用的 AWS 簽署區域，credentials_provider也是提供用於身分驗證的 AWS 登入資料。從命令列傳入 Region (區域)，並在呼叫 mqtt_connection_builder.websockets_with_default_aws_signing 之前將 credentials_provider 物件實例化。
- 若使用代理主機，websocket_proxy_options 是 HTTP 代理選項。在 shadow.py 範例應用程式中，此值在呼叫 mqtt_connection_builder.websockets_with_default_aws_signing 之前進行實例化。

訂閱 Shadow 主題和事件

shadow.py 範例嘗試建立連線，並等待完整連線。若未連線，則指令會排入佇列中。連線後，範例會訂閱差異事件及更新和取得訊息，並發佈服務品質 (QoS) 層級為 1 (mqtt.QoS.AT_LEAST_ONCE) 的訊息。

當裝置訂閱 QoS 層級 1 的訊息時，訊息代理程式會儲存裝置訂閱的訊息，直至其可傳送至裝置為止。訊息代理程式會重新傳送訊息，直至收到來自裝置的 PUBACK 回應。

如需 MQTT 通訊協定的詳細資訊，請參閱 檢閱 MQTT 通訊協定 和 MQTT。

如需用於本教學課程中的 MQTT、透過 WSS 的 MQTT、持續性工作階段及 QoS 層級的詳細資訊，請參閱 檢閱 pubsub.py 裝置 SDK 範例應用程式。

步驟 3：使用 **shadow.py** 範例應用程式進行疑難排解

當您執行 shadow.py 範例應用程式時，您應會看到一些顯示於終端機中的訊息，並提示輸入 desired 值。若程式拋出錯誤，則如要偵錯，您可先檢查是否為您的系統執行正確的命令。

在某些狀況下，錯誤訊息可能表示連線問題，且看似如下：Host name was invalid for dns resolution 或 Connection was closed unexpectedly。於此狀況下，您可檢查下列事項：

- 檢查命令中的端點的地址

  請檢閱您為執行範例應用程式 (例如 a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com) 輸入命令中的 endpoint 引數，並在 AWS IoT 主控台 中檢查此值。

  如要檢查您是否使用了正確的值：

  1. 於 AWS IoT 主控台中，依序選擇 Manage (管理) 和 Things (物件)。
  2. 選擇您為範例應用程式建立的物件 (例如，My_light_bulb)，然後選擇 Interact (互動)。

  在物件詳細資料頁面上，您的端點會顯示於 HTTPS 區段中。您應該也會看到訊息：This thing already appears to be connected.
- 檢查憑證啟用

  憑證會使用 驗證您的裝置 AWS IoT Core。

  如要檢查您的憑證是否為作用中：

  1. 於 AWS IoT 主控台中，依序選擇 Manage (管理) 和 Things (物件)。
  2. 選擇您為範例應用程式建立的物件 (例如，My_light_bulb)，然後選擇 Security (安全性)。
  3. 選取憑證，然後從憑證的詳細資料頁面選擇 Select the certificate (選取憑證)，然後從憑證的詳細資料頁面選擇 Actions (動作)。

若於下拉式清單中 Activate (啟用) 無法使用，則您只能選擇 Deactivate(停用)，表示您的憑證處於作用中。若無，請選擇 Activate (啟用)，然後重新執行範例程式。

若程式仍然無法執行，請檢查 `certs` 資料夾中的憑證檔案名稱。

- 檢查連接至該物件資源的政策

  當憑證驗證您的裝置時， AWS IoT 政策會允許裝置執行 AWS IoT 操作，例如訂閱或發佈至 MQTT 預留主題。

  如要檢查是否已連接正確的政策：

  1. 如先前所述尋找憑證，然後選擇 Policies (政策)。
  2. 選擇顯示的政策，並檢查其是否說明了授予裝置發佈和訂閱 MQTT 預留主題之許可權限的 `connect`、`subscribe`、`receive` 和 `publish` 動作。

     若是範例政策，請參閱 步驟 1：建立 Device Shadow AWS IoT 的政策。

  如果您看到錯誤訊息指出無法連線至 AWS IoT，這可能是因為您用於政策的許可。如果是這種情況，我們建議您從提供 AWS IoT 資源完整存取權的政策開始，然後重新執行範例程式。您可編輯目前的政策，或選擇目前的政策，選擇 Detach (分離)，然後建立另一個提供完整存取權的政策，並將其連接至您的物件資源。您可於稍後將政策限制為僅執行程式所需的動作和政策。

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:*"
            ],
            "Resource": "*"
        }
    ]
  }
  ```

- 檢查您的裝置 SDK 安裝

  若程式仍然無法執行，您可重新安裝裝置 SDK，以確定您的 SDK 安裝完整且正確。

步驟 4：檢閱結果及後續步驟

在本教學課程中，您會了解如何：

- 安裝必要的軟體、工具和適用於 Python 的 AWS IoT 裝置 SDK。
- 了解範例應用程式 `shadow.py` 會如何使用 MQTT 通訊協定來擷取和更新影子的目前狀態。
- 執行 Device Shadows 的範例應用程式，並在 AWS IoT 主控台中觀察 Shadow 文件的更新。您還會了解在執行程式時如何疑難排解任何問題並修復錯誤。

後續步驟

您現在可以執行 `shadow.py` 範例應用程式，並使用 Device Shadows 來控制狀態。您可以觀察 AWS IoT 主控台中 Shadow 文件的更新，並觀察範例應用程式回應的差異事件。使用 MQTT 測試用戶端，您可以訂閱預留的影子主題，並在執行範例程式時觀察主題所收到的訊息。如需如何執行本教學課程的詳細資訊，請參閱 教學課程：使用範例應用程式和 MQTT 測試用戶端，與 Device Shadow 互動。

教學課程：使用範例應用程式和 MQTT 測試用戶端，與 Device Shadow 互動

如要與 `shadow.py` 範例應用程式產生互動，請為 `desired` 值在終端機中輸入一個值。例如，您可以指定類似流量燈的顏色，並 AWS IoT 回應請求並更新報告的值。

於本教學課程中，您會了解如何：

- 使用 `shadow.py` 範例應用程式來指定所需的狀態並更新影子的目前狀態。
- 編輯 Shadow 文件，以觀察差異事件，及 `shadow.py` 範例應用程序如何加以回應。
- 使用 MQTT 測試用戶端訂閱影子主題，並在執行範例程式時觀察更新。

執行本教學課程之前，您必須具備：

設定您的 AWS 帳戶、設定您的 Raspberry Pi 裝置，並建立 AWS IoT 物件和政策。您也必須安裝了必要的軟體、裝置 SDK、憑證檔案，並在終端機中執行範例程式。如需詳細資訊，請參閱先前的教學課程 教學課程：準備好 Raspberry Pi 來執行影子應用程式 和 步驟 1：執行 shadow.py 範例應用程式。若您尚未完成，您必須完成這些教學課程。

於本教學課程中，您將會：

- 步驟 1：使用 shadow.py 範例應用程式更新所需和回報的值
- 步驟 2：檢視 MQTT 測試用戶端中 shadow.py 範例應用程式的訊息

- 步驟 3：疑難排解 Device Shadow 互動的錯誤
- 步驟 4：檢閱結果及後續步驟

此教學課程約需 45 分鐘方能完成。

步驟 1：使用 **shadow.py** 範例應用程式更新所需和回報的值

在先前的教學課程中步驟 1：執行 shadow.py 範例應用程式，您已了解如何在輸入所需的值時，觀察在 AWS IoT 主控台中發佈至 Shadow 文件的訊息，如 一節所述教學課程：安裝裝置 SDK 並執行 Device Shadows 的範例應用程式。

在上一個範例中，我們將所需的顏色設定為 yellow。輸入每個值後，終端機會提示您輸入另一個 desired 值。若您再次輸入相同的值 (yellow)，應用程序會辨識這一點，並提示您輸入新的 desired 值。

```
Enter desired value:
yellow
Local value is already 'yellow'.
Enter desired value:
```

現在，假設您輸入顏色 green。 會 AWS IoT 回應請求，並將reported值更新為 green。當 desired 狀態與 reported 狀態不同而造成差異時，這就是更新發生了。

shadow.py 範例應用程式如何模擬 Device Shadow 互動：

1. 於終端機中輸入 desired 值 (例如 yellow)以發布所需的狀態。
2. 由於 desired 狀態與 reported 狀態 (例如，顏色 green) 不同，就會發生差異，而訂閱差異的應用程式會收到這個訊息。
3. 應用程式會回應訊息，並將其狀態更新為 desired 值，yellow。
4. 然後，應用程式會發佈更新訊息，其包含裝置狀態 yellow 的新回報值。

下列展示了終端機中顯示的訊息，其中顯示如何發佈更新請求。

```
Enter desired value:
green
Changed local shadow value to 'green'.
Updating reported shadow value to 'green'...
Update request published.
```

```
Finished updating reported shadow value to 'green'.
```

在 AWS IoT 主控台中，Shadow 文件會將 `reported`和 `desired` 欄位`green`的更新值反映至 ，而版本編號會遞增 1。例如，若先前的版本編號顯示為 10，則目前的版本編號會顯示為 11。

> **Note**
>
> 刪除影子不會將版本編號重置為 0。當您發佈更新請求或建立另一個具相同名稱的陰子時，您會看到陰子版本會遞增 1。

編輯 Shadow 文件以觀察差異事件

`shadow.py` 範例應用程式也會訂閱 `delta` 事件，並在 `desired` 值發生變更時加以回應。例如，您可將 `desired` 值變更為顏色 `red`。若要這樣做，請在 AWS IoT 主控台中按一下編輯來編輯 Shadow 文件，然後在 JSON `red` 中將`desired`值設定為 ，同時將`reported`值保留為 `green`。儲存變更之前，請讓 Raspberry Pi 上的終端機保持開啟狀態，因為您會在變更發生時看到顯示於終端機中的訊息。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

儲存新值之後，`shadow.py` 範例應用程式會回應此變更，並在終端機中顯示指出差異的訊息。接著，您應該會在輸入 `desired` 值的提示下方看到下列訊息。

```
Enter desired value:
Received shadow delta event.
Delta reports that desired value is 'red'. Changing local value...
Changed local shadow value to 'red'.
Updating reported shadow value to 'red'...
Finished updating reported shadow value to 'red'.
Enter desired value:
Update request published.
Finished updating reported shadow value to 'red'.
```

步驟 2：檢視 MQTT 測試用戶端中 **shadow.py** 範例應用程式的訊息

您可使用 AWS IoT 主控台中的 MQTT 測試用戶端，監控傳入您 AWS 帳戶中的 MQTT 訊息。訂閱 Device Shadow 服務使用的預留 MQTT 主題，您可在執行範例應用程式時觀察主題收到的訊息。

若您尚未使用 MQTT 測試用戶端，您可檢閱 使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息。這有助於您了解如何使用 AWS IoT 主控台中的 MQTT 測試用戶端，在 MQTT 訊息通過訊息代理程式時，檢視訊息。

1. 開啟 MQTT 測試用戶端

   在新視窗開啟 AWS IoT 主控台中的 MQTT 測試用戶端，如此您便可觀察 MQTT 主題所收到的訊息，不會遺失 MQTT 測試用戶端的組態。若您讓 MQTT 用戶端前往主控台中的另一個頁面，其不會保留任何訂閱或訊息記錄。在教學課程的本節中，您可以讓 AWS IoT 物件的 Shadow 文件和 MQTT 測試用戶端在不同的視窗中開啟，以更輕鬆地觀察與 Device Shadow 的互動。

2. 訂閱 MQTT 預留的 Shadow 主題

   您可使用 MQTT 測試用戶端輸入 Device Shadow 之 MQTT 預留主題的名稱，並加以訂閱，以於執行 `shadow.py` 範例應用程式時收到更新。如要訂閱主題：

   a. 於 AWS IoT 主控台的 MQTT 測試用戶端中，選擇 Subscribe to a topic (訂閱主題)。

   b. 於 主題篩選條件區段中，輸入：$aws/things/***thingname***/shadow/update/#。在此，`thingname` 是您先前所建立物件資源的名稱 (例如，`My_light_bulb`)。

   c. 保留其他組態設定的預設值，然後選擇 Subscribe (訂閱)。

   在主題訂閱中使用 # 萬用字元，您可同時訂閱多個 MQTT 主題，並在單一視窗中觀察裝置及其 Shadow 間交換的所有訊息。如需萬用字元及其用法的詳細資訊，請參閱 MQTT 主題。

3. 執行 **shadow.py** 範例程式並觀察訊息

   在 Raspberry Pi 的命令列視窗中，若您已中斷程式的連線，請再次執行範例應用程式，並於 AWS IoT 主控台中查看 MQTT 測試用戶端中的訊息。

   a. 執行下列命令以重新啟動範例程式。將 *your-iot-thing-name* 和 *your-iot-endpoint* 取代為您先前建立的 AWS IoT 物件名稱 （ 例如 `My_light_bulb`)，以及要與裝置互動的端點。

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
   ```

```
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

shadow.py 範例應用程式接著會執行並擷取目前的影子狀態。若您已刪除影子或清除目前狀態，該程式會將目前值設定為 off，然後提示您輸入 desired 值。

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow document lacks 'color' property. Setting defaults...
Changed local shadow value to 'off'.
Updating reported shadow value to 'off'...
Update request published.
Finished updating reported shadow value to 'off'...
Enter desired value:
```

另一方面，若程式正在執行，而您將其重新啟動，則會在終端機中看到回報的最新顏色值。於 MQTT 測試用戶端中，您會看到主題更新 $aws/things/***thingname***/shadow/get 和 $aws/things/***thingname***/shadow/get/accepted。

假設回報的最新顏色是 green。下列會顯示 $aws/things/***thingname***/shadow/get/accepted JSON 檔案的內容。

```
{
"state": {
  "desired": {
    "welcome": "aws-iot",
    "color": "green"
  },
  "reported": {
    "welcome": "aws-iot",
    "color": "green"
  }
},
```

```
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  }
},
"version": 10,
"timestamp": 1620173908
}
```

b. 於終端機中輸入 desired 值，例如 yellow。shadow.py 範例應用程式會在終端機中回應並顯示下列訊息，顯示 reported 值變更為 yellow。

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

於 AWS IoT console (主控台) MQTT test client (MQTT 測試用戶端) 的 Subscriptions (訂閱) 之下，您會看到下列主題收到訊息：

- $aws/things/***thingname***/shadow/update：顯示 desired 和 updated 值會變更為顏色 yellow。
- $aws/things/***thingname***/shadow/update/accepted：顯示 desired 和 reported 狀態的目前值及其中繼資料和版本資訊。

- $aws/things/***thingname***/shadow/update/documents：顯示 `desired` 和 `reported` 狀態的先前的值和目前值及其中繼資料和版本資訊。

作為文件 $aws/things/***thingname***/shadow/update/documents 也包含其他兩個主題中所包含的資訊，我們可以加以檢閱以查看狀態資訊。先前狀態會顯示設定為 `green` 的回報值、其中繼資料和版本資訊，目前狀態會顯示已更新為 `yellow` 的回報值。

```
{
"previous": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "green"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "green"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    }
  },
  "version": 10
},
"current": {
  "state": {
    "desired": {
```

```
      "welcome": "aws-iot",
      "color": "yellow"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "yellow"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
    }
  },
  "version": 11
},
"timestamp": 1617297904
}
```

c. 若您現在輸入另一個 desired 值，您會看到這些主題所收到 reported 值和訊息更新的進一步變更。版本編號也會遞增 1。例如，若您輸入值 green 時，先前的狀態會報告值 yellow，而目前狀態會報告值 green。

4. 編輯 Shadow 文件以觀察差異事件

如要觀察差異主題的變更，請編輯 AWS IoT 主控台中的 Shadow 文件。例如，您可將 desired 值變更為顏色 red。若要這樣做，請在 AWS IoT 主控台中選擇編輯，然後在 JSON 中將desired值設定為紅色，同時將reported值設定為 green。儲存變更之前，請保持終端開啟狀態，因為您會在終端機中看到回報的差異訊息。

```
{
```

```
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

shadow.py 範例應用程式會回應此變更，並在終端機中顯示指出差異的訊息。於 MQTT 測試用戶端中，update 主題將會收到一則訊息，顯示 desired 和 reported 值的變更。

您也會看到主題 $aws/things/***thingname***/shadow/update/delta 收到訊息。如要查看訊息，請選擇此列於 Subscriptions (訂閱) 之下的主題。

```
{
"version": 13,
"timestamp": 1617318480,
"state": {
  "color": "red"
},
"metadata": {
  "color": {
    "timestamp": 1617318480
  }
}
}
```

步驟 3：疑難排解 Device Shadow 互動的錯誤

當您執行 Shadow 範例應用程式時，您可能會遇到觀察與 Device Shadow 服務互動的問題。

若程式執行成功，並提示您輸入 desired 值，您應可使用如前所述的 Shadow 文件和 MQTT 測試用戶端，觀察 Device Shadow 互動。不過，若您無法看到互動，您可檢查下列事項：

- 在 AWS IoT 主控台中檢查物件名稱及其影子

  若您看不到 Shadow 文件中的訊息，請檢閱命令，並確定其符合 AWS IoT 主控台中的物件名稱。您還可依序選擇您的物件名稱和 Shadows (影子)，來檢查您是否有經典影子。本教學課程主要著重於與經典影子的互動。

您也可以確認您使用的裝置已連線至網際網路。於 AWS IoT 主控台中，選擇您先前建立的物件，然後選擇 Interact (互動)。在物件詳細資訊頁面上，您應會在此處看到訊息：`This thing already appears to be connected.`

- 查看您訂閱的 MQTT 預留主題

  若您在 MQTT 測試用戶端中看不到訊息，請檢查您訂閱的主題是否已正確格式化。MQTT Device Shadow 主題具有格式 $aws/things/***thingname***/shadow/，且可能具有 `update`、`get` 或 `delete`，根據您希望對影子執行的動作而定。本教學課程使用主題 $aws/things/***thingname***/shadow/#，因此，在訂閱測試用戶端之主題篩選條件部分中的主題時，請確定您已正確輸入。

  當您輸入主題名稱時，請確定 *`thingname`* 與您先前建立的 AWS IoT 物件名稱相同。您還可訂閱其他 MQTT 主題，查看是否已成功執行更新。例如，您可訂閱主題 $aws/things/***thingname***/shadow/update/rejected，在更新請求失敗時收到訊息，您便可對連線問題進行偵錯。如需有關預留主題的更多資訊，請參閱 the section called “影子主題” 和 Device Shadow MQTT 主題。

步驟 4：檢閱結果及後續步驟

在本教學課程中，您會了解如何：

- 使用 `shadow.py` 範例應用程式來指定所需的狀態並更新影子的目前狀態。
- 編輯 Shadow 文件，以觀察差異事件，及 `shadow.py` 範例應用程序如何加以回應。
- 使用 MQTT 測試用戶端訂閱影子主題，並在執行範例程式時觀察更新。

後續步驟

您可訂閱其他 MQTT 預留主題，以觀察影子應用程式的更新。例如，若您僅訂閱主題 $aws/things/***thingname***/shadow/update/accepted，則更新成功執行時，您只會看到目前的狀態資訊。

您也可訂閱其他影子主題來偵錯問題，或深入了解 Device Shadow 互動，還可對 Device Shadow 互動的任何問題進行偵錯。如需詳細資訊，請參閱 the section called “影子主題” 及 Device Shadow MQTT 主題。

您也可選擇使用命名影子或使用為 LED 與 Raspberry Pi 連接其他的硬體擴展您的應用程式，並使用從終端傳送訊息來觀察其狀態的變化。

如需有關 Device Shadows 服務及在裝置、應用程式和服務中使用服務的詳細資訊，請參閱 AWS IoT Device Shadow 服務、在裝置中使用影子 and 在應用程式和服務中使用影子。

教學課程：為 建立自訂授權方 AWS IoT Core

本教學課程示範使用 AWS CLI來建立、驗證和使用自訂身分驗證的步驟。使用本教學課程，您可以選擇性地使用 Postman，藉由使用 HTTP 發佈 API 將資料傳送到 AWS IoT Core 。

本教學課程會向您展示如何建立範例 Lambda 函數，搭配使用 create-authorizer 呼叫與啟用的字符簽署，以實作授權和身分驗證邏輯，以及自訂授權方。然後，使用 驗證授權方test-invoke-authorizer，最後您可以使用 HTTP 發佈 API AWS IoT Core 將資料傳送至測試 MQTT 主題。範例請求將使用 `x-amz-customauthorizer-name` 標頭指定要叫用的授權方，並以請求標頭傳遞 token-key-name 和 `x-amz-customauthorizer-signature`。

您會在本教學課程中學到什麼：

- 如何將 Lambda 函數建立為自訂授權方處理常式
- 如何使用啟用字符簽署 AWS CLI 的 建立自訂授權方
- 如何使用 test-invoke-authorizer 命令測試您的自訂授權方
- 如何使用Postman 發佈 MQTT 主題，以及如何利用您的自訂授權方驗證請求

此教學課程約需 60 分鐘方能完成。

在本教學課程中，您將會執行下列動作：

- 步驟 1：為自訂授權方建立 Lambda 函數
- 步驟 2：為自訂授權方建立公有和私有金鑰對
- 步驟 3：建立自訂授權方資源及其授權
- 步驟 4：呼叫 test-invoke-authorizer 來測試授權方
- 步驟 5：使用 Postman 測試發佈 MQTT 郵件
- 步驟 6：在 MQTT 測試用戶端中檢視訊息
- 步驟 7：檢閱結果及後續步驟
- 步驟 8：清理

開始本教學課程之前，請確定您有：

- 設定 AWS 帳戶

  您需要 AWS 帳戶 和 AWS IoT 主控台才能完成本教學課程。

當您用於本教學課程的帳戶至少包含這些 AWS 受管政策時，此帳戶最適用：

- IAMFullAccess
- AWSIoTFullAccess
- AWSLambda_FullAccess

> **Important**
>
> 本教學課程中使用的 IAM 政策比您應該在生產實作中遵循的 IAM 政策更寬鬆。在生產環境中，請確定您的帳戶和資源策略僅授予必要的許可。
> 當您建立 IAM 政策進行生產時，請判斷使用者和角色需要哪些存取權，然後設計政策，讓使用者只能執行這些任務。
> 如需詳細資訊，請參閱 IAM 中的安全最佳實務。

- 已安裝 AWS CLI

  如需如何安裝 的資訊 AWS CLI，請參閱安裝 AWS CLI。本教學課程需要 AWS CLI 版本 `aws-cli/2.1.3 Python/3.7.4 Darwin/18.7.0 exe/x86_64` 或更新版本。

- OpenSSL 工具

  本教學課程中的範例使用 LibreSSL 2.6.5。您也可以針對本教學課程使用 OpenSSL v1.1.1i 工具。

- 檢閱了 AWS Lambda 概觀

  如果您 AWS Lambda 未曾使用過 ，請檢閱AWS Lambda和開始使用 Lambda 以了解其術語和概念。

- 已檢閱如何在 Postman 中建置請求

  如需詳細資訊，請參閱建置請求。

- 已從前一個教學課程中移除自訂授權方

  您的 一次 AWS 帳戶 只能設定有限數量的自訂授權方。如需如何移除自訂授權方的相關資訊，請參閱 the section called “步驟 8：清理”。

步驟 1：為自訂授權方建立 Lambda 函數

中的自訂身分驗證 AWS IoT Core 會使用您建立的授權方資源來驗證和授權用戶端。您將在本節中建立的 函數會在用戶端連線至 和存取 AWS IoT 資源時，進行身分驗證 AWS IoT Core 和授權。

Lambda 函數會執行下列動作：

- 如果請求來自 test-invoke-authorizer，它會傳回一個具有 Deny 動作的 IAM 政策。
- 如果請求來自使用 HTTP 的 Postman，且 `actionToken` 參數的值為 `allow`，則會傳回具有 `Allow`動作的 IAM 政策。否則，它會傳回一個具有 Deny 動作的 IAM 政策。

若要為自訂授權方建立 Lambda 函數

1. 在 Lambda 主控台中，開啟 Functions (函數)。
2. 選擇建立函數。
3. 確認已選取 Author from scratch (從頭開始撰寫)。
4. 在 Basic information (基本資訊) 下：
   a. 在函數名稱 中，輸入 **`custom-auth-function`**。
   b. 在執行期中，確認 Node.js 18.x
5. 選擇建立函數。

   Lambda 會建立 Node.js 函數和許可函數上傳記錄的執行角色。Lambda 函數會在您叫用函數時擔任執行角色，並使用執行角色來建立 AWS SDK 的登入資料，以及從事件來源讀取資料。
6. 若要在 AWS Cloud9 編輯器中查看函數的程式碼和組態，請在設計工具視窗中選擇 custom-auth-function，然後在編輯器的導覽窗格中選擇 index.js。

   對於指令碼語言 (例如 Node.js)，Lambda 包含可傳回成功回應的基本函數。您可以使用 AWS Cloud9 編輯器來編輯函數，只要原始碼不超過 3 MB。
7. 將編輯器中的 index.js 程式碼取代為下列程式碼：

```
// A simple Lambda function for an authorizer. It demonstrates
// How to parse a CLI and Http password to generate a response.

export const handler = async (event, context, callback) => {

    //Http parameter to initiate allow/deny request
    const HTTP_PARAM_NAME='actionToken';
    const ALLOW_ACTION = 'Allow';
    const DENY_ACTION = 'Deny';

    //Event data passed to Lambda function
    var event_str = JSON.stringify(event);
```

```
    console.log('Complete event :'+ event_str);

    //Read protocolData from the event json passed to Lambda function
    var protocolData = event.protocolData;
    console.log('protocolData value---> ' + protocolData);

    //Get the dynamic account ID from function's ARN to be used
    // as full resource for IAM policy
    var ACCOUNT_ID = context.invokedFunctionArn.split(":")[4];
    console.log("ACCOUNT_ID---"+ACCOUNT_ID);

    //Get the dynamic region from function's ARN to be used
    // as full resource for IAM policy
    var REGION = context.invokedFunctionArn.split(":")[3];
    console.log("REGION---"+REGION);

    //protocolData data will be undefined if testing is done via CLI.
    // This will help to test the set up.
    if (protocolData === undefined) {

        //If CLI testing, pass deny action as this is for testing purpose only.
        console.log('Using the test-invoke-authorizer cli for testing only');
        callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

    } else{

        //Http Testing from Postman
        //Get the query string from the request
        var queryString = event.protocolData.http.queryString;
        console.log('queryString values -- ' + queryString);
        /*         global URLSearchParams       */
        const params = new URLSearchParams(queryString);
        var action = params.get(HTTP_PARAM_NAME);

        if(action!=null && action.toLowerCase() === 'allow'){

            callback(null, generateAuthResponse(ALLOW_ACTION,ACCOUNT_ID,REGION));

        }else{

            callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

        }
```

```
    }

};

// Helper function to generate the authorization IAM response.
var generateAuthResponse = function(effect,ACCOUNT_ID,REGION) {

    var full_resource = "arn:aws:iot:"+ REGION + ":" + ACCOUNT_ID + ":*";
    console.log("full_resource---"+full_resource);

    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'principalId';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var statement = {};
    statement.Action = 'iot:*';
    statement.Effect = effect;
    statement.Resource = full_resource;
    policyDocument.Statement[0] = statement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 600;

    console.log('custom auth policy function called from http');
    console.log('authResponse --> ' + JSON.stringify(authResponse));
    console.log(authResponse.policyDocuments[0]);

    return authResponse;
}
```

8. 選擇部署。
9. 在 Changes deployed (已部署變更) 出現在編輯器上方之後：
   a. 捲動至編輯器上方的 Function overview (函數概觀) 區段。
   b. 複製 Function ARN (函數 ARN) 並加以儲存，以便稍後可在本教學課程中使用。
10. 測試您的函數
    a. 選擇 Test (測試) 標籤。
    b. 使用預設測試設定，選擇 Invoke (叫用)。

c. 如果測試成功，請在 Execution results (執行結果) 下，開啟 Details (詳細資訊) 檢視。您應該會看到函數傳回的政策文件。

如果測試失敗或看不到政策文件，請檢閱程式碼以尋找並更正錯誤。

步驟 2：為自訂授權方建立公有和私有金鑰對

您的自訂授權方需要公有和私有金鑰來驗證它。本節中的命令會使用 OpenSSL 工具來建立此金鑰對。

若要為自訂授權方建立公有和私有金鑰對

1. 建立私有金鑰檔案。

```
openssl genrsa -out private-key.pem 4096
```

2. 驗證您剛建立的私有金鑰檔案。

```
openssl rsa -check -in private-key.pem -noout
```

如果命令未顯示任何錯誤，則私有金鑰檔案是有效的。

3. 建立公有金鑰檔案。

```
openssl rsa -in private-key.pem -pubout -out public-key.pem
```

4. 驗證公有金鑰檔案。

```
openssl pkey -inform PEM -pubin -in public-key.pem -noout
```

如果命令未顯示任何錯誤，則公有金鑰檔案是有效的。

步驟 3：建立自訂授權方資源及其授權

AWS IoT 自訂授權方是將先前步驟中建立的所有元素連結在一起的資源。在本節中，您將建立自訂授權方資源，並許可其執行您先前建立的 Lambda 函數。您可以使用 AWS IoT 主控台 AWS CLI、 或 AWS API 來建立自訂授權方資源。

在本教學課程中，您只需建立一個自訂授權方即可。本節說明如何使用 AWS IoT 主控台和 建立 AWS CLI，讓您可以使用最方便的方法。由任一種方法建立的自訂授權方資源之間沒有任何差異。

建立自訂授權方資源

選擇其中一個選項，來建立自訂授權方資源

- 使用 AWS IoT 主控台建立自訂授權方
- 使用 AWS CLI建立自訂授權方

若要建立自訂授權方 (主控台)

1. 開啟AWS IoT 主控台的自訂授權方頁面，然後選擇建立授權方。
2. 在建立授權方中：
   a. 在授權方名稱中，輸入 **my-new-authorizer**。
   b. 在授權方狀態中，勾選作用中。
   c. 在 Authorizer function (授權方函數) 中，選擇您先前建立的 Lambda 函數。
   d. 在 Token validation - optional (字符驗證 - 選用) 中：
      i. 切換字符驗證。
      ii. 在符記金鑰名稱中，輸入 **tokenKeyName**。
      iii. 選擇 Add key (新增金鑰)。
      iv. 在金鑰名稱中，輸入 **FirstKey**。
      v. 在公有金鑰中，輸入 public-key.pem 檔案的內容。務必包括檔案中具有 -----BEGIN PUBLIC KEY----- 和 -----END PUBLIC KEY----- 的字行，並且不得從檔案內容中新增或刪除任何換行字元、歸位字元或其他字元。您輸入的字串應該看起來與這個範例相似。

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
```

```
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----
```

3. 選擇 Create Authorizer (建立授權方)。
4. 如果已建立自訂授權方資源，您會看到自訂授權方的清單，而且您的新自訂授權方應該會出現在清單中，接著您可以繼續進行下一節來測試它。

   如果您看到錯誤，請檢閱錯誤並嘗試重新建立您的自訂授權方，然後再次檢查這些項目。請注意，每個自訂授權方資源都必須具有唯一名稱。

建立自訂授權方 (AWS CLI)

1. 將您的值替代為 `authorizer-function-arn` 和 `token-signing-public-keys`，然後執行下列命令：

```
aws iot create-authorizer \
--authorizer-name "my-new-authorizer" \
--token-key-name "tokenKeyName" \
--status ACTIVE \
--no-signing-disabled \
--authorizer-function-arn "arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function" \
--token-signing-public-keys FirstKey="-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----"
```

   其中：

   - `authorizer-function-arn` 值是您為自訂授權方建立之 Lambda 函數的 Amazon 資源名稱 (ARN)。

- token-signing-public-keys 值包含金鑰的名稱 **FirstKey** 以及 public-key.pem 檔案的內容 務必包括檔案中具有 -----BEGIN PUBLIC KEY----- 和 -----END PUBLIC KEY----- 的字行，並且不得從檔案內容中新增或刪除任何換行字元、歸位字元或其他字元。

  注意：輸入公有金鑰時要小心，因為公有金鑰值若有任何更改，都會使其無法使用。

2. 如果已建立自訂授權方，則命令會傳回新資源的名稱和 ARN，如下所示。

```
{
    "authorizerName": "my-new-authorizer",
    "authorizerArn": "arn:aws:iot:Region:57EXAMPLE833:authorizer/my-new-authorizer"
}
```

儲存 authorizerArn 值，以便用於下一個步驟。

記住，每個自訂授權方資源都必須具有唯一名稱。

授權自訂授權方資源

在本節中，您將許可您剛建立的自訂授權方資源執行 Lambda 函數。若要授與許可，您可以使用 add-permission CLI 命令。

使用 將許可授予 Lambda 函數 AWS CLI

1. 在插入您的值之後，輸入以下命令。請注意，statement-id 值必須是唯一的。如果您之前已執行本教學課程，或者如果您得到 ResourceConflictException 錯誤，請將 *Id-1234* 取代為另一個值，。

```
aws lambda add-permission  \
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. 如果命令成功，它會傳回許可陳述式，例如此範例。您可以繼續下一節來測試自訂授權方。

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
```

```
auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
}
```

如果命令未成功，它會傳回錯誤，例如此範例。您必須先檢閱並更正錯誤，然後才能繼續進行。

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

步驟 4：呼叫 test-invoke-authorizer 來測試授權方

定義所有資源後，在本節中，您會從命令列呼叫 test-invoke-authorizer 來測試授權是否通過。

請注意，從命令列叫用授權方時，未定義 `protocolData`，因此授權方將一律傳回 DENY 文件。不過，此測試會確認您的自訂授權方和 Lambda 函數已正確設定，即使它未完全測試 Lambda 函數也一樣。

使用 測試您的自訂授權方及其 Lambda 函數 AWS CLI

1. 在具有您在前一個步驟中建立之 `private-key.pem` 檔案的目錄中，執行下列命令。

```
echo -n "tokenKeyValue" | openssl dgst -sha256 -sign private-key.pem | openssl
 base64 -A
```

此命令會建立要在下一個步驟中使用的簽章字串。簽章字串看起來像這樣：

```
dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V
+MMWu09YSA86+64Y3Gt4tOykpZqn9mn
VB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeeh
bQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjj
szEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY
+d1vTvdthKtYHBq8MjhzJ0kggbt29V
QJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuX
f3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+K
```

```
EWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFH
xRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

複製此簽章字串以在下一個步驟中使用。請小心不要包含任何額外的字元或留下任何額外的字元。

2. 在此命令中，會將 token-signature 值取代為來自前一個步驟的簽章字串，並執行此命令來測試您的授權方。

```
aws iot test-invoke-authorizer \
--authorizer-name my-new-authorizer \
--token tokenKeyValue \
--token-signature dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr
+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp
+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsH
+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/
P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuXf3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o
+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

如果命令成功，它會傳回自訂授權方函數產生的資訊，例如此範例。

```
{
    "isAuthenticated": true,
    "principalId": "principalId",
    "policyDocuments": [
        "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Action\":\"iot:*\",
\"Effect\":\"Deny\",\"Resource\":\"arn:aws:iot:Region:57EXAMPLE833:*\"}]}"
    ],
    "refreshAfterInSeconds": 600,
    "disconnectAfterInSeconds": 3600
}
```

如果指令傳回錯誤，請檢閱錯誤並再次檢查您在本節中使用的命令。

步驟 5：使用 Postman 測試發佈 MQTT 郵件

1. 若要從命令列取得您的裝置資料端點，請呼叫 describe-endpoint，如這裡所示

```
aws iot describe-endpoint --output text --endpoint-type iot:Data-ATS
```

儲存此地址，以在稍後步驟中用作 *device_data_endpoint_address*。

2. 開啟新的 Postman 視窗並建立新的 HTTP POST 請求。

   a. 從您的電腦中，開啟 Postman 應用程式。

   b. 在 Postman 的 File (檔案) 選單中，選擇 New (新增)。

   c. 在 New (新增) 對話方塊中，選擇 Request (請求)。

   d. 在儲存請求中，

      i. 在 Request name (請求名稱) 中，輸入 **Custom authorizer test request**。

      ii. 在 Select a collection or folder to save to: (選取要儲存到哪個集合或資料夾：) 中，選擇或建立要儲存此請求的集合。

      iii. 選擇 Save to ***collection_name*** (儲存至 collection_name)。

3. 建立 POST 請求來測試您的自訂授權方。

   a. 在 URL 欄位旁邊的請求方法選取器中，選擇 POST。

   b. 在 URL 欄位中，為您的請求建立 URL，方法為搭配使用下列 URL 與來自前一個步驟中 describe-endpoint 命令的 *device_data_endpoint_address*。

      ```
      https://device_data_endpoint_address:443/topics/test/cust-auth/topic?
      qos=0&actionToken=allow
      ```

      請注意，此 URL 包含 `actionToken=allow` 查詢參數，它會告訴您的 Lambda 函數傳回允許存取 AWS IoT的政策文件。輸入 URL 之後，查詢參數也會出現在 Postman 的 Params (參數) 標籤中。

   c. 在 Auth (身分驗證) 標籤的 Type (類型) 欄位中，選擇 No Auth (無需身分驗證)。

   d. 在標頭標籤中：

      i. 如果有已核取的 Host (主機) 金鑰，請取消核取此金鑰。

      ii. 在標頭清單的底部，加入這些新標題並確認已核取它們。將 **Host** 值取代為您的 *device_data_endpoint_address*，並將 **x-amz-customauthorizer-signature** 值取代為在上節中與 test-invoke-authorize 命令搭配使用的簽章字串。

金鑰	值
x-amz-customauthorizer-name	**my-new-authorizer**
Host	*device_data_endpoint_address*
tokenKeyName	**tokenKeyValue**
x-amz-customauthorizer-signature	*dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjjszEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca+tsDuXf3LzCwQQF/YSUy02u5XkWn+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=*

e. 在主體標籤中：

i. 在資料格式選項方塊中，選擇 Raw (原始)。

ii. 在資料類型清單中，選擇 JavaScript。

iii. 在文字欄位中，為您的測試訊息輸入此 JSON 訊息承載：

```
{
    "data_mode": "test",
    "vibration": 200,
    "temperature": 40
}
```

4. 選擇 Send (傳送) 以傳送請求。

如果請求成功，它會傳回：

```
{
    "message": "OK",
    "traceId": "ff35c33f-409a-ea90-b06f-fbEXAMPLE25c"
}
```

成功回應表示您的自訂授權方允許與 的連線， AWS IoT 且測試訊息已交付至 中的代理程式 AWS IoT Core。

如果傳回錯誤，請檢閱錯誤訊息、*device_data_endpoint_address*、簽章字串，以及其他標頭值。

請將此請求保留在 Postman，以供下一節使用。

步驟 6：在 MQTT 測試用戶端中檢視訊息

在上一個步驟中，您使用 Postman AWS IoT 將模擬裝置訊息傳送到 。成功回應指出您的自訂授權方允許連線到 AWS IoT ，並指出測試訊息已傳遞至 AWS IoT Core中的代理程式。在本節中，您將使用 AWS IoT 主控台中的 MQTT 測試用戶端，以其他裝置和服務可能的方式查看該訊息的訊息內容。

若要查看自訂授權方所授權的測試訊息

1. 在 AWS IoT 主控台中，開啟 MQTT 測試用戶端。

2. 在 Subscribe to topic (訂閱主題) 標籤的 Topic filter (主題篩選條件) 中，輸入 **`test/cust-auth/topic`**，這是前一節的 Postman 範例中使用的訊息主題。
3. 選擇 Subscribe (訂閱)。

   保留此視窗讓後續步驟可以看到它。
4. 在 Postman 中，於您為前一節建立的請求中，選擇 Send (傳送)。

   檢閱回應以確定回應成功。如果未成功，請依照前一節所述疑難排解錯誤。
5. 在 MQTT test client (MQTT 測試用戶端) 中，您應該會看到顯示訊息主題的新項目，而且如果展開的話，則會看到您從 Postman 傳送之請求中的訊息承載。

   如果未在 MQTT test client (MQTT 測試用戶端) 中看到您的訊息，以下是一些需要檢查的事項：

   - 確定您的 Postman 請求成功傳回。如果 AWS IoT 拒絕連線並傳回錯誤，則請求中的訊息不會傳遞給訊息中介裝置。
   - 確定 AWS 區域 用來開啟 AWS IoT 主控台的 AWS 帳戶 和 與您在 Postman URL 中使用的 相同。
   - 請確定您使用自訂授權方的適當端點。預設 IoT 端點可能不支援搭配 Lambda 函數使用自訂授權方。反之，您可以使用網域組態來定義新的端點，然後為自訂授權方指定該端點。
   - 確定您已在 MQTT test client (MQTT 測試用戶端) 中正確地輸入主題。主題篩選條件會區分大小寫。如有疑問，您也可以訂閱 主題，該#主題會訂閱傳遞訊息代理程式 AWS 帳戶 並 AWS 區域 用來開啟 AWS IoT 主控台的所有 MQTT 訊息。

步驟 7：檢閱結果及後續步驟

在本教學課程中：

- 已將 Lambda 函數建立為自訂授權方處理常式
- 已在啟用字符簽署的情況下建立自訂授權方
- 已使用 test-invoke-authorizer 命令測試您的自訂授權方
- 已使用 Postman 發佈 MQTT 主題，並利用您的自訂授權方驗證請求
- 已使用 MQTT test client (MQTT 測試用戶端) 來檢視從 Postman 測試傳送的訊息

後續步驟

從 Postman 傳送一些訊息以驗證自訂授權方是否正在運作之後，請試驗看看變更本教學課程的不同層面如何影響結果。以下是幾個入門範例。

- 變更簽章字串，以便查看未經授權之連線嘗試的處理方式不再有效。您應該會得到錯誤回應 (例如這個錯誤回應)，且訊息應該不會出現在 MQTT test client (MQTT 測試用戶端) 中。

```
{
    "message": "Forbidden",
    "traceId": "15969756-a4a4-917c-b47a-5433e25b1356"
}
```

- 若要進一步了解如何尋找在您開發和使用 AWS IoT 規則時可能發生的錯誤，請參閱 監控 AWS IoT。

步驟 8：清理

如果想要重複本教學課程，您可能需要移除某些自訂授權方。您的 AWS 帳戶 一次只能設定有限數量的自訂授權方，而且當您嘗試新增新的授權方而不移除現有的自訂授權方`LimitExceededException`時，您可以取得 。

移除自訂授權方 (主控台)

1. 開啟AWS IoT 主控台的自訂授權方頁面，然後在自訂授權方清單中，尋找要移除的自訂授權方。
2. 開啟自訂授權方詳細資訊頁面，然後從 Actions (動作) 選單中，選擇 Edit (編輯)。
3. 取消核取 Activate authorizer (啟用授權方)，然後選擇 Update (更新)。

   在自訂授權方作用中時，您無法將其刪除。
4. 從自訂授權方詳細資訊頁面中，開啟 Actions (動作) 選單，然後選擇 Delete (刪除)。

移除自訂授權方 (AWS CLI)

1. 列出您已安裝的自訂授權方，並找出要刪除的自訂授權方名稱。

   ```
   aws iot list-authorizers
   ```
2. 在將 *Custom_Auth_Name* 取代為要刪除之自訂授權方的 `authorizerName` 之後，執行此命令將自訂授權方設定為 `inactive`。

```
aws iot update-authorizer --status INACTIVE --authorizer-name Custom_Auth_Name
```

3. 在將 *Custom_Auth_Name* 取代為要刪除之自訂授權方的 authorizerName 之後，執行此命令來刪除自訂授權方。

```
aws iot delete-authorizer --authorizer-name Custom_Auth_Name
```

教學課程：使用 AWS IoT 和 Raspberry Pi 監控土壤濕度

本教學課程說明如何使用 Raspberry Pi、濕度感應器，以及 AWS IoT 監控房屋植物或花園的土壤濕度等級。Raspberry Pi 會執行程式碼，從感應器讀取濕度等級和溫度，然後將資料傳送至 AWS IoT。您可以在 中建立規則 AWS IoT ，當濕度等級低於閾值時，將電子郵件傳送至訂閱 Amazon SNS 主題的地址。

> **Note**
>
> 本教學課程可能不是最新版本。自本主題最初發佈後，某些參考文件可能已被取代。

內容

- 先決條件
- 設定 AWS IoT
  - 步驟 1：建立 AWS IoT 政策
  - 步驟 2：建立 AWS IoT 物件、憑證和私有金鑰
  - 步驟 3：建立 Amazon SNS 主題與訂閱。
  - 步驟 4：建立 AWS IoT 規則來傳送電子郵件
- 設定您的 Raspberry Pi 和濕度感應器

先決條件

為完成此教學課程您需要：

- AWS 帳戶。
- 擁有管理員許可的 IAM 使用者。

- 執行 Windows、macOS、Linux 或 Unix 的開發電腦，用來存取 AWS IoT 主控台。
- 執行最新 Raspberry Pi 作業系統的 Raspberry Pi 3B 或 4B。 https://www.raspberrypi.com/software/operating-systems/如需安裝說明，請參閱 Raspberry Pi 網站上的安裝作業系統。
- 用於 Raspberry Pi 的顯示器、鍵盤、滑鼠和 Wi-Fi 網路或乙太網路連線。
- 與 Raspberry Pi 相容的濕度感應器。本教學課程中使用的感應器是 Adafure STEMMA I2C 電容式濕度感應器搭配 JST 4 針腳轉母頭插槽纜線接頭。

設定 AWS IoT

若要完成此教學課程，您需要建立以下資源。若要將裝置連線至 AWS IoT，您可以建立 IoT 物件、裝置憑證和 AWS IoT 政策。

- 實 AWS IoT 物。

  代表實體裝置 (在此案例中為 Rasberry Pi)，包含有關裝置的靜態中繼資料的物件。

- 裝置憑證。

  所有裝置都必須擁有裝置憑證，才能連接至並向 AWS IoT驗證。

- AWS IoT 政策。

  每個裝置憑證都有一或多個與其相關聯的 AWS IoT 政策。這些政策會決定裝置可存取 AWS IoT 的資源。

- AWS IoT 根 CA 憑證。

  裝置和其他用戶端使用 AWS IoT 根 CA 憑證來驗證與其通訊的 AWS IoT 伺服器。如需詳細資訊，請參閱伺服器驗證。

- AWS IoT 規則。

  規則包含查詢和一或多個規則動作。查詢會從裝置訊息擷取資料，以判斷是否應處理訊息資料。規則動作會指定當資料符合查詢時要採取的動作。

- Amazon SNS 主題和主題訂閱。

  此規則會監聽來自 Raspberry Pi 的濕度資料。如果值低於閾值，它會將訊息傳送到 Amazon SNS 主題。Amazon SNS 會將該訊息傳送到訂閱該主題的所有電子郵件地址。

步驟 1：建立 AWS IoT 政策

建立允許 Raspberry Pi 連線和傳送訊息 AWS IoT 的政策 AWS IoT。

1. 在 AWS IoT 主控台中，如果 Get started (開始) 按鈕出現，請選擇它。否則，請在導覽窗格中展開 Security (安全性)，然後選擇 Policies (政策)。
2. 如果 You don't have any policies yet (您尚未有任何政策) 對話方塊出現，請選擇 Create a policy (建立政策)。否則，請選擇 Create (建立)。
3. 輸入 AWS IoT 政策的名稱 （ 例如 **MoistureSensorPolicy**)。
4. 在 Add statements (新增陳述式) 區段中，將現有政策取代為下列 JSON。使用 和 AWS 帳戶 號碼取代*##* AWS 區域 和*##*。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/RaspberryPi"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/accepted",
```

```
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/RaspberryPi"
        }
    ]
}
```

5. 選擇建立。

步驟 2：建立 AWS IoT 物件、憑證和私有金鑰

在 AWS IoT 登錄檔中建立物件以代表您的 Raspberry Pi。

1. 在 AWS IoT 主控台的導覽窗格中，依序選擇 Manage (管理) 和 Things (物件)。

2. 如果顯示 You don't have any things yet (尚無任何物件) 對話方塊，請選擇 Register a thing (註冊物件)。否則，請選擇 Create (建立)。
3. 在建立 AWS IoT 物件頁面上，選擇建立單一物件。
4. 在 Add your device to the device registry (將裝置新增至裝置登錄檔) 頁面上，輸入您 IoT 物件的名稱 (例如 **RaspberryPi**)，然後選擇 Next (下一步)。您無法在建立之後變更物件的名稱。要變更物件的名稱，您必須建立一個新的物件並為它命名，然後刪除舊的物件。
5. 在 Add a certificate for your thing (新增物件的憑證) 頁面上，選擇 Create certificate (建立憑證)。
6. 選擇 Download (下載) 連結來下載憑證、私有金鑰和根憑證授權機構憑證。

   > **Important**
   >
   > 這是您可以下載憑證和私有金鑰唯一機會。

7. 若要啟用憑證，請選擇 Activate (啟用)。憑證必須作用中，裝置才能連接到 AWS IoT。
8. 選擇 Attach a policy (連接政策)。
9. 針對 Add a policy for your thing (新增您的物件的政策)，選擇 MoistureSensorPolicy，然後選擇 Register Thing (註冊物件)。

步驟 3：建立 Amazon SNS 主題與訂閱。

建立 Amazon SNS 主題與訂閱。

1. 在 AWS SNS 主控台的導覽窗格中，選擇 Topics (主題)，然後選擇 Create topic (建立主題)。
2. 選擇類型作為標準，然後輸入主題的名稱 （ 例如，**MoistureSensorTopic**)。
3. 輸入主題的顯示名稱 (例如，**Moisture Sensor Topic**)。這是在 Amazon SNS 主控台中針對您的主題顯示的名稱。
4. 請選擇建立主題。
5. 在 Amazon SNS 主題詳細資訊頁面中，選擇 Create subscription (建立訂閱)。
6. 對於通訊協定，選擇電子郵件。
7. 針對 Endpoint (端點)，輸入電子郵件地址。
8. 選擇建立訂閱。
9. 開啟您的電子郵件用戶端，並尋找主旨為 **MoistureSensorTopic** 的訊息。開啟電子郵件，然後按一下 Confirm subscription (確認訂閱) 連結。

> **Important**
>
> 在您確認訂閱之前，不會收到來自此 Amazon SNS 主題的任何電子郵件提醒。

您應該會收到含有您輸入文字的電子郵件訊息。

步驟 4：建立 AWS IoT 規則來傳送電子郵件

AWS IoT 規則會定義查詢，以及從裝置接收訊息時要採取的一或多個動作。 AWS IoT 規則引擎會接聽裝置傳送的訊息，並使用訊息中的資料來判斷是否應採取一些動作。如需詳細資訊，請參閱的規則 AWS IoT。

在此教學中，您的 Raspberry Pi 會將訊息發佈至 `aws/things/RaspberryPi/shadow/update`。這是內部的 MQTT 主題，供裝置和物件影子服務使用。Raspberry Pi 發佈的訊息會具備以下格式：

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
    }
}
```

您會建立一個查詢，從傳入的訊息擷取濕度和溫度資料。您也可以建立 Amazon SNS 動作，在濕度讀數低於閾值時，取得資料並將其傳送給 Amazon SNS 主題訂閱者。

建立 Amazon SNS 規則

1. 在 AWS IoT 主控台中，選擇訊息路由，然後選擇規則。如果 You don't have any rules yet (您尚未有任何規則) 對話方塊出現，請選擇 Create a rule (建立規則)。否則，請選擇建立規則。
2. 在規則屬性頁面中，輸入規則名稱，例如 **`MoistureSensorRule`**，並提供簡短的規則描述，例如 **`Sends an alert when soil moisture level readings are too low`**。
3. 選擇下一步並設定 SQL 陳述式。選擇 SQL 版本為 2016-03-23，然後輸入下列 AWS IoT SQL 查詢陳述式：

```
SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
 state.reported.moisture < 400
```

此陳述式會在 `moisture` 讀數小於 `400` 時觸發規則動作。

> **Note**
>
> 您可能需要使用不同的值。在 Raspberry Pi 上執行程式碼之後，您可以碰觸感應器、將其放入水中或將其放入花盆中，以查看從感應器取得的值。

4. 選擇下一步並連接規則動作。針對動作 1，選擇簡易通知服務。此規則動作的描述是傳送訊息做為 SNS 推送通知。
5. 針對 SNS 主題，選擇您在 步驟 3：建立 Amazon SNS 主題與訂閱。、MoistureSensorTopic 中建立的主題，並將訊息格式保留為 RAW。對於 IAM 角色，選擇建立新角色。輸入角色的名稱，例如 **`LowMoistureTopicRole`**，然後選擇建立角色。
6. 選擇下一步以檢閱，然後選擇建立以建立規則。

設定您的 Raspberry Pi 和濕度感應器

將 microSD 卡插入 Raspberry Pi，連接顯示器、鍵盤、滑鼠，以及乙太網路纜線 (如果未使用 Wi-Fi)。還不要連接電源線。

將 JST 跳線纜線連接至濕度感應器。跳線的另一端有四條配線：

- 綠色：I2C SCL
- 白色：I2C SDA
- 紅色：功率 (3.5 V)
- 黑色：接地

按住 Raspberry Pi 與右側的乙太網路插孔。在此方向中，上方有兩列 GPIO 接腳。按照以下順序，將濕度感應器的配線連接到接腳的下方排。從最左側接腳開始，連接紅色 (電源)、白色 (SDA) 和綠色 (SCL)。略過一個接腳，然後連接黑色 (接地) 配線。如需詳細資訊，請參閱 Python Computer Wiring。

將電源線連接至 Raspberry Pi，並將另一端插入牆上插座，以將其開啟。

設定您的 Raspberry Pi

1. 在 Welcome to Raspberry Pi (歡迎使用 Raspberry Pi) 上，選擇 Next (下一步)。

2. 選擇您的國家/地區、語言、時區及鍵盤配置。選擇 Next (下一步)。
3. 輸入您的 Raspberry Pi 的密碼，然後選擇 Next (下一步)。
4. 選擇您的 Wi-Fi 網路，然後選擇 Next (下一步)。如果您不是使用 Wi-Fi 網路，請選擇 Skip (略過)。
5. 選擇 Next (下一步) 以檢查軟體更新。更新完成時，選擇 Restart (重新啟動) 以重新啟動您的 Raspberry Pi。

在 Raspberry Pi 啟動後，啟用 I2C 介面。

1. 在 Raspbian 桌面的左上角，按一下 Raspberry 圖示，選擇 Preferences (偏好設定)，然後選擇 Raspberry Pi Configuration (Raspberry Pi 組態)。
2. 在 Interfaces (介面) 標籤上，針對 I2C，選擇 Enable (啟用)。
3. 選擇確定。

Adafruit STEMMA 濕度感應器的程式庫是針對 CircuitPython 所撰寫。若要在 Raspberry Pi 上執行它們，您需要安裝最新版本的 Python 3。

1. 從命令提示執行下列命令，以更新您的 Raspberry Pi 軟體：

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```
2. 執行以下命令來更新您的 Python 3 安裝：

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. 執行以下命令來安裝 Raspberry Pi GPIO 程式庫：

   ```
   pip3 install RPI.GPIO
   ```
4. 執行以下命令來安裝 Adafruit Blinka 程式庫：

   ```
   pip3 install adafruit-blinka
   ```

   如需詳細資訊，請參閱 在 Raspberry Pi 上安裝 CircuitPython 程式庫。
5. 執行以下命令來安裝 Adafruit Seesaw 程式庫：

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```
6. 執行下列命令來安裝適用於 Python 的 AWS IoT 裝置 SDK：

```
pip3 install AWSIoTPythonSDK
```

您的 Raspberry Pi 現在擁有所有必要的程式庫。建立名為 **moistureSensor.py** 的檔案，並將下列 Python 程式碼複製到檔案：

```
from adafruit_seesaw.seesaw import Seesaw
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from board import SCL, SDA

import logging
import time
import json
import argparse
import busio

# Shadow JSON schema:
#
# {
#   "state": {
#       "desired":{
#           "moisture":<INT VALUE>,
#           "temp":<INT VALUE>
#       }
#   }
# }

# Function called when a shadow is updated
def customShadowCallback_Update(payload, responseStatus, token):

    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")
```

```
    if responseStatus == "rejected":
        print("Update request " + token + " rejected!")

# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):

     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")

    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")


# Read in command-line parameters
def parseArgs():

    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
 help="Your device data endpoint")
    parser.add_argument("-r", "--rootCA", action="store", required=True,
 dest="rootCAPath", help="Root CA file path")
    parser.add_argument("-c", "--cert", action="store", dest="certificatePath",
 help="Certificate file path")
    parser.add_argument("-k", "--key", action="store", dest="privateKeyPath",
 help="Private key file path")
    parser.add_argument("-p", "--port", action="store", dest="port", type=int,
 help="Port number override")
    parser.add_argument("-n", "--thingName", action="store", dest="thingName",
 default="Bot", help="Targeted thing name")
    parser.add_argument("-id", "--clientId", action="store", dest="clientId",
 default="basicShadowUpdater", help="Targeted client id")

    args = parser.parse_args()
    return args


# Configure logging
# AWSIoTMQTTShadowClient writes data to the log
```

```
def configureLogging():

    logger = logging.getLogger("AWSIoTPythonSDK.core")
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
 %(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)


# Parse command line arguments
args = parseArgs()

if not args.certificatePath or not args.privateKeyPath:
    parser.error("Missing credentials for authentication.")
    exit(2)

# If no --port argument is passed, default to 8883
if not args.port:
    args.port = 8883


# Init AWSIoTMQTTShadowClient
myAWSIoTMQTTShadowClient = None
myAWSIoTMQTTShadowClient = AWSIoTMQTTShadowClient(args.clientId)
myAWSIoTMQTTShadowClient.configureEndpoint(args.host, args.port)
myAWSIoTMQTTShadowClient.configureCredentials(args.rootCAPath, args.privateKeyPath,
 args.certificatePath)

# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTShadowClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTShadowClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTShadowClient.configureMQTTOperationTimeout(5) # 5 sec

# Initialize Raspberry Pi's I2C interface
i2c_bus = busio.I2C(SCL, SDA)

# Intialize SeeSaw, Adafruit's Circuit Python library
ss = Seesaw(i2c_bus, addr=0x36)

# Connect to AWS IoT
myAWSIoTMQTTShadowClient.connect()
```

```
# Create a device shadow handler, use this to update and delete shadow document
deviceShadowHandler =
 myAWSIoTMQTTShadowClient.createShadowHandlerWithName(args.thingName, True)

# Delete current shadow JSON doc
deviceShadowHandler.shadowDelete(customShadowCallback_Delete, 5)

# Read data from moisture sensor and update shadow
while True:

    # read moisture level through capacitive touch pad
    moistureLevel = ss.moisture_read()

    # read temperature from the temperature sensor
    temp = ss.get_temp()

    # Display moisture and temp readings
    print("Moisture Level: {}".format(moistureLevel))
    print("Temperature: {}".format(temp))

    # Create message payload
    payload = {"state":{"reported":{"moisture":str(moistureLevel),"temp":str(temp)}}}

    # Update shadow
    deviceShadowHandler.shadowUpdate(json.dumps(payload), customShadowCallback_Update,
 5)
    time.sleep(1)
```

將檔案儲存到您可以找到的位置。從命令列搭配下列參數執行 `moistureSensor.py`：

端點

您的自訂 AWS IoT 端點。如需詳細資訊，請參閱Device Shadow REST API。

rootCA

AWS IoT 根 CA 憑證的完整路徑。

cert

AWS IoT 您的裝置憑證的完整路徑。

金鑰

裝置 AWS IoT 憑證私有金鑰的完整路徑。

thingName

您的物件名稱 (在此案例中為 RaspberryPi)。

clientId

MQTT 用戶端 ID。請使用 RaspberryPi。

命令列看起來應該如下：

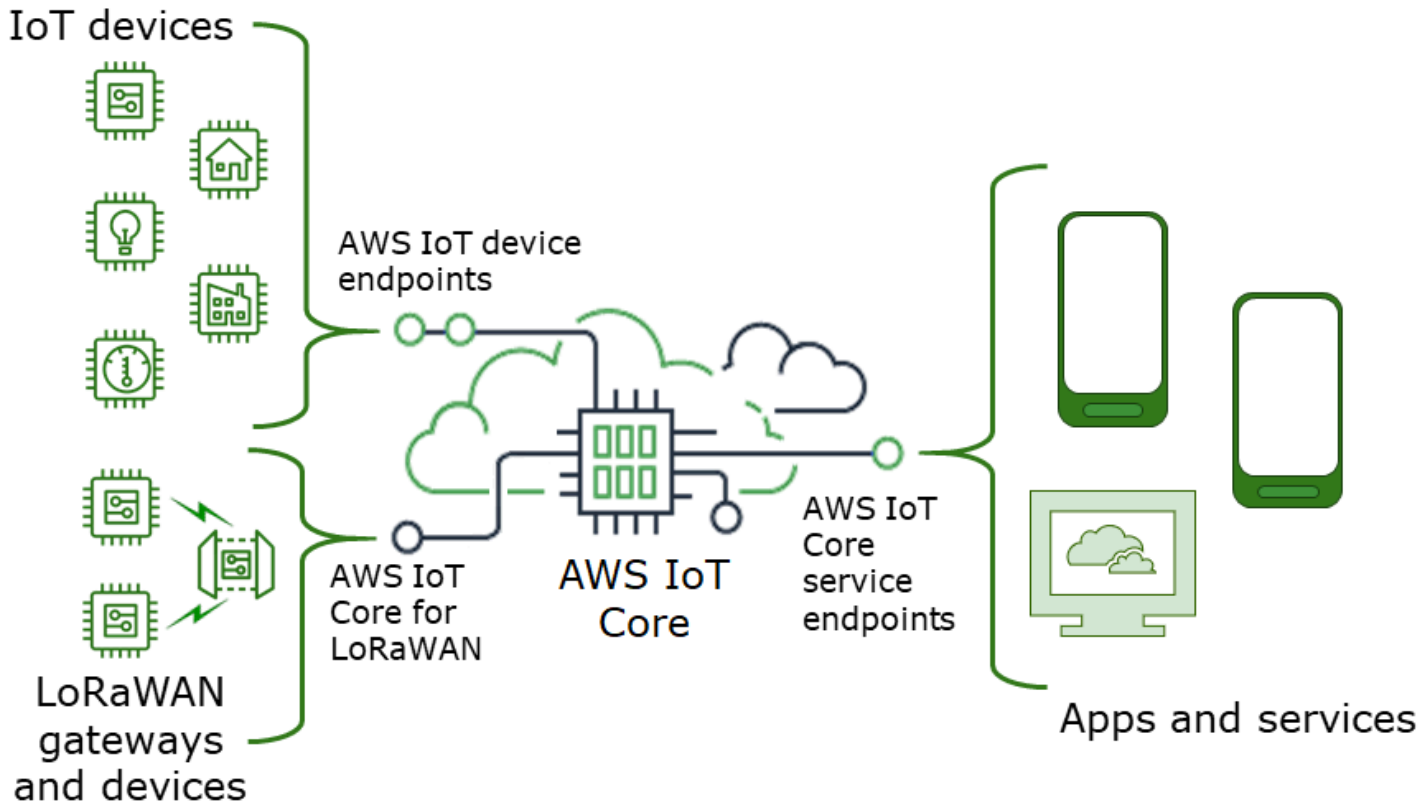
```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key
~/certs/raspberrypi-private.pem.key --thingName RaspberryPi --clientId
RaspberryPi
```

嘗試碰觸感應器、將其放入花盆中，或將其放入杯水中，以查看感應器如何回應各種濕度。如有需要，您可以在 MoistureSensorRule 中變更閾值。當濕度感應器讀數低於規則 SQL 查詢陳述式中指定的值時， 會將訊息 AWS IoT 發佈至 Amazon SNS 主題。您應該會收到包含濕度和溫度資料的電子郵件訊息。

在您驗證收到來自 Amazon SNS 的電子郵件訊息後，請按 CTRL+C 來停止 Python 程式。Python 程式傳送的訊息量不足以產生費用，但最佳實務是在完成時停止程式。

連線至 AWS IoT Core

AWS IoT Core 支援與 IoT 裝置、無線閘道、服務和應用程式的連線。裝置會連線至 ， AWS IoT Core 以便將資料傳送至 AWS IoT 服務和其他裝置並從中接收資料。應用程式和其他 服務也會連線至 AWS IoT Core ，以控制和管理 IoT 裝置，並處理 IoT 解決方案中的資料。本節說明如何 AWS IoT Core 針對 IoT 解決方案的每個層面，選擇與 連線和通訊的最佳方式。



有幾種方式可以與 互動 AWS IoT。應用程式和服務可以使用 AWS IoT Core：控制平面端點，而裝置可以使用 AWS IoT Core AWS IoT 裝置端點或 AWS IoT Core for LoRaWAN 區域和端點連線到 。

AWS IoT Core：控制平面端點

AWS IoT Core - 控制 平面端點可讓您存取控制和管理 AWS IoT 解決方案的 函數。

- 端點

  AWS IoT Core - 控制平面和AWS IoT Core Device Advisor 控制平面端點是 AWS IoT Core 端點與配額 中列出的特定區域。端點的格式如下。

端點用途	端點格式	服務
AWS IoT Core：控制平面	AWS IoT 控制平面端點	AWS IoT 控制平面 API
AWS IoT Core Device Advisor - 控制平面	`api.iotdeviceadvisor.` *`aws-region`* `.amazonaws.com`	AWS IoT Core Device Advisor 控制平面 API

- IPv4 端點 — IPv4 端點僅支援 IPv4 流量，且適用於所有區域。

  IPv4 端點使用以下命名慣例：

  ```
  iot.aws-region.amazonaws.com
  ```

  例如，us-east-1 區域的 IPv4 端點名稱為 `iot.us-east-1.amazonaws.com`。

- 雙堆疊 (IPv4 和 IPv6) 端點 — 雙堆疊端點支援 IPv4 和 IPv6 流量。向雙堆疊端點提出請求時，根據網路和用戶端使用的通訊協定，端點 URL 會解析為 IPv6 或 IPv4 地址。

  雙堆疊端點使用以下命名慣例：

  ```
  iot.aws-region.api.aws
  ```

  例如，us-east-1 區域的雙堆疊端點名稱為 `iot.us-east-1.api.aws`。

- 軟體開發套件和工具

  AWS SDKs為 AWS IoT Core APIs 和其他 AWS 服務的 APIs 提供特定語言的支援。AWS Mobile SDKs 為應用程式開發人員提供 AWS IoT Core API 和其他行動裝置上 AWS 服務的平台特定支援。

  AWS CLI 提供命令列存取 AWS IoT 服務端點提供的函數。AWS 適用於 PowerShell 的工具提供工具來管理 PowerShell 指令碼環境中 AWS 的服務和資源。

- 身分驗證

  服務端點使用 IAM 使用者和 AWS 登入資料來驗證使用者。

- 進一步了解

如需詳細資訊和軟體開發套件參考的連結，請參閱 the section called “連線至 AWS IoT Core 服務端點”。

AWS IoT 裝置端點

AWS IoT 裝置端點支援 IoT 裝置與 之間的通訊 AWS IoT。

- 端點

  裝置端點支援 AWS IoT Core 和 AWS IoT Device Management 函數。它們專屬於 AWS 帳戶 ，您可以使用 describe-endpoint命令查看它們的內容。

端點用途	端點格式	服務
AWS IoT Core：資料平面	請參閱 ???。	AWS IoT 資料平面 API
AWS IoT Device Management：任務資料	請參閱 ???。	AWS IoT 任務資料平面 API
AWS IoT Device Advisor - 資料平面	請參閱 ???。	不適用
AWS IoT Device Management：安全通道	`api.tunneling.iot.` *`aws-region`* `.amazonaws.com`	AWS IoT 安全通道 API

  如果您使用雙堆疊端點 (IPv4 和 IPv6) 進行資料平面操作，請使用 `iot:Data-ATS`端點類型。`iot:Jobs` 只能用於 IPv4。如需這些端點及其支援之功能的詳細資訊，請參閱 the section called “AWS IoT 裝置資料和服務端點”。

- 軟體開發套件

  AWS IoT 裝置SDKs為訊息佇列遙測傳輸 (MQTT) 和 WebSocket 安全 (WSS) 通訊協定提供特定語言的支援，這些通訊協定是裝置用來與之通訊 AWS IoT。 AWS 行動 SDKs也支援 MQTT 裝置通訊、 AWS IoT APIs 和其他行動裝置上 AWS 服務的 APIs。

- 身分驗證

裝置端點使用 X.509 憑證或具有憑證 AWS 的 IAM 使用者來驗證使用者。

- 進一步了解

  如需詳細資訊和軟體開發套件參考的連結，請參閱 the section called “AWS IoT 裝置 SDKs”。

AWS IoT Core for LoRaWAN 閘道和裝置

AWS IoT Core for LoRaWAN 會將無線閘道和裝置連接到其中 AWS IoT Core。

- 端點

  AWS IoT Core for LoRaWAN 會管理與帳戶和區域特定 AWS IoT Core 端點的閘道連線。閘道可以連接到您帳戶的組態和更新伺服器 (CUPS) 端點， AWS IoT Core 供 LoRaWAN 提供。

端點用途	端點格式	服務
組態與更新伺服器 (CUPS)	*account-specific-prefix* .cups.lorawan. *aws-region* .amazonaws.com:443	與 AWS IoT Core for LoRaWAN 提供的組態和更新伺服器的閘道通訊
LoRaWAN 網路伺服器 (LNS)	*account-specific-prefix* .gateway.lorawan. *aws-region* .amazonaws.com:443	與 AWS IoT Core for LoRaWAN 提供的 LoRaWAN 網路伺服器的閘道通訊

- 軟體開發套件

  AWS 開發套件支援 AWS IoT Core 為 LoRaWAN 建置的 AWS IoT 無線 API。如需詳細資訊，請參閱AWS 開發套件與工具組。

- 身分驗證

  AWS IoT Core for LoRaWAN 裝置通訊使用 X.509 憑證來保護與 的通訊 AWS IoT。

- 進一步了解

  如需設定和連接無線裝置的詳細資訊，請參閱 AWS IoT Core for LoRaWAN 區域和端點。

連線至 AWS IoT Core 服務端點

您可以使用 AWS CLI、您偏好語言的 AWS SDK，或直接呼叫 REST API 來存取AWS IoT Core - 控制 平面的功能。我們建議您使用 AWS CLI 或 AWS 開發套件與 互動， AWS IoT Core 因為它們包含呼叫 AWS 服務的最佳實務。直接呼叫 REST API 是一個選項，但您必須提供必要的安全憑證以便存取 API。

> **Note**
>
> IoT 裝置應使用 AWS IoT 裝置 SDKs。裝置SDKs已針對在裝置上使用進行最佳化、支援與 進行 MQTT 通訊 AWS IoT，以及支援裝置最常使用的 AWS IoT APIs。如需裝置軟體開發套件及其提供之功能的詳細資訊，請參閱 AWS IoT 裝置 SDKs。
> 行動裝置應使用 AWS 行動 SDKs。Mobile SDKs 支援 AWS IoT APIs、MQTT 裝置通訊，以及 AWS 行動裝置上其他服務APIs。如需 Mobile 軟體開發套件及其提供之功能的詳細資訊，請參閱 AWS 行動 SDKs。

您可以在 Web 和行動應用程式中使用 AWS Amplify 工具和資源，以便更輕鬆地連線 AWS IoT Core。如需使用 Amplify AWS IoT Core 連線至 的詳細資訊，請參閱 Amplify 文件中的 PubSub。

下列各節說明您可以用來開發和與 AWS IoT 和其他 AWS 服務互動SDKs。如需可用於建置和管理應用程式之 AWS 工具和開發套件的完整清單 AWS，請參閱建置工具 AWS。

AWS CLI 適用於 的 AWS IoT Core

AWS CLI 提供 AWS APIs命令列存取。

- 安裝

  如需如何安裝 的資訊 AWS CLI，請參閱安裝 AWS CLI。
- 身分驗證

  AWS CLI 使用來自 的登入資料 AWS 帳戶。
- 參考資料

  如需 AWS IoT Core 這些服務 AWS CLI 命令的相關資訊，請參閱：
  - AWS CLI IoT 的命令參考
  - AWS CLI IoT 資料的命令參考

- AWS CLI IoT 任務資料的命令參考
- AWS CLI IoT 安全通道的命令參考

如需在 PowerShell 指令碼環境中管理 AWS 服務和資源的工具，請參閱 AWS Tools for PowerShell。

AWS SDKs

透過 AWS SDKs，您的應用程式和相容裝置可以呼叫 AWS IoT APIs和其他 APIs。 AWS 本節提供 AWS IoT Core 服務 API 的 AWS SDKs APIs 參考文件的連結。

AWS SDKs 支援這些 AWS IoT Core APIs

- AWS IoT
- AWS IoT 資料平面
- AWS IoT 任務資料平面
- AWS IoT 安全通道
- AWS IoT 無線

C++

若要安裝 適用於 C++ 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 遵循適用於 C++ 的 AWS 開發套件入門中的指示

   這些指示說明如何：
   - 從來源檔案安裝和建置軟體開發套件
   - 提供憑證以搭配您的 AWS 帳戶使用軟體開發套件
   - 在您的應用程式或服務中初始化和關閉軟體開發套件
   - 建立一個 CMake 專案來建置應用程式或服務
2. 建立並執行範例應用程式。如需使用適用於 C++ 的 AWS SDK 的範例應用程式，請參閱 適用於 C++ 的 AWS SDK 程式碼範例。

適用於 C++ 的 AWS SDK 支援的 AWS IoT Core 服務文件

- AWS::IoTClient" 參考文件
- Aws::IoTDataPlane::IoTDataPlaneClient 參考文件

- Aws::IoTJobsDataPlane::IoTJobsDataPlaneClient 參考文件
- Aws::IoTSecureTunneling::IoTSecureTunnelingClient 參考文件

Go

若要安裝 適用於 Go 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 遵循 入門 適用於 Go 的 AWS SDK中的指示

   這些指示說明如何：

   - 安裝 適用於 Go 的 AWS SDK
   - 取得軟體開發套件的存取金鑰以存取您的 AWS 帳戶
   - 將套件匯入我們應用程式或服務的原始程式碼
2. 建立並執行範例應用程式。如需使用 適用於 Go 的 AWS SDK的範例應用程式，請參閱 適用於 Go 的 AWS SDK 程式碼範例。

適用於 Go 的 AWS SDK 支援的 AWS IoT Core 服務文件

- 物聯網參考文件
- IoTDataPlane 參考文件
- IoTJobsDataPlane 參考文件
- IoTSecureTunneling 參考文件

Java

若要安裝 適用於 Java 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 遵循 入門 AWS SDK for Java 2.x中的指示

   這些指示說明如何：

   - 註冊 AWS 並建立 IAM 使用者
   - 下載軟體開發套件
   - 設定 AWS 登入資料和區域
   - 使用軟體開發套件搭配 Apache Maven
   - 使用軟體開發套件搭配 Gradle

2. 使用其中一個 AWS SDK for Java 2.x 程式碼範例來建立和執行範例應用程式。
3. 檢閱軟體開發套件 API 參考文件

適用於 Java 的 AWS SDK 支援的 AWS IoT Core 服務的文件

- IotClient 參考文件
- IotDataPlaneClient 參考文件
- IotJobsDataPlaneClient 參考文件
- IoTSecureTunnelingClient 參考文件

JavaScript

若要安裝 適用於 JavaScript 的 AWS SDK 並使用它來連線至 AWS IoT：

1. 遵循設定 適用於 JavaScript 的 AWS SDK 中的指示。這些指示適用於在瀏覽器 適用於 JavaScript 的 AWS SDK 中使用 和搭配 Node.JS 使用 。請確定您遵循適用於安裝的指示。

   這些指示說明如何：

   - 查看先決條件
   - 安裝適用於 JavaScript 的軟體開發套件
   - 載入適用於 JavaScript 的軟體開發套件
2. 建立並執行範例應用程式，以開始使用軟體開發套件，如您環境的入門選項所描述。
   - 開始使用瀏覽器中適用於 JavaScript 的AWS SDK，或
   - 開始使用 Node.js 中適用於 JavaScript 的AWS SDK

適用於 JavaScript 的 AWS SDK 支援的 AWS IoT Core 服務文件

- `AWS.Iot reference documentation`
- `AWS.IotData reference documentation`
- `AWS.IotJobsDataPlane reference documentation`
- `AWS.IotSecureTunneling reference documentation`

.NET

若要安裝 適用於 .NET 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 遵循設定 適用於 .NET 的 AWS SDK 環境中的指示
2. 遵循設定 適用於 .NET 的 AWS SDK 專案中的指示

   這些指示說明如何：
   - 啟動新的專案
   - 取得和設定 AWS 登入資料
   - 安裝 AWS SDK 套件
3. 在適用於 .NET 的 AWS SDK 中使用 AWS 服務中建立並執行其中一個範例程式
4. 檢閱軟體開發套件 API 參考文件

適用於 .NET 的 AWS SDK 支援的 AWS IoT Core 服務文件

- Amazon.IoT.Model 參考文件
- Amazon.IotData.Model 參考文件
- Amazon.IoTJobsDataPlane.Model 參考文件
- Amazon.IoTSecureTunneling.Model 參考文件

PHP

若要安裝 適用於 PHP 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 遵循 第 3 適用於 PHP 的 AWS SDK 版入門中的指示

   這些指示說明如何：
   - 查看先決條件
   - 安裝軟體開發套件
   - 將軟體開發套件套用於 PHP 指令碼
2. 使用其中一個適用於 PHP 的 AWS SDK 第 3 版程式碼範例來建立和執行範例應用程式

適用於 PHP 的 AWS SDK 支援的 AWS IoT Core 服務文件

- IoTClient 參考文件

- IoTDataPlaneClient 參考文件
- IoTJobsDataPlaneClient 參考文件
- IoTSecureTunnelingClient 參考文件

Python

若要安裝 適用於 Python (Boto3) 的 AWS SDK 並使用它來連接至 AWS IoT：

1. 按照適用於 Python (Boto3) 的 AWS SDK 快速入門中的指示操作

   這些指示說明如何：

   - 安裝軟體開發套件
   - 設定軟體開發套件
   - 在您的程式碼中使用軟體開發套件

2. 建立並執行使用 適用於 Python (Boto3) 的 AWS SDK的範例程式

   此程式會顯示帳戶目前設定的日誌記錄選項。安裝軟體開發套件並為您的帳戶進行設定之後，您應該可以執行此程式。

   ```
   import boto3
   import json

   # initialize client
   iot = boto3.client('iot')

   # get current logging levels, format them as JSON, and write them to stdout
   response = iot.get_v2_logging_options()
   print(json.dumps(response, indent=4))
   ```

   如需此範例中所使用函數的詳細資訊，請參閱 the section called “設定 AWS IoT 記錄”。

適用於 Python (Boto3) 的 AWS SDK 支援的 AWS IoT Core 服務文件

- 物聯網參考文件
- IoTDataPlane 參考文件
- IoTJobsDataPlane 參考文件
- IoTSecureTunneling 參考文件

Ruby

若要安裝 適用於 Ruby 的 AWS SDK 並使用它來連接至 AWS IoT：

- 遵循 入門 適用於 Ruby 的 AWS SDK中的指示

  這些指示說明如何：

  - 安裝軟體開發套件
  - 設定軟體開發套件
- 建立和執行 Hello World 教學課程

適用於 Ruby 的 AWS SDK 支援的 AWS IoT Core 服務文件

- Aws::IoT::Client 參考文件
- Aws::IoTDataPlane::Client 參考文件
- Aws::IoTJobsDataPlane::Client 參考文件
- Aws::IoTSecureTunneling::Client 參考文件

AWS 行動 SDKs

AWS Mobile SDK SDKs 為行動應用程式開發人員平台特定的 AWS IoT Core 服務 APIs、使用 MQTT 的 IoT 裝置通訊，以及其他 AWS 服務的 APIs 提供支援。

Android

適用於 Android 的 AWS Mobile SDK

適用於 Android 的 AWS Mobile SDK 包含程式庫、範例和文件，供開發人員使用 建置連線的行動應用程式 AWS。此 SDK 也包含對 MQTT 裝置通訊和呼叫 AWS IoT Core 服務的 APIs的支援。如需詳細資訊，請參閱下列內容：

- GitHub 上適用於 Android 的AWS Mobile SDK
- AWS 適用於 Android 的 Mobile SDK 讀我檔案
- AWS 適用於 Android 的 Mobile SDK 範例
- AWS 適用於 Android 的 SDK API 參考
- AWSIoTClient 類別參考文件

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS 是開放原始碼軟體開發套件，根據 Apache Open Source 授權分佈。適用於 iOS 的 SDK 提供程式庫、程式碼範例和文件，協助開發人員使用 建置連線的行動應用程式 AWS。此 SDK 也包含對 MQTT 裝置通訊和呼叫 AWS IoT Core 服務的 APIs的支援。如需詳細資訊，請參閱下列內容：

- AWS Mobile SDK for iOS GitHub 上的
- AWS 適用於 iOS 的 SDK 讀我檔案
- AWS 適用於 iOS 的 SDK 範例
- AWS IoT 適用於 iOS 的 AWS SDK 中的類別參考文件

AWS IoT Core 服務的 REST APIs

您可以使用 HTTP 請求直接呼叫 AWS IoT Core 服務的 REST APIs。

- 端點 URL

  公開 AWS IoT Core 服務之 REST API 的服務端點會因區域而有所不同，並列於 AWS IoT Core 端點和配額中。您必須針對具有您要存取之 AWS IoT 資源的區域使用 端點，因為 AWS IoT 資源是區域特定的。

- 身分驗證

  AWS IoT Core 服務的 REST APIs 使用 AWS IAM 登入資料進行身分驗證。如需詳細資訊，請參閱《 AWS 一般參考》中的簽署 AWS API 請求。

- API 參考

  如需 AWS IoT Core 服務 REST APIs 提供的特定函數的相關資訊，請參閱：

  - 適用於 IoT 的 API 參考。
  - 適用於 IoT 資料的 API 參考。
  - 適用於 IoT 任務資料的 API 參考。
  - 適用於 IoT 安全通道的 API 參考。

將裝置連接至 AWS IoT

裝置透過 連線至 AWS IoT 和其他 服務 AWS IoT Core。透過 AWS IoT Core，裝置會使用您帳戶專屬的裝置端點來傳送和接收訊息。the section called “AWS IoT 裝置 SDKs” 支援使用 MQTT 和 WSS 通訊協定的裝置通訊。如需裝置可使用之通訊協定的詳細資訊，請參閱 the section called “裝置通訊協定”。

訊息代理程式

AWS IoT 透過訊息代理程式管理裝置通訊。裝置和用戶端會將訊息發佈至訊息代理程式，並訂閱訊息代理程式發佈的訊息。訊息由應用程式定義的主題識別。當訊息代理程式收到裝置或用戶端發佈的訊息時，會將該訊息重新發佈至已訂閱該訊息之主題的裝置和用戶端。訊息中介裝置也會轉送訊息到 AWS IoT 規則引擎，該引擎可以對訊息的內容採取行動。

AWS IoT 訊息安全性

AWS IoT 要使用 the section called “X.509 用戶端憑證”和AWS 簽章 V4 進行身分驗證的裝置連線。裝置通訊由 TLS 1.3 版保護，並 AWS IoT 要求裝置在連線時傳送伺服器名稱指示 (SNI) 延伸。如需詳細資訊，請參閱 中的 Transport Security AWS IoT。

AWS IoT 裝置資料和服務端點

> **Important**
>
> 您可以在裝置中快取或儲存端點。這意味著您不需要在每次連接新裝置時查詢 `DescribeEndpoint` API。為您的帳戶 AWS IoT Core 建立端點之後，端點不會變更。

每個帳戶都有數個帳戶獨有的裝置端點，並支援特定的 IoT 功能。 AWS IoT 裝置資料端點支援專為 IoT 裝置通訊需求設計的發佈/訂閱通訊協定；不過，如果其他用戶端的應用程式需要這些端點提供的特殊功能，例如應用程式和服務，也可以使用此界面。 AWS IoT 裝置服務端點支援以裝置為中心的存取安全性和管理服務。

若要了解您帳戶的裝置資料端點，您可以在 AWS IoT Core 主控台的設定頁面中找到它。

若要了解您帳戶的裝置端點特定用途，包括裝置資料端點，請使用此處顯示的 describe-endpoint CLI 命令或 `DescribeEndpoint` REST API，並提供下表中的 *`endpointType`* 參數值。

```
aws iot describe-endpoint --endpoint-type endpointType
```

這個命令會傳回下列格式的 *iot-endpoint*：*account-specific-prefix*.iot.*aws-region*.amazonaws.com。

每個客戶都有一個 iot:Data-ATS 和一個 iot:Data 端點。每個端點都使用 X.509 憑證來驗證用戶端。強烈建議客戶使用較新的 iot:Data-ATS 端點類型，以避免與普遍不信任 Symantec 憑證授權單位相關的問題。我們為裝置提供 iot:Data 端點，以便從使用 VeriSign 憑證的舊端點擷取資料，藉此提供回溯相容性。如需詳細資訊，請參閱伺服器身分驗證。

AWS IoT 裝置端點

端點用途	***endpointType*** 值	說明
AWS IoT Core：資料平面操作	iot:Data-ATS	用來將資料傳送至訊息代理程式、Device Shadow 和 AWS IoT的規則引擎元件，以及從中接收資料。 iot:Data-ATS 傳回 ATS 簽署的資料端點。
AWS IoT Core：資料平面操作 (舊式)	iot:Data	iot:Data 會傳回 VeriSign 簽署的資料端點，以提供回溯相容性。Symantec (iot:Data) 端點不支援 MQTT 5。
AWS IoT Core 登入資料存取	iot:CredentialProvider	用於將裝置的內建 X.509 憑證交換為臨時憑證，以直接與其他 AWS 服務連線。如需連線至其他 AWS 服務的詳細資訊，請參閱授權直接呼叫至 AWS 服務。
AWS IoT Device Management：任務資料操作	iot:Jobs	用來讓裝置使用 AWS IoT Jobs Device HTTPS APIs 與 Jobs 服務互動。 iot:Jobs 僅適用於 IPv4。如果您使用的是雙堆疊端點 (IPv4

端點用途	*endpointType* 值	說明
		和 IPv6)，請使用 iot:Data-ATS 端點類型。
AWS IoT Device Advisor 操作	iot:DeviceAdvisor	用於使用 Device Advisor 測試裝置的測試端點類型。如需詳細資訊，請參閱???。
AWS IoT Core data beta （預覽）	iot:Data-Beta	保留給 測試版的端點類型。如需其目前使用方式的詳細資訊，請參閱 ???。

您也可以使用自己的完整網域名稱 (FQDN)，例如 *example.com*，以及相關聯的伺服器憑證， AWS IoT 使用 將裝置連線至 the section called “網域組態”。

AWS IoT 裝置 SDKs

AWS IoT 裝置SDKs可協助您將 IoT 裝置連線至 ， AWS IoT Core 並且支援透過 WSS 通訊協定的 MQTT 和 MQTT。

AWS IoT 裝置 SDKs 與 AWS SDKs 不同之處在於 AWS IoT ，裝置 SDKs支援 IoT 裝置的特殊通訊需求，但不支援 AWS SDKs 支援的所有服務。 AWS IoT 裝置SDKs與支援所有 AWS 服務的 AWS SDKs 相容；不過，它們使用不同的身分驗證方法並連線到不同的端點，這可能會導致在 IoT 裝置上使用 AWS SDKs 變得不切實際。

行動裝置

同時the section called “AWS 行動 SDKs”支援 MQTT 裝置通訊、部分 AWS IoT 服務 APIs，以及其他服務的 APIs AWS 。如果您是在支援的行動裝置上進行開發，請檢閱其軟體開發套件，以確認這是否為開發 IoT 解決方案的最佳選擇。

C++

AWS IoT C++ 裝置 SDK

AWS IoT C++ 裝置 SDK 可讓開發人員使用 AWS 和 AWS IoT Core 服務的 APIs來建置連線的應用程式。此 SDK 特別是為並未受限於資源的裝置所設計，需要訊佇列、多重執行緒支援、最新語言功能等進階功能。如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上的裝置 SDK C++ v2
- AWS IoT 裝置 SDK C++ v2 讀我檔案
- AWS IoT 裝置 SDK C++ v2 範例
- AWS IoT 裝置 SDK C++ v2 API 文件

Python

AWS IoT 適用於 Python 的裝置 SDK

適用於 Python 的 AWS IoT 裝置 SDK 可讓開發人員撰寫 Python 指令碼，以使用其裝置透過 MQTT 或透過 WebSocket Secure (WSS) 通訊協定的 MQTT 存取 AWS IoT 平台。透過將其裝置連接到 AWS IoT Core 服務的 APIs，使用者可以安全地使用訊息中介裝置、規則和 Device Shadow 服務，該服務 AWS IoT Core 提供 和 ，並搭配其他 AWS 服務，例如 AWS Lambda、Amazon Kinesis 和 Amazon S3 等。

- AWS IoT GitHub 上的適用於 Python v2 的裝置 SDK
- AWS IoT 適用於 Python v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 Python v2 的 裝置 SDK 範例
- AWS IoT 適用於 Python v2 的 裝置 SDK API 文件

JavaScript

AWS IoT 適用於 JavaScript 的裝置 SDK

適用於 JavaScript 的 AWS IoT 裝置 SDK 可讓開發人員撰寫 JavaScript 應用程式，透過 WebSocket 通訊協定 AWS IoT Core 使用 MQTT 或 MQTT 存取 APIs。可用於 Node.js 環境和瀏覽器應用程式中。如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上適用於 JavaScript v2 的裝置 SDK
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK 範例
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK API 文件

Java

AWS IoT 適用於 Java 的裝置 SDK

適用於 Java 的 AWS IoT 裝置開發套件可讓 Java 開發人員 AWS IoT Core 透過 MQTT 或 MQTT 透過 WebSocket 通訊協定存取 的 APIs。軟體開發套件支援 Device Shadow 服務。您可以使用 HTTP 方法來存取影子，包括 GET、UPDATE 與 DELETE。此軟體開發套件亦支援簡化的影子存取模式，開發人員只需使用 getter 和 setter 方法，即可與影子交換資料，而無需將任何 JSON 文件序列化或還原序列化。如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上的適用於 Java 的 裝置 SDK v2
- AWS IoT 適用於 Java v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 Java 的 裝置 SDK v2 範例
- AWS IoT 適用於 Java v2 的 裝置 SDK API 文件

Embedded C

AWS IoT 適用於 Embedded C 的 裝置 SDK

> **Important**
>
> 此 SDK 適合經驗豐富的嵌入式軟體開發人員使用。

適用於 Embedded C 的 AWS IoT Device SDK (C-SDK) 是 MIT 開放原始碼授權下的 C 來源檔案集合，可用於內嵌應用程式，以安全地將 IoT 裝置連線至 AWS IoT Core。它包含 MQTT、JSON Parser 和 AWS IoT Device Shadow 程式庫等。它以原始碼形式分配，並且將會與應用程式碼、其他程式庫，以及 (可選的) RTOS (即時作業系統)，一起內建於客戶韌體中。

通常以需要最佳化 C 語言執行時間的資源限制裝置 適用於 Embedded C 的 AWS IoT Device SDK 為目標。您可以在任何作業系統上使用軟體開發套件，並將其裝載在任何處理器類型 (例如 MCU 和 MPU) 上。如果您的裝置有足夠的記憶體和處理資源，我們建議您使用其他裝置 AWS IoT 和行動 SDKs，例如適用於 C++、Java、JavaScript 或 Python 的 AWS IoT 裝置 SDK。

如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上適用於內嵌 C 的裝置 SDK
- AWS IoT 適用於 Embedded C Readme 的 裝置 SDK
- AWS IoT 適用於內嵌 C 的 裝置 SDK 範例

裝置通訊協定

AWS IoT Core 支援使用 MQTT 和 MQTT over WebSocket Secure (WSS) 通訊協定來發佈和訂閱訊息的裝置和用戶端、使用 HTTPS 通訊協定來發佈訊息的裝置和用戶端，以及使用 Direct Messaging API 將訊息傳送至特定連線用戶端。所有通訊協定皆支援 IPv4 和 IPv6。本節說明裝置和用戶端的不同連線選項。

TLS 通訊協定版本

AWS IoT Core 使用 TLS 1.2 版和 TLS 1.3 版來加密所有通訊。您可以在網域組態中設定 TLS 設定，為您的端點設定其他 TLS 政策版本。將裝置連線至 時 AWS IoT Core，用戶端可以傳送伺服器名稱指示 (SNI) 延伸，這是多帳戶註冊、可設定端點、自訂網域和 VPC 端點等功能的必要項目。如需詳細資訊，請參閱 AWS IoT中的傳輸安全性。

AWS IoT 裝置 SDKs 支援 MQTT 或是經 WSS 的 MQTT，並支援用戶端連線的安全性需求。建議您使用 AWS IoT 裝置 SDKs 將用戶端連線至 AWS IoT。

通訊協定、連接埠映射和身分驗證

裝置或用戶端連線至訊息中介裝置的方式，可使用 身分驗證類型進行設定。根據預設，或未傳送 SNI 延伸時，身分驗證方法是根據裝置使用的應用程式通訊協定、連接埠和應用程式層通訊協定交涉 (ALPN) TLS 延伸。下表列出根據連接埠、連接埠和 ALPN 預期的身分驗證。

通訊協定、身分驗證和連接埠對應

通訊協定	支援的操作	身分驗證	站點	ALPN 通訊協定名稱
MQTT over WebSocket	發佈、訂閱	Signature 第 4 版	443	N/A
MQTT over WebSocket	發佈、訂閱	自訂身分驗證	443	N/A
MQTT	發佈、訂閱	X.509 用戶端憑證	443†	`x-amzn-mqtt-ca`
MQTT	發佈、訂閱	X.509 用戶端憑證	8883	N/A
MQTT	發佈、訂閱	自訂身分驗證	443†	`mqtt`
HTTPS	僅發佈	Signature 第 4 版	443	N/A

通訊協定	支援的操作	身分驗證	站點	ALPN 通訊協定名稱
HTTPS	僅發佈	X.509 用戶端憑證	443†	`x-amzn-http-ca`
HTTPS	僅發佈	X.509 用戶端憑證	8443	N/A
HTTPS	僅發佈	自訂身分驗證	443	N/A

應用程式層通訊協定交涉 (ALPN)

†使用預設端點組態時，透過 X.509 用戶端憑證身分驗證在連接埠 443 上連線的用戶端，必須實作應用程式層通訊協定交涉 (ALPN) TLS 延伸，並使用用戶端傳送的 ALPN ProtocolNameList 中列出的 ALPN 通訊協定名稱，做為`ClientHello`訊息的一部分。ProtocolNameList

在連接埠 443 上，IoT:Data-ATS 端點支援 ALPN x-amzn-http-ca HTTP，但 IoT:Jobs 端點不支援。

ALPN x-amzn-mqtt-ca 在連接埠 8443 HTTPS 和連接埠 443 MQTT 上無法使用自訂身分驗證。

用戶端會連線至 AWS 帳戶其裝置端點。如需有關如何尋找帳戶的裝置端點的資訊，請參閱the section called “AWS IoT 裝置資料和服務端點”。

Note

AWS SDKs不需要整個 URL。它們只需要端點主機名稱，例如 pubsub.py GitHub 上適用於 Python 的 AWS IoT Device SDK 範例。如下表所示，傳遞完整 URL 可能會產生錯誤，例如主機名稱無效。

連線至 AWS IoT Core

通訊協定	端點或 URL
MQTT	*iot-endpoint*
透過 WSS 的 MQTT	wss://*iot-endpoint* /mqtt
HTTPS	https://*iot-endpoint* /topics

選擇裝置通訊的應用程式通訊協定

對於透過裝置端點進行大多數 IoT 裝置通訊，您會想要使用 Secure MQTT 或 MQTT over WebSocket Secure (WSS) 通訊協定；不過，裝置端點也支援 HTTPS。

下表比較 AWS IoT Core 如何使用兩個高階通訊協定 (MQTT 和 HTTPS) 進行裝置通訊。

AWS IoT 裝置通訊協定 side-by-side (MQTT 和 HTTPS)

功能	MQTT	HTTPS
發佈/訂閱支援	發佈和訂閱	僅發佈
開發套件支援	AWS 裝置 SDKs支援 MQTT 和 WSS 通訊協定	不支援軟體開發套件，但您可以使用特定語言的方法來發出 HTTPS 要求
服務品質支援	MQTT QoS 層級 0 和 1	QoS 的支援是透過傳遞查詢字串參數 ?qos=qos 來進行，其中值可以是 0 或 1。您可以新增此查詢字串，以發佈內含您所需之 QoS 值的訊息。
可接收裝置離線時所遺漏的訊息	是	否
clientId 欄位支援	是	否
裝置中斷連線偵測	是	否

功能	MQTT	HTTPS
安全通訊	是。請參閱 ???	是。請參閱 ???
主題定義	應用程式定義	應用程式定義
訊息資料格式	應用程式定義	應用程式定義
通訊協定負荷	較低	較高
耗電量	較低	較高

選擇裝置通訊的身分驗證類型

您可以使用可設定的端點來設定 IoT 端點的身分驗證類型。或者，使用預設組態，並判斷您的裝置如何使用應用程式通訊協定、連接埠和 ALPN TLS 延伸組合進行身分驗證。您選擇的身分驗證類型決定您的裝置在連線時將如何進行身分驗證 AWS IoT Core。有五種身分驗證類型：

X.509 憑證

使用 X.509 用戶端憑證來驗證裝置，以 AWS IoT Core 驗證裝置。此身分驗證類型適用於 Secure MQTT (MQTT over TLS) 和 HTTPS 通訊協定。

具有自訂授權方的 X.509 憑證

使用 X.509 用戶端憑證驗證裝置，並使用自訂授權方執行其他身分驗證動作，這會接收 X.509 用戶端憑證資訊。此身分驗證類型適用於 Secure MQTT (MQTT over TLS) 和 HTTPS 通訊協定。此身分驗證類型只能使用可設定的端點搭配 X.509 自訂身分驗證。沒有 ALPN 選項。

AWS Signature 第 4 版 (SigV4)

使用 Cognito 或您的後端服務來驗證裝置，以支援社交和企業聯合。此身分驗證類型適用於 MQTT over WebSocket Secure (WSS) 和 HTTPS 通訊協定。

自訂授權方

透過設定 Lambda 函數來驗證裝置，以處理傳送至其中的自訂身分驗證資訊 AWS IoT Core。此身分驗證類型適用於 Secure MQTT (MQTT over TLS)、HTTPS 和 MQTT over WebSocket Secure (WSS) 通訊協定。

預設

根據裝置使用的連接埠和/或應用程式層通訊協定交涉 (ALPN) 延伸來驗證裝置。不支援某些額外的身分驗證選項。如需詳細資訊，請參閱???。

下表顯示所有支援的身分驗證類型和應用程式通訊協定組合。

支援的身分驗證類型和應用程式通訊協定組合

身分驗證類型	安全 MQTT（透過 TLS 的 MQTT)	透過 WebSocket Secure (WSS) 的 MQTT	HTTPS	預設
X.509 憑證	✓		✓	
具有自訂授權方的 X.509 憑證	✓		✓	
AWS Signature 第 4 版 (SigV4)		✓	✓	
自訂授權方	✓	✓	✓	
預設				✓

連線持續時間限制

HTTPS 連線的持續時間不保證比接收和回應要求所需時間更長。

MQTT 連線持續時間取決於您使用的身分驗證功能。下表列出每項功能在理想條件下的連線持續時間上限。

依身分驗證功能分類的 MQTT 連線持續時間

功能	持續時間上限 *
X.509 用戶端憑證	1–2 週
自訂身分驗證	1–2 週
Signature 第 4 版	長達 24 小時

* 不保證

使用 X.509 憑證和自訂身分驗證時，連線持續時間沒有硬性限制，但可以短至幾分鐘。有多種原因可能導致連線中斷。下方清單中包含一些最常見的原因。

- Wi-Fi 可用性中斷
- 網際網路服務供應商 (ISP) 連線中斷
- 服務修補程式
- 服務部署
- 服務自動擴展
- 服務主機無法使用
- 負載平衡器問題和更新
- 用戶端錯誤

您的裝置必須實作偵測中斷連線和重新連線的策略。如需中斷連線事件的詳細資訊，以及處理前述事件的指導，請參閱 ??? 中的 ???。

MQTT

MQTT (訊息佇列遙測傳輸) 是輕量級且廣泛採用的訊息通訊協定，專為受限的裝置所設計。 AWS IoT Core 支援的 MQTT 以 MQTT v3.1.1 規格和 MQTT v5.0 規格 為基礎，但如 the section called “AWS IoT 與 MQTT 規格的差異” 所述具有若干差異。作為該標準的最新版本，MQTT 5 推出數項關鍵功能，使基於 MQTT 的系統更加強大，包括新的可擴展性增強功能、具備原因代碼回應的改良版錯誤報告、訊息和工作階段過期計時器，以及自訂使用者訊息標頭。如需 AWS IoT Core 支援的 MQTT 5 功能的詳細資訊，請參閱 MQTT 5 支援的功能。 AWS IoT Core 也支援跨 MQTT 版本 (MQTT 3 和 MQTT 5) 通訊。MQTT 3 發佈者可以將 MQTT 3 訊息傳送給可接收 MQTT 5 發佈訊息的 MQTT 5 訂閱者，反之亦然。

AWS IoT Core 支援使用 MQTT 通訊協定和透過 WSS 通訊協定的 MQTT，並由用戶端 ID 識別的裝置連線。AWS IoT 裝置 SDKs 支援這兩種通訊協定，並且是將裝置連接至 AWS IoT Core的建議方式。 AWS IoT 裝置 SDKs 支援裝置和用戶端連線至和存取 AWS IoT 服務所需的功能。裝置 SDKs 支援 AWS IoT 服務所需的身分驗證通訊協定，以及 MQTT 通訊協定和 MQTT over WSS 通訊協定所需的連線 ID 需求。如需如何使用 AWS IoT AWS 裝置SDKs連線至 的資訊，以及支援語言 AWS IoT 中範例的連結，請參閱 the section called “使用 AWS IoT 裝置 SDKs 與 MQTT 連線”。如需 MQTT 訊息之身分驗證方法和連接埠映射的詳細資訊，請參閱???。

雖然我們建議使用 AWS IoT 裝置SDKs來連線至 AWS IoT，但它們並非必要。不過，如果您不使用 AWS IoT 裝置SDKs，則必須提供必要的連線和通訊安全性。用戶端必須在連線請求中傳送伺服器名稱指示 (SNI) TLS 延伸。不包含 SNI 的連線嘗試將會被拒絕。如需詳細資訊，請參閱 中的 Transport Security AWS IoT。使用 IAM 使用者和 AWS 登入資料來驗證用戶端的用戶端必須提供正確的 Signature 第 4 版身分驗證。

用戶端連線後，您可以使用主控台或 APIs 來監控和管理其 MQTT 用戶端連線。如需詳細資訊，請參閱the section called “MQTT 連線管理”。

在本主題中：

- 使用 AWS IoT 裝置 SDKs 與 MQTT 連線
- MQTT 服務品質 (QoS) 選項
- MQTT 持久性工作階段
- MQTT 保留訊息
- MQTT Last Will and Testament (LWT) 訊息
- 使用 connectAttributes
- MQTT 5 支援的功能
- MQTT 5 屬性
- MQTT 原因代碼
- AWS IoT 與 MQTT 規格的差異
- MQTT 連線管理

使用 AWS IoT 裝置 SDKs 與 MQTT 連線

本節包含 AWS IoT 裝置 SDKs和範例程式原始碼的連結，說明如何將裝置連接至其中 AWS IoT。此處連結的範例應用程式示範如何使用 MQTT AWS IoT 通訊協定和透過 WSS 的 MQTT 連線至 。

> **Note**
>
> AWS IoT 裝置 SDKs 已發行 MQTT 5 用戶端。

C++

使用 AWS IoT C++ 裝置 SDK 連接裝置

- 以 C++ 撰寫的 MQTT 連線範例的應用程式原始碼
- AWS IoT GitHub 上適用於 C++ v2 的裝置 SDK

Python

使用 AWS IoT 適用於 Python 的裝置 SDK 來連接裝置

- 以 Python 撰寫的 MQTT 連線範例的應用程式原始碼
- AWS IoT GitHub 上的適用於 Python 的裝置 SDK v2

JavaScript

使用適用於 JavaScript 的 AWS IoT 裝置開發套件來連接裝置

- 以 JavaScript 撰寫的 MQTT 連線範例的應用程式原始碼
- GitHub 上適用於 JavaScript 的AWS IoT 裝置 SDK v2

Java

使用適用於 Java 的 AWS IoT 裝置開發套件來連接裝置

> **Note**
>
> Device AWS IoT SDK for Java v2 現在支援 Android 開發。如需詳細資訊，請參閱適用於 AWS IoT Android 的 裝置開發套件。

- 以 Java 撰寫的 MQTT 連線範例的應用程式原始碼
- AWS IoT GitHub 上的適用於 Java 的 裝置 SDK v2

Embedded C

使用適用於 Embedded C 的 AWS IoT 裝置開發套件來連接裝置

> **Important**
>
> 此 SDK 適合經驗豐富的嵌入式軟體開發人員使用。

- 以 Embedded C 撰寫的 MQTT 連線範例的應用程式原始碼
- AWS IoT GitHub 上適用於內嵌 C 的裝置 SDK

MQTT 服務品質 (QoS) 選項

AWS IoT 和 AWS IoT 裝置SDKs支援 MQTT 服務品質 (QoS) 層級 0和 1。MQTT 通訊協定會定義第三個 QoS 層級 2，但 AWS IoT 不支援。只有 MQTT 通訊協定支援 QoS 功能。HTTPS 透過傳遞查詢字串參數 `?qos=qos` 來支援 QoS，其中值可以是 0 或 1。

此表格說明各個 QoS 層級如何影響訊息發佈至代理程式的方式，以及訊息代理程式發佈訊息的方式。

具有 QoS 層級...	訊息為...	說明
QoS 層級 0	傳送零次或更多次	此層級應用於透過可靠通訊連結傳送的訊息，或者可能會錯過沒有問題的訊息。
QoS 層級 1	至少傳送一次，然後重複，直到接收到 PUBACK 回應	在寄件者收到 PUBACK 回應以表示成功傳遞之後，才會將訊息視為完整的。

MQTT 持久性工作階段

持久性工作階段會存放用戶端的訂閱和訊息，服務品質 (QoS) 為 1，而這些用戶端尚未確認此服務品質。該裝置重新連線至持久性工作階段時，工作階段會繼續執行，訂閱會復原，且在重新連線之前接收和儲存的未確認的訂閱訊息皆會傳送至用戶端。

儲存訊息的處理會記錄在 CloudWatch 指標和 CloudWatch Logs 中。如需有關將項目寫入 CloudWatch 和 CloudWatch Logs 的資訊，請參閱 訊息代理程式指標 和 佇列日誌項目。

建立持久性工作階段

在 MQTT 3 中，您可以傳送 `CONNECT` 訊息並將 `cleanSession` 旗標設定為 `0`，以建立 MQTT 持久性工作階段。如果傳送 `CONNECT` 訊息的用戶端沒有工作階段，則會建立新的持久性工作階段。如果用戶端的工作階段已經存在，用戶端會繼續現有的工作階段。若要建立全新工作階段，請傳送 `CONNECT` 訊息並將 `cleanSession` 旗標設定為 `1`，當用戶端中斷連線時，代理程式將不會儲存任何工作階段狀態。

在 MQTT 5 中，您可以設定 Clean Start 旗標和 Session Expiry Interval 來處理持久性會話。「全新啟動」可控制連線工作階段的開始和前一個工作階段的結束。當您設定 Clean Start = 1 時，系統會建立新的工作階段，任何既有的先前工作階段都會終止。當您設定Clean Start = 0 時，連線工作階段會繼續先前的工作階段 (若有)。工作階段過期間隔控制了連線工作階段的結束。工作階段過期間隔指定了工作階段在中斷連線後維持的時間 (以秒為單位的 4 位元組整數)。如果設定 Session Expiry interval =0，系統會在中斷連線時立即終止工作階段。如果 CONNECT 訊息中未指定工作階段過期間隔，則預設值為 0。

MQTT 5 全新啟動和工作階段過期

屬性值	說明
Clean Start= 1	建立一個新的工作階段，並終止先前的工作階段 (若有)。
Clean Start= 0	若有先前的工作階段，則恢復工作階段。
Session Expiry Interval> 0	維持工作階段。
Session Expiry interval= 0	不維持工作階段。

在 MQTT 5 中，如果您設定Clean Start =1 和 Session Expiry Interval >0，則等同於 MQTT 3 全新工作階段。如果您設定Clean Start =0 和 Session Expiry Interval >0，則等同於 MQTT 3 持久性工作階段。

> **Note**
>
> 不支援跨 MQTT 版本 (MQTT 3 和 MQTT 5) 持久性工作階段。MQTT 3 持久性工作階段無法恢復為 MQTT 5 工作階段，反之亦然。

持久性工作階段期間的操作

用戶端使用連線認可 (CONNACK) 訊息中的 sessionPresent 屬性，以判斷持久性工作階段是否存在。如果 sessionPresent 是 1，表示存在持久性工作階段，且用戶端的任何已儲存訊息會在用戶端收到 CONNACK 後傳遞至用戶端，如重新連線至持久性工作階段後的訊息流量所述。如果 sessionPresent 是 1，則用戶端不需要重新訂閱。但是，如果 sessionPresent 是 0，表示不存在持久性工作階段，而且用戶端必須重新訂閱其主題篩選條件。

在用戶端加入持久性工作階段之後，它可以繼續發佈訊息並訂閱主題篩選條件，各個操作上無需任何額外的旗標。

重新連線至持久性工作階段後的訊息流量

持久性工作階段代表用戶端與 MQTT 訊息代理程式之間的持續連線。當用戶端使用持久性工作階段連接至訊息代理程式時，訊息代理程式會儲存用戶端在連線期間所進行的所有訂閱。當用戶端中斷連線時，訊息代理程式會存放未認可的 QoS 1 訊息，以及發佈到用戶端所訂閱之主題的新 QoS 1 訊息。系統會根據帳戶限制儲存訊息。將捨棄超出上限的訊息將。如需持久性訊息限制的詳細資訊，請參閱 AWS IoT Core 端點和配額。當用戶端重新連線至持久性工作階段時，所有訂閱都會恢復，而且所有存放的訊息都會傳送至用戶端，其傳送速率為每秒最多 10 則訊息。在 MQTT 5 中，如果具有訊息過期間隔的傳出 QoS 1 在用戶端離線時過期，則在連線恢復之後，用戶端不會收到過期的訊息。

重新連線之後，存放的訊息會以每秒 10 個存放訊息為限的速率傳送至用戶端，以及任何目前的訊息流量，直到達到 `Publish requests per second per connection` 限制為止。由於已存放訊息的傳遞速率有限，如果工作階段有 10 個以上的存放訊息要在重新連線後傳遞，將需要數秒鐘才能傳遞所有存放訊息。

對於共用訂閱者，如果群組的至少一個訂閱者使用持久性工作階段，且沒有訂閱者在線上接收 QoS 1 訊息，則訊息會排入佇列。取消佇列訊息會以群組中每個作用中訂閱用戶每秒 20 則訊息的最大速度完成。如需詳細資訊，請參閱共用訂閱訊息佇列。

結束持久性工作階段

持久性工作階段可透過下列方式結束：

- 當持久性工作階段過期時間經過時。當訊息代理程式偵測到用戶端已中斷連線時 (包括用戶端中斷連線或連線逾時)，持久性工作階段過期計時器將會啟動。
- 用戶端傳送將 `cleanSession` 旗標設定為 1 的 `CONNECT` 訊息。
- 您可以使用主控台或 `DeleteConnection` API 手動中斷連接用戶端並清除工作階段。如需詳細資訊，請參閱the section called “MQTT 連線管理”。

在 MQTT 3 中，持久性工作階段到期時間預設值是一小時，這適用於帳戶中的所有工作階段。

在 MQTT 5 中，您可以設定 CONNECT 和 DISCONNECT 封包上的每個工作階段的工作階段過期間隔。

對於 DISCONNECT 封包的工作階段過期間隔：

- 如果目前工作階段的過期間隔為 0，則無法將 DISCONNECT 封包上的工作階段過期間隔設定為大於 0。
- 如果目前工作階段的過期間隔大於 0，且您將 DISCONNECT 封包上的工作階段過期間隔設為 0，則工作階段將於 DISCONNECT 結束。
- 否則，DISCONNECT 上的工作階段過期間隔會更新目前工作階段的過期間隔。

> **Note**
>
> 當工作階段結束時，等待傳送給用戶端的存放訊息會被捨棄；但即使無法傳送，仍會依照標準的簡訊費率計費。如需訊息定價的詳細資訊，請參閱 AWS IoT Core 定價。您可以設定過期時間間隔。

持久性工作階段過期後重新連線

如果用戶端在過期之前沒有重新連接至其持久性工作階段，工作階段將會結束並捨棄其存放的訊息。當用戶端在工作階段過期後重新連線，並將 `cleanSession` 旗標設為 `0` 時，服務會建立新的持久性工作階段。這個工作階段無法使用先前工作階段的任何訂閱或訊息，因為上一個工作階段過期時，這些訂閱或訊息將被捨棄。

持久性工作階段訊息費用

當訊息中介裝置傳送訊息給用戶端或離線持久性工作階段 AWS 帳戶 時，會向您的 收取訊息費用。當具有持久性工作階段的離線裝置重新連線並繼續其工作階段時，存放的訊息會傳遞至裝置，並再次向您的帳戶收費。如需訊息定價的詳細資訊，請參閱 AWS IoT Core 定價：簡訊。

使用標準限制增加程序，可以增加預設的持久性工作階段一小時過期時間。請注意，增加工作階段到期時間可能會增加您的訊息費用，因為額外的時間可能允許離線裝置存放更多訊息，而且將會按照標準簡訊速率向您的帳戶收取這些額外的訊息費用。工作階段過期時間為近似值，工作階段最多可以比帳戶限制長 30 分鐘；不過，工作階段不會短於帳戶限制。如需工作階段限制的詳細資訊，請參閱 AWS Service Quotas。

MQTT 保留訊息

AWS IoT Core 支援 MQTT 通訊協定中所述的`RETAIN`旗標。當用戶端在其發佈的 MQTT 訊息上設定 `RETAIN`旗標時， AWS IoT Core 會儲存訊息。然後可以將其傳送給新訂閱者，透過呼叫 `GetRetainedMessage` 操作對其進行檢索，並在 AWS IoT 主控台內予以檢視。

使用 MQTT 保留訊息的範例

- 作為初始組態訊息

  MQTT 保留訊息會在用戶端訂閱主題後傳送至用戶端。如果您希望訂閱主題的所有用戶端在訂閱後立即收到 MQTT 保留訊息，則可以發佈具有RETAIN標記集的組態訊息。只要發佈新的組態訊息，訂閱用戶端也會收到該組態的更新。

- 作為最後一個已知的狀態訊息

  裝置可以在目前狀態訊息上設定 RETAIN旗標，以便 AWS IoT Core 儲存它們。當應用程式連線或重新連線時，他們可以訂閱此主題，並在訂閱保留的訊息主題後立即取得最後報告的狀態。透過這種方式，可以避免等到來自裝置的下一個訊息才能查看目前狀態。

在本節中：

- 在 中使用 MQTT 保留訊息的一般任務 AWS IoT Core
- 計費和保留訊息
- 比較 MQTT 保留訊息和 MQTT 持久性工作階段
- MQTT 保留的訊息和 AWS IoT 裝置影子

在 中使用 MQTT 保留訊息的一般任務 AWS IoT Core

AWS IoT Core 會使用 RETAIN旗標集儲存 MQTT 訊息。這些保留訊息會像一般 MQTT 訊息一樣，傳送給已訂閱主題的所有用戶端，而且這些訊息也會存放起來，以傳送給該主題的新訂閱者。

MQTT 保留訊息需要特定的政策動作，才能授權用戶端存取這些訊息。如需保留訊息政策的使用範例，請參閱 保留訊息政策範例。

本節說明與保留訊息有關的常見操作。

- 建立保留訊息

  用戶端會決定當其發佈 MQTT 訊息時是否保留訊息。用戶端可以在使用 裝置 SDK 發佈訊息時設定 RETAIN旗標。應用程式和服務可以在使用 Publish動作發佈 MQTT 訊息時設定 RETAIN旗標。

  每個主題名稱只會保留一則訊息。已設定 RETAIN 旗標並發佈至主題的新訊息，會取代先前傳送至主題的任何現有保留訊息。

> **Note**
>
> 您無法發佈至具有 RETAIN旗標集的預留主題。

- 訂閱保留訊息主題

  用戶端會訂閱保留訊息主題，如同訂閱其他任何 MQTT 訊息主題一樣。訂閱保留訊息主題收到的保留訊息會設定 RETAIN 旗標。

  當用戶端將具有 0 位元組訊息承載的保留訊息發佈至保留訊息主題 AWS IoT Core 時，保留訊息會從 中刪除。訂閱保留訊息主題的用戶端也會收到 0 個位元組訊息。

  訂閱包含保留訊息主題的萬用字元主題篩選條件，可讓用戶端接收該主題的後續訊息。不過，保留的訊息不會在訂閱時傳送。

  > **Note**
  >
  > 若要在訂閱時接收保留的訊息，訂閱請求中的主題篩選條件必須完全符合保留的訊息主題。

  訂閱保留訊息主題時收到的保留訊息已設定 RETAIN 旗標。訂閱用戶端在訂閱後接收的保留訊息未設定 RETAIN 旗標。

- 擷取保留訊息

  當用戶端訂閱帶有保留訊息的主題時，系統就會自動將保留訊息傳遞給用戶端。若要讓用戶端在訂閱時即接收保留訊息，則必須訂閱保留訊息的確切主題名稱。訂閱內含保留訊息主題的萬用字元主題篩選條件，可讓用戶端接收向保留訊息主題發佈的後續訊息，但不會在訂閱時即傳遞保留訊息。

  服務和應用程式可以透過呼叫 `ListRetainedMessages` 和 `GetRetainedMessage` 來列出和擷取保留訊息。

  在未設定RETAIN旗標的情況下，不會阻止用戶端將訊息發佈至保留的訊息主題。這可能會導致非預期的結果，例如保留訊息與訂閱主題接收到的訊息不相符。

  使用 MQTT 5 時，如果保留的訊息已設定訊息過期間隔，且保留的訊息已經過期，則該主題的新訂閱者在訂閱成功後將不會收到保留的訊息。

- 列出保留訊息主題

您可以透過呼叫 ListRetainedMessages 列出保留訊息，且可以在 AWS IoT 主控台中檢視保留訊息。

- 取得保留訊息的詳細資訊

  您可以透過呼叫 GetRetainedMessage 取得保留訊息的詳細資訊，且可以在 AWS IoT 主控台中檢視這些資訊。

- 保留「Will」訊息

  裝置連線時建立的 MQTT Will 訊息，可以藉由在 Connect Flag bits 欄位中設定 Will Retain 旗標來加以保留。

- 刪除保留訊息

  裝置、應用程式和服務可以透過將具有RETAIN旗標集和空白 (0 位元組） 訊息承載的訊息發佈至要刪除之保留訊息的主題名稱，來刪除保留訊息。這類訊息會從 刪除保留的訊息 AWS IoT Core，並傳送給訂閱主題的用戶端，但 不會保留這些訊息 AWS IoT Core。

  您也可以透過在 AWS IoT 主控台中存取保留訊息，來刪除保留訊息。使用 AWS IoT 主控台刪除的保留訊息，也會將 0 個位元組的訊息傳送給已訂閱保留訊息主題的用戶端。

  保留訊息遭刪除後即無法還原。用戶端需要發佈新的保留訊息，以取代已刪除的訊息。

- 對保留訊息進行偵錯和疑難排解

  此 AWS IoT 主控台提供多種工具，可協助您針對保留訊息進行疑難排解：

  - Retained messages (保留訊息) 頁面

    AWS IoT 主控台中的 Retained messages (保留訊息) 頁面會提供分頁清單，列出由您帳戶在目前區域存放的保留訊息。在此頁面上，您可以：

    - 查看每則保留訊息的詳細資訊，例如訊息承載、QoS，以及保留訊息的接收時間。
    - 更新保留訊息的內容。
    - 刪除保留訊息。

  - MQTT test client (MQTT 測試用戶端)

    AWS IoT 主控台中的 MQTT test client (MQTT 測試用戶端) 頁面可以訂閱和發佈至 MQTT 主題。發佈選項可讓您在發佈的訊息上設定 RETAIN 旗標，以模擬裝置可能出現的行為。您也可以使用 MQTT 測試用戶端，從透過用戶端連線介面管理的連線用戶端監控訊息。如需管理用戶端連線的詳細資訊，請參閱 the section called “MQTT 連線管理”。

有些非預期的結果可能是保留訊息在其中實作方式的這些層面的結果 AWS IoT Core。

- 保留訊息限制

  當帳戶存放了保留訊息的最大數量時， 會 AWS IoT Core 傳回以 RETAIN 集發佈且承載大於 0 位元組的訊息的限流回應，直到刪除某些保留訊息且保留訊息計數低於限制為止。

- 保留訊息的傳遞順序

  我們不能保證保留訊息和訂閱訊息的傳遞順序。

計費和保留訊息

從用戶端、使用 AWS IoT 主控台或呼叫 發佈具有 RETAIN 旗標的訊息，`Publish`會產生AWS IoT Core 定價 - 訊息中所述的額外簡訊費用。

除了一般 API 使用費之外，透過用戶端、使用 AWS IoT 主控台或呼叫 來擷取保留的訊息`GetRetainedMessage`會產生簡訊費用。AWS IoT Core 定價：簡訊中說明了額外費用。

裝置意外中斷連線時發佈的 MQTT Will 訊息會產生簡訊費用，如 AWS IoT Core 定價：簡訊中所述。

如需簡訊成本的詳細資訊，請參閱 AWS IoT Core 定價：簡訊。

比較 MQTT 保留訊息和 MQTT 持久性工作階段

保留訊息和持久性工作階段是 MQTT 的標準功能，可讓裝置接收在離線時發佈的訊息。可以透過持久性工作階段發佈的保留訊息。本節說明這些功能的主要特色及其運作方式。

	保留訊息	持久性工作階段
主要功能	保留訊息可用於在連線後對大型裝置群組進行設定或通知。 若您希望裝置在重新連線後，僅接收發佈至主題之最後一則訊息時，即可使用保留訊息。	對於具有間歇性連線且可能會遺漏數則重要訊息的裝置，持久性工作階段能發揮重要作用。 裝置可以與持久性工作階段連線，以便接收在離線時傳送的訊息。
範例	保留訊息可以在裝置上線時提供與其環境有關的組態資訊。	透過行動網路連線且具有間歇性連線的裝置，可能會使用持

	保留訊息	持久性工作階段
	初始組態可能包含其應訂閱的其他訊息主題清單，或是應該如何設定其本機時區的相關資訊。	久性工作階段，以避免在裝置超出網路覆蓋範圍或需要關閉行動無線電時遺失所傳送的重要訊息。
在主題的初始訂閱時收到的訊息	訂閱含有保留訊息的主題後，就會收到最近保留訊息。	若訂閱的主題沒有保留訊息，在有發佈至該主題的訊息前，不會收到任何訊息。
重新連線後的訂閱主題	如果沒有持久性工作階段，用戶端必須在重新連線後訂閱主題。	已訂閱的主題會在重新連線後還原。
重新連線後收到的訊息	訂閱含有保留訊息的主題後，就會收到最近保留訊息。	在裝置中斷連線時，以 QOS = 1 發佈且以 QOS =1 訂閱的所有訊息都會在裝置重新連線後傳送。
資料/工作階段過期	在 MQTT 3 中的訊息不會過期。這些訊息在遭取代或刪除前都會存放起來。在 MQTT 5 中，保留的訊息會在您設定的訊息過期間隔後過期。如需詳細資訊，請參閱訊息過期。	如果用戶端未在逾時期間內重新連線，則持久性工作階段會過期。持久性工作階段過期後，在裝置中斷連線時，用戶端的訂閱和以 QOS = 1 發佈並以 QOS =1 訂閱的已儲存訊息都會遭到刪除。過期的訊息將不會傳遞。如需持久性工作階段之工作階段過期的詳細資訊，請參閱 the section called “MQTT 持久性工作階段”。

如需持久性工作階段的相關資訊，請參閱 the section called “MQTT 持久性工作階段”。

發佈用戶端會透過「保留訊息」判斷是否要在連線後保留訊息並將其傳遞至裝置，以及先前是否有工作階段。儲存訊息的選擇是由發佈者做出，而儲存的訊息會交付給訂閱 QoS 0 或 QoS 1 訂閱的所有目前和未來用戶端。保留訊息一次只會保留一則指定主題的相關訊息。

帳戶存放的保留訊息數量達到上限時，在系統刪除部分保留訊息且保留訊息計數低於上限前， AWS IoT Core 會針對已設定 RETAIN 且承載大於 0 個位元組的發佈訊息傳回調節回應。

MQTT 保留的訊息和 AWS IoT 裝置影子

保留訊息和 Device Shadow 都會保留裝置中的資料，但它們的行為不同，用途也不同。本節說明其異同。

	保留訊息	Device Shadow
訊息承載具有預先定義的結構或結構描述	如實作所定義。MQTT 不會為其訊息承載指定結構或結構描述。	AWS IoT 支援特定資料結構。
更新訊息承載會產生事件訊息	發佈保留訊息會將訊息傳送至訂閱的用戶端，但不會產生其他更新訊息。	更新 Device Shadow 會產生更新訊息，其中會描述變更。
已將訊息更新進行編號	不會自動將保留訊息進行編號。	Device Shadow 文件具有自動版本號碼和時間戳記。
訊息承載會連接至物件資源	保留訊息不會連接至物件資源。	Device Shadow 會連接至物件資源。
更新訊息承載的個別元素	如果不更新整個訊息承載，則無法變更訊息的個別元素。	您可以更新 Device Shadow 文件的個別元素，無需更新整個 Device Shadow 文件。
用戶端在訂閱時收到訊息資料	用戶端在訂閱具有保留訊息的主題後，會自動收到保留訊息。	用戶端可以訂閱 Device Shadow 更新，但必須特意請求目前狀態。
索引和可搜尋性	不會為保留訊息建立可供搜尋的索引。	機群索引製作會建立 Device Shadow 資料的索引，以供搜尋和彙總。

MQTT Last Will and Testament (LWT) 訊息

Last Will and Testament (LWT) 是 MQTT 的功能。用戶端可使用 LWT 指定一則訊息，代理程式將在發生意外中斷連線的情況時，將該訊息發佈至用戶端定義的主題，並傳送至訂閱該主題的所有用戶端。用戶端指定的訊息稱為 LWT 訊息或 Will 訊息，而用戶端定義的主題稱為「Will 主題」。您可以在裝置連線至代理程式時指定 LWT 訊息。這些訊息可藉由在連線期間，於 `Connect Flag bits` 欄位中設定 `Will Retain` 旗標的方式加以保留。例如，如果 `Will Retain` 旗標設定為 `1`，Will 訊息將會儲存在代理程式中相關聯的 Will 主題中。如需詳細資訊，請參閱 Will 訊息。

管理用戶端連線時，您可以控制在中斷用戶端連線時是否發佈 LWT 訊息。如需詳細資訊，請參閱the section called “MQTT 連線管理”。

代理程式會儲存 Will 訊息，直到發生意外中斷連線。發生這種情況時，代理程式會將訊息發佈到訂閱 Will 主題的所有用戶端，以通知發生中斷連線。如果用戶端使用 MQTT DISCONNECT 訊息透過用戶端起始的中斷連線從代理程式中斷連線，則代理程式將不會發佈儲存的 LWT 訊息。在所有其他情況下，都會送出 LWT 訊息。由於中斷連線處理的非同步性質，LWT 訊息不保證在重新連線期間會依序分派。我們建議您使用生命週期事件來改善連線狀態偵測的準確性，因為這些事件會提供時間戳記和版本編號等屬性來管理out-of-order的事件。如需代理程式將傳送 LWT 訊息的完整中斷連線案例清單，請參閱連線/中斷連線事件。

使用 connectAttributes

`ConnectAttributes` 可讓您在 IAM 政策中指定要在連線訊息中使用的屬性，例如 `PersistentConnect` 和 `LastWill`。透過 `ConnectAttributes`，您可以建置預設不允許裝置存取新功能的政策，如果裝置遭到入侵，這會很有幫助。

`connectAttributes` 支援下列功能：

`PersistentConnect`

當用戶端和代理程式之間的連線中斷時，使用 `PersistentConnect` 功能可儲存用戶端在連線期間進行的所有訂閱。

`LastWill`

當用戶端意外中斷連線時，使用 `LastWill` 功能可將訊息發佈至 `LastWillTopic`。

根據預設，您的政策具有非持久性連線，而且沒有傳遞此連線的屬性。如果您想要有持久性連線，必須在 IAM 政策中指定持久性連線。

透過主控台或 APIs 管理用戶端連線時，您可以檢視連線用戶端的連線屬性和工作階段組態。如需詳細資訊，請參閱the section called “MQTT 連線管理”。

如需 ConnectAttributes 範例，請參閱連線政策範例。

MQTT 5 支援的功能

AWS IoT Core MQTT 5 的支援是以 MQTT v5.0 規格為基礎，但有一些差異，如 中所述the section called “AWS IoT 與 MQTT 規格的差異”。

AWS IoT Core 支援下列 MQTT 5 功能：

- 共用訂閱
- 全新啟動和工作階段過期
- 所有 ACKs上的原因碼
- 主題別名
- 訊息到期
- 其他 MQTT 5 功能

共用訂閱

AWS IoT Core 同時支援 MQTT 3.1.1 和 MQTT 5 的共用訂閱。共用訂閱允許多個用戶端共用主題的訂閱，只有一個用戶端會使用隨機分佈接收發佈至該主題的訊息。共用訂閱可以有效地在多個訂閱者之間平衡 MQTT 訊息。例如，假設您有 1,000 部裝置發佈至相同主題，而 10 個後端應用程式則處理這些訊息。在這種情況下，後端應用程式可以訂閱相同的主題，每個應用程式都會隨機接收裝置發佈到共用主題的訊息。這實際上是「共享」這些訊息的負載。共用訂閱也允許更好的彈性。當任何後端應用程式中斷連線時，代理程式會將負載分配給群組中剩餘的訂閱用戶。當所有訂閱者中斷連線時，訊息會排入佇列。

MQTT 3.1.1 和 MQTT 5 連線上的共用訂閱皆可使用訊息佇列功能，以增強訊息傳遞可靠性。

若要使用共用訂閱，用戶端會訂閱共用訂閱的主題篩選條件，如下所示：

```
$share/{ShareName}/{TopicFilter}
```

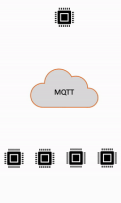
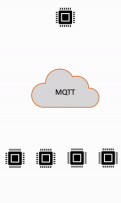
- $share 是個常值字串，表示共享訂閱的主題篩選條件，必須以 $share 開頭。
- {ShareName} 是個字元字串，用來指定一組訂閱用戶使用的共享名稱。共用訂閱的主題篩選條件必須包含 ShareName，後面接著 / 字元。{ShareName} 不得包含下列字元：/、+、或 #。的大小上限為 {ShareName} 128 個 UTF-8 字元。

- {TopicFilter} 遵循與非共用訂閱相同的主題篩選條件語法。的大小上限為 {TopicFilter} 256 個 UTF-8 字元。
- $share/{ShareName}/{TopicFilter} 所需的兩個斜線 (/) 不包含於主題和主題篩選條件中最大斜線數目限制中。

具有相同 的訂閱{ShareName}/{TopicFilter}屬於相同的共用訂閱群組。您可以建立多個共用訂閱群組，並且不超過每個群組的共用訂閱限制。如需詳細資訊，請參閱《AWS 一般參考》中的 AWS IoT Core 端點和配額。

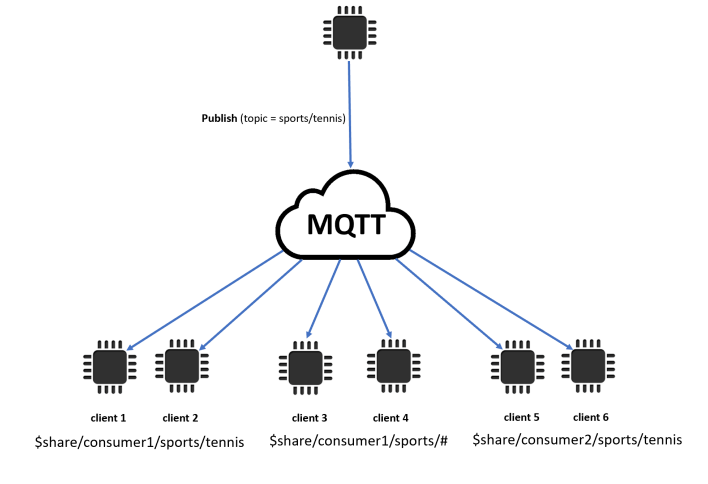
下表比較非共用訂閱和共用訂閱：

訂閱	說明	主題篩選條件範例
非共用訂閱	每個用戶端都會建立個別訂閱以接收已發佈的訊息。將訊息發佈至某個主題時，該主題的所有訂閱用戶都會收到該訊息的副本。	`sports/tennis` `sports/#`
共用訂閱	多個用戶端可共享一個主題的訂閱，且只有一個用戶端會使用隨機分佈接收發佈至該主題的訊息。	`$share/consumer/sports/tennis` `$share/consumer/sports/#`

非共用訂閱流程	共用訂閱流程		
			

使用共用訂閱的重要注意事項

- 如果共用訂閱者群組包含任何持久性工作階段訂閱者、當共用群組中的所有訂閱者中斷連線，或有任何訂閱者違反每個連線限制每秒發佈請求，則任何未確認的 QoS 1 訊息和發佈至共用訂閱群組的未交付 QoS 1 訊息都會排入佇列。如需詳細資訊，請參閱共用訂閱訊息佇列。
- 發佈至共用訂閱群組的 QoS 0 訊息將在任何失敗時捨棄。
- 作為共用訂閱者群組的一部分訂閱主題模式時，共用訂閱不會收到保留的訊息。在具有共用訂閱者且具有RETAIN旗標集的主題上發佈的訊息會交付給共用訂閱者，就像任何其他發佈訊息一樣。
- 當共用訂閱包含萬用字元 (# 或 +) 時，主題可能有多個相符的共用訂閱。如果發生這種情況，訊息代理程式會複製發佈訊息，並將其傳送至每個相符共用訂閱中的隨機用戶端。共用訂閱的萬用字元行為可在下圖中說明。



在此範例中，有三個與發佈 MQTT 主題 相符的共用訂閱sports/tennis。訊息代理程式會複製已發佈的訊息，並將該訊息傳送至每個相符群組中的隨機用戶端。

用戶端 1 和用戶端 2 共享訂閱：$share/consumer1/sports/tennis

用戶端 3 和用戶端 4 共享訂閱：$share/consumer1/sports/#

用戶端 5 和用戶端 6 共享訂閱：$share/consumer2/sports/tennis

如需共用訂閱限制的詳細資訊，請參閱 AWS 一般參考中的AWS IoT Core 端點和配額。若要在AWS IoT 主控台中使用 AWS IoT MQTT 用戶端測試共用訂閱，請參閱 ???。您也可以使用用戶端連線管理功能，檢視已連線用戶端訂閱的主題，包括共用訂閱。如需詳細資訊，請參閱the section called “MQTT 連線管理”。如需共用訂閱的詳細資訊，請參閱 MQTTv5.0 規格中的共用訂閱。

共用訂閱訊息佇列

為了增強訊息傳遞可靠性，共用訂閱包含訊息佇列功能，可在沒有線上訂閱者可用時儲存訊息。當共用訂閱群組包含至少一個具有持久性工作階段的成員時，該群組會啟用佇列功能。分發訊息時，會選取線上成員做為收件人。當線上找不到成員或訂閱者超過Publish requests per second per connection限制時，QoS 1 訊息會排入佇列。佇列訊息會在現有成員繼續其持久性工作階段，或新成員加入群組時傳送。佇列訊息會以每個作用中群組訂閱者每秒最多 20 個佇列訊息的方式傳遞，以及根據訂閱交付給訂閱者的任何其他訊息。

根據預設，排入佇列的訊息保留會遵循Persistent Session expiry period配額。不過，如果傳入發佈訊息中設定了訊息過期間隔 (MEI)，則以 MEI 為優先。當 MEI 存在時，它會決定訊息保留期間，無論持久性工作階段到期期間為何。

訊息佇列速率會根據Queued messages per second per account配額來限制，而訊息數量則受限於Maximum number of queued messages per shared subscription group配額。若要檢視和管理配額，請存取 Service Quotas 主控台。

您可以在AWS/Usage命名空間ApproximateQueueDepth下搜尋 來監控佇列 CloudWatch，也可以使用下列 CLI 命令列出與每個共用訂閱群組的佇列深度相關聯的指標。

```
aws cloudwatch list-metrics --namespace "AWS/Usage" --dimensions
 Name=Resource,Value='ApproximateQueueDepth'
```

超過這些限制時，只會保留在達到限制之前已排入佇列的訊息。會捨棄超過限制的新傳入訊息。系統不會將較舊的佇列訊息取代為較新的訊息。

訊息的佇列會記錄在 CloudWatch 指標和 CloudWatch Logs 中。如需有關將項目寫入 CloudWatch 和 CloudWatch Logs 的資訊，請參閱 訊息代理程式指標 和 佇列日誌項目。排入佇列的訊息仍會以標準簡訊費率計費。如需訊息定價的詳細資訊，請參閱 AWS IoT Core 定價。

共用訂閱群組中的工作階段生命週期

當乾淨的工作階段訂閱群組時，它會成為群組的線上成員。當它取消訂閱或中斷連線時，清除工作階段會離開 群組。

當持久性工作階段訂閱群組時，它會成為群組的線上成員。中斷連線時，仍會保留在群組中，但會成為群組的離線成員。重新連線時，它會再次成為線上成員。持久性工作階段會在群組明確取消訂閱或在中斷連線後過期時離開群組。

全新啟動和工作階段過期

您可以使用「全新啟動」和「工作階段過期」，以更具彈性的方式處理持久性工作階段。「全新啟動」旗標指出工作階段是否應在不使用現有工作階段的情況下啟動。「工作階段過期」間隔指出中斷連線後保留工作階段的時間長度。可在中斷連線時修改工作階段過期間隔。如需詳細資訊，請參閱the section called “MQTT 持久性工作階段”。

所有 ACKs上的原因碼

您可以使用原因代碼更輕鬆地偵錯或處理錯誤訊息。訊息代理程式會根據與代理程式的互動類型 (「訂閱」、「發佈」、「確認」) 傳回原因代碼。如需更多詳細資訊，請參閱 MQTT 原因代碼。如需 MQTT 原因代碼的完整清單，請參閱 MQTT v5 規格。

主題別名

主題別名是一種雙位元組整數，可用來取代主題名稱。使用主題別名可以最佳化主題名稱的傳輸，以減少計量資料服務的資料成本。預設限制 AWS IoT Core 為 8 個主題別名。如需詳細資訊，請參閱《AWS 一般參考》中的 AWS IoT Core 端點和配額。

訊息到期

您可以對已發佈的訊息新增訊息過期值。這些值代表訊息過期間隔 (MEI)，以秒為單位。如果訊息尚未在該間隔內傳送給訂閱者，則訊息將會過期並予以移除。如果未設定訊息過期值，訊息就不會過期。

在傳出流量中，訂閱者會收到一條訊息，告知過期間隔的剩餘時間。例如，如果傳入發佈訊息的訊息過期時間為 30 秒，且在 20 秒後路由給訂閱者，則訊息過期欄位會更新為 10。訂閱者收到的訊息可能具有值為 0 的更新 MEI。這是因為在剩餘時間達到 999 毫秒以下，它將更新為 0。

在 中 AWS IoT Core，最小 MEI 為 1。如果從用戶端將間隔設定為 0，則系統會將其調整為 1。訊息過期間隔上限為 604800 (7 天)。任何較高的值都將調整為最大值。

在跨版本通訊中，訊息過期行為取決於傳入發佈訊息的 MQTT 版本。例如，若某個訊息的訊息過期時間是由透過 MQTT5 連線的工作階段所傳送，則其可能會在使用 MQTT3 工作階段訂閱的裝置上過期。下表列出訊息過期功能如何支援下列類型的發佈訊息：

發佈訊息類型	訊息過期間隔
常規發佈	如果伺服器無法在指定的時間內傳遞訊息，則過期的訊息將遭移除，而訂閱者將不會收到這些訊息。這包括裝置未發佈其 QoS 1 訊息等情況。

發佈訊息類型	訊息過期間隔
保留	如果保留的訊息到期，而新用戶端訂閱了該主題，則該用戶端將不會在訂閱時收到訊息。
遺言	遺言訊息的間隔會在用戶端中斷連線之後啟動，而伺服器會嘗試將遺言訊息傳遞給其訂閱者。
佇列訊息	如果具有訊息過期間隔的傳出 QoS 1 在用戶端離線時過期，則在持久性工作階段繼續後，用戶端將不會收到過期的訊息。

其他 MQTT 5 功能

伺服器中斷連線

中斷連線時，伺服器可以主動向用戶端傳送 DISCONNECT 以通知連線關閉，並提供中斷連線的原因代碼。

請求/回應

發佈者可以請求接收者在接收時將回應傳送至發佈者指定的主題。

封包大小上限

用戶端和伺服器可以分別指定其支援的最大封包大小。

承載格式和內容類型

您可以在發佈訊息時指定承載格式 (二進位、文字) 和內容類型。這些會轉發給訊息接收者。

MQTT 5 屬性

MQTT 5 屬性是 MQTT 標準的重要新增項目，旨在支援新的 MQTT 5 功能，例如工作階段過期和請求/回應模式。在 中 AWS IoT Core，您可以建立規則來轉送傳出訊息中的屬性，或使用 HTTP 發佈來發佈具有一些新屬性的 MQTT 訊息。

下表列出 AWS IoT Core 支援的所有 MQTT 5 屬性。

屬性	說明	輸入類型	封包
承載格式指示器	布林值，指出承載的格式是否為 UTF-8。	位元組	PUBLISH、CONNECT
內容類型	描述承載內容的 UTF-8 字串。	UTF-8 字串。	PUBLISH、CONNECT
回應主題	此 UTF-8 字串描述了要在請求-回應流程中作為接收者發佈目標的主題。主題不得包含萬用字元。	UTF-8 字串。	PUBLISH、CONNECT
相互關聯資料	請求訊息的傳送者使用的二進位資料，用以識別回應訊息所對應的請求。	二進位	PUBLISH、CONNECT
使用者屬性	一組 UTF-8 字串對。這個屬性可能在單一封包中出現多次。接收者將以與傳送索引鍵值對相同的順序來接收索引鍵。	UTF-8 字串對	CONNECT、PUBLISH、遺言屬性、SUBSCRIBE、DISCONNECT、UNSUBSCRIBE
訊息過期間隔時間	代表訊息過期間隔的 4 位元組整數，以秒為單位。如果不存在，則訊息不會到期。	4 位元組整數	PUBLISH、CONNECT
工作階段過期間隔時間	代表工作階段到期間隔的 4 位元組整數，以秒為單位。 AWS IoT Core 支援最多 7 天，預設最多 1 小時。如果您設定的值超過帳戶的最大值， AWS IoT Core 會在 CONNACK 中傳回調整後的值。	4 位元組整數	CONNECT、CONNACK、DISCONNECT
已指派的用戶端識別碼	裝置未指定用戶端 ID AWS IoT Core 時，由產生的隨機用戶端 ID。隨機用戶端 ID 必須是新的用戶端識別碼，且目前由代理程式管理的任何其他工作階段都未予以使用。	UTF-8 字串。	CONNACK

屬性	說明	輸入類型	封包
伺服器保持連線	2 位元組整數，代表伺服器指派的保持連線時間。如果用戶端離線的時間超過保持連線時間，伺服器將中斷用戶端的連線。	2 位元組整數	CONNACK
請求問題資訊	布林值，指出在失敗時是否傳送原因字串或使用者屬性。	位元組	CONNECT
接收上限	2 位元組整數，代表無需接收 PUBACK 即可傳送的 PUBLISH QOS > 0 封包數目上限。	2 位元組整數	CONNECT、CONNACK
主題別名上限	此值表示將接受為主題別名的最高值。預設值為 0。	2 位元組整數	CONNECT、CONNACK
QoS 上限	AWS IoT Core 支援的 QoS 最大值。預設為 1。 AWS IoT Core 不支援 QoS 2。	位元組	CONNACK
保留可用	布林值，指出 AWS IoT Core 訊息中介裝置是否支援保留的訊息。預設為 1。	位元組	CONNACK
封包大小上限	AWS IoT Core 接受和傳送的最大封包大小。不能超過 128 KB。	4 位元組整數	CONNECT、CONNACK
可用的萬用字元訂閱	布林值，指出 AWS IoT Core 訊息中介裝置是否支援可用的萬用字元訂閱。預設為 1。	位元組	CONNACK
可用的訂閱識別碼	布林值，指出 AWS IoT Core 訊息中介裝置是否支援可用的訂閱識別符。預設值為 0。	位元組	CONNACK

MQTT 原因代碼

MQTT 5 引進改善錯誤報告與原因碼回應。 AWS IoT Core 可以傳回原因碼，包括但不限於下列依封包分組。如需 MQTT 5 支援原因代碼的完整清單，請參閱 MQTT v5 規格。

CONNACK 原因代碼

Value	Hex	原因代碼名稱	說明
0	0x00	成功	已接受連線。
128	0x80	未指定的錯誤	伺服器不願揭露失敗原因，或者沒有其他適用的原因代碼。
133	0x85	用戶端識別碼無效	用戶端識別碼是有效字串，但未獲伺服器允許。
134	0x86	使用者名稱或密碼錯誤	伺服器不接受用戶端指定的使用者名稱或密碼。
135	0x87	未獲授權	用戶端未獲得連線的授權。
144	0x90	主題名稱無效	遺言主題名稱已正確產生，但未獲伺服器接受。
151	0x97	超過配額	已超出實作或管理施加的限制。
155	0x9B	不支援 QoS。	伺服器不支援遺言 QoS 中的 QoS 集。

PUBACK 原因代碼

Value	Hex	原因代碼名稱	說明
0	0x00	成功	已接受訊息。QoS 1 訊息的發佈作業繼續進行。
128	0x80	未指定的錯誤	接收者不接受發佈內容，但不願揭露原因，或不符合其中一個其他值。
135	0x87	未獲授權	PUBLISH 未經授權。
144	0x90	主題名稱無效	主題名稱格式不正確，但未獲用戶端或伺服器接受。
145	0x91	使用中的封包識別碼	封包識別碼已在使用中。這可能表示用戶端與伺服器之間的工作階段狀態不相符。
151	0x97	超過配額	已超出實作或管理施加的限制。

DISCONNECT 原因代碼

Value	Hex	原因代碼名稱	說明
129	0x81	格式錯誤的封包	接收到的封包不符合此規格。
130	0x82	協定錯誤	收到非預期或亂序的封包。
135	0x87	未獲授權	請求未獲得授權。
139	0x8B	伺服器關閉	伺服器正在關閉。
141	8D	保持連線逾時	連線已關閉，因為未接收任何封包的時間已達到保持連線時間的 1.5 倍。
142	0x8E	已接管工作階段	系統已連接到使用相同 ClientID 的另一個連線，導致此連線關閉。
143	0x8F	主題篩選條件無效	意願主題名稱格式正確，但未獲伺服器接受。
144	0x90	主題名稱無效	主題名稱格式正確，但未獲此用戶端或伺服器接受。
147	0x93	超過接收上限	用戶端或伺服器已收到超過其未傳送 PUBACK 或 PUBCOMP 的發佈項目接收上限。
148	0x94	主題別名無效	用戶端或伺服器已收到 PUBLISH 封包，其中包含的主題別名大於 CONNECT 或 CONNACK 封包中傳送的主題別名上限。
151	0x97	超過配額	已超出實作或管理施加的限制。
152	0x98	管理動作	由於系統執行管理動作，導致連線關閉。
155	0x9B	不支援 QoS。	用戶端指定的 QoS 大於 CONNACK 中 QoS 上限指定的 QoS。
161	0xA1	不支援訂閱識別碼	伺服器不支援訂閱識別碼；未接受訂閱。

PUBACK 原因代碼

Value	Hex	原因代碼名稱	說明
0	0x00	授予 QoS 0	已接受訂閱，且傳送的 QoS 上限為 QoS 0。這可能是低於請求的 QoS。
1	0x01	授予 QoS 1	已接受訂閱，且傳送的 QoS 上限為 QoS 1。這可能是低於請求的 QoS。
128	0x80	未指定的錯誤	未接受訂閱，伺服器不願揭露原因，或者其他原因代碼均不適用。
135	0x87	未獲授權	用戶端未獲授權進行此訂閱。
143	0x8F	主題篩選條件無效	主題篩選器的格式正確，但不允許此用戶端使用。
145	0x91	使用中的封包識別碼	指定的封包識別碼已在使用中。
151	0x97	超過配額	已超出實作或管理施加的限制。

UNSUBACK 原因代碼

Value	Hex	原因代碼名稱	說明
0	0x00	成功	訂閱已刪除。
128	0x80	未指定的錯誤	取消訂閱無法完成，伺服器不願揭露原因，或者其他原因代碼均不適用。
143	0x8F	主題篩選條件無效	主題篩選器的格式正確，但不允許此用戶端使用。
145	0x91	使用中的封包識別碼	指定的封包識別碼已在使用中。

AWS IoT 與 MQTT 規格的差異

雖然訊息代理程式實作以 MQTT v3.1.1 規格與 MQTT 5.0 規格為基礎，但有些部分與規格有異，內容如下：

- AWS IoT 不支援 MQTT 3 的下列封包：PUBREC、PUBREL 和 PUBCOMP。
- AWS IoT 不支援 MQTT 5 的下列封包：PUBREC、PUBREL、PUBCOMP 和 AUTH。
- AWS IoT 不支援 MQTT 5 伺服器重新導向。
- AWS IoT 僅支援 MQTT 服務品質 (QoS) 層級 0 和 1。 AWS IoT 不支援發佈或訂閱 QoS 層級 2。請求 QoS 2 時，訊息代理程式不會傳送 PUBACK 或 SUBACK。
- 在 中 AWS IoT，訂閱 QoS 層級 0 的主題表示訊息傳遞零次或多次。一則訊息可能會傳送超過一次。傳送超過一次的訊息在發送時可能會使用不同的封包 ID。在這些情況下，DUP 旗標就不會設置。
- 在回應連線請求時，訊息代理程式會傳送 CONNACK 訊息。此訊息包含了一個旗標，指出連線是否會恢復先前的工作階段。
- 在傳送額外的控制封包或中斷連線請求之前，用戶端必須等待其裝置從 AWS IoT 訊息代理程式接收 CONNACK 訊息。
- 當用戶端訂閱主題，訊息代理程式傳送 SUBACK 以及用戶端開始接收符合的新訊息這段期間可能會發生延遲。
- 當用戶端在主題篩選條件中使用萬用字元 # 以訂閱主題時，主題階層中位於及低於其層級的所有字串都會相符。不過，父系主題不相符。例如，訂閱主題 `sensor/#` 會收到向主題 `sensor/`、`sensor/temperature`、`sensor/temperature/room1` 發佈的訊息，但不會收到向 `sensor` 發佈的訊息。如需萬用字元的詳細使用資訊，請參閱 主題名稱篩選條件。
- 訊息代理程式使用用戶端 ID 來識別每一個用戶端。用戶端 ID 作為 MQTT 承載的一部分從用戶端傳遞至訊息代理程式。具有相同用戶端 ID 的兩個用戶端，不能同時連線至訊息代理程式。當用戶端使用另一個用戶端正在使用的用戶端 ID 連線至訊息代理程式，系統會接受新的用戶端連線，而先前連線的用戶端會中斷連線。您也可以使用主控台或 APIs手動中斷用戶端連線。如需詳細資訊，請參閱the section called “MQTT 連線管理”。
- 在極少數情況下，訊息代理程式可能會以不同的封包 ID 重新傳送相同的邏輯 PUBLISH 訊息。
- 包含萬用字元的主題篩選條件訂閱無法接收保留訊息。若要接收保留訊息，訂閱請求必須包含與保留訊息主題完全相符的主題篩選條件。
- 訊息代理程式不保證接收訊息及 ACK 的順序。
- AWS IoT 可以具有與規格不同的限制。如需詳細資訊，請參閱AWS IoT 參考指南中的 AWS IoT Core 訊息代理程式和通訊協定限制和配額。

- 不支援 MQTT DUP 旗標。

MQTT 連線管理

AWS IoT Core 提供 APIs和主控台功能，協助您管理 MQTT 連線，包括中斷用戶端連線、檢視連線詳細資訊和管理其工作階段的功能。這些功能可讓您進一步控制 AWS IoT 用戶端機群，並協助疑難排解連線問題。

本節內容

- the section called “中斷連接 MQTT 用戶端”
- the section called “檢視連線詳細資訊”
- the section called “管理用戶端訂閱”
- the section called “CloudWatch Logs AWS IoT 日誌項目”

中斷連接 MQTT 用戶端

使用 `DeleteConnection` API AWS IoT Core 透過指定 MQTT 裝置用戶端 IDs中斷連線。當您中斷用戶端連線時， 會 AWS IoT Core 中斷用戶端與 AWS IoT Core 訊息代理程式的連線，並可選擇清除工作階段狀態，並隱藏最後遺囑和試驗 (LWT) 訊息。

當您呼叫 `DeleteConnection` API 時， AWS IoT Core 會執行數個動作以確保乾淨的中斷連線。AWS IoT Core 會先傳送 MQTT 中斷連線訊息給用戶端，以終止 MQTT 工作階段。然後，服務會關閉基礎 TCP/TLS 通訊端。

訊息代理程式會將`DISCONNECT`封包傳送至裝置，並發佈生命週期事件，其中包含中斷連線原因 `API_INITIATED_DISCONNECT`。這可協助您識別何時透過 API 啟動中斷連線，而不是由用戶端啟動，或是因為網路問題。您可以監控這些事件，以實現可見性、疑難排解和稽核目的。例如，您可以使用 AWS IoT 規則來處理這些事件，以追蹤用戶端中斷連線的時間和原因。

如果您將 `cleanSession` 參數設定為 `true`， 會 AWS IoT Core 移除用戶端的工作階段狀態，包括所有訂閱和排入佇列的訊息。如果您清除工作階段，持久性工作階段會終止。如果用戶端是持久性工作階段，且 `preventWillMessage` 參數設定為 `false`，則服務會在可用時分派 LWT 訊息，這在計劃的維護操作期間很有用。

當您呼叫 `DeleteConnection` API 時，中斷連線程序會立即開始，但用戶端辨識中斷連線的確切時間可能會因網路條件和用戶端實作而有所不同。大多數用戶端會在幾秒鐘內偵測到中斷連線，但在某些情況下，網路連線能力不佳時，用戶端可能需要更長的時間才能辨識中斷連線。

> **Note**
>
> 透過強制中斷連線，用戶端必須重新驗證和重新授權新的工作階段狀態。API 呼叫本身不會阻止用戶端重新連線。如果需要，則在發出 DeleteConnection API 呼叫之前，還必須修改用戶端的登入資料或政策。

如需定價的詳細資訊，請參閱 AWS IoT Core 定價。

使用案例

DeleteConnection API 有助於管理行為有問題或消耗過多資源的用戶端。透過強制中斷連線，您可以確保用戶端使用適當的身分驗證和授權重新建立其連線，這有助於解決資源消耗問題。

用戶端重新導向案例也受益於此 API。當您需要將用戶端重新導向至不同的端點，或者，您可以透過程式設計方式中斷連線 AWS 區域，並透過變更 DNS 設定讓用戶端重新連線至不同的 AWS IoT Core 端點。此 API 可協助解決停滯的連線，或清除可能阻礙正常操作的有問題工作階段狀態。

API 參數

DeleteConnection API 接受下列參數：

clientId （必要）

要中斷連線之 MQTT 用戶端的唯一識別符。這是在 URL 路徑中指定的。用戶端 ID 不能以貨幣符號 ($) 開頭。

> **Note**
>
> MQTT IDs 可包含在 HTTP 請求中無效的字元。使用 DeleteConnection API 時，您必須對用戶端 ID 中的任何字元進行 URL 編碼（百分比編碼），這些字元在 MQTT 中有效，但在 HTTP 中無效。這包括特殊字元，例如空格、斜線 (/) 和 UTF-8 字元。例如，空格變成 %20，正斜線變成 %2F，而 UTF-8 字元 ü 變成 %C3%BC。適當的編碼可確保 MQTT 用戶端 IDs HTTP 型 API 呼叫中正確傳輸。

cleanSession （選用）

指定是否在中斷連線時移除用戶端的工作階段狀態。設定為 true 以刪除所有工作階段資訊，包括訂閱和排入佇列的訊息。設定為 false 以保留工作階段狀態。根據預設，這會設定為 false（保留工作階段狀態）。對於清除工作階段，會忽略此參數。

preventWillMessage （選用）

如果中斷連線時可用，控制 是否 AWS IoT Core 分派 Last Will and Testament (LWT) 訊息。設定為 `true` 以防止分派 LWT 訊息。設定為 `false`以允許分派。根據預設，這會設定為 `false` (如果可用，則分派 LWT)。

API 語法

`DeleteConnection` API 使用以下 HTTP 請求格式：

```
DELETE /connections/<clientId>?
cleanSession=<cleanSession>&preventWillMessage=<preventWillMessage> HTTP/1.1
```

請求範例：

```
// Basic disconnect (preserves session, allows LWT message)
DELETE /connections/myDevice123 HTTP/1.1

// Disconnect and clear session
DELETE /connections/myDevice123?cleanSession=TRUE HTTP/1.1

// Disconnect, clear session, and prevent LWT message
DELETE /connections/myDevice123?cleanSession=TRUE&preventWillMessage=TRUE HTTP/1.1
```

成功的請求會傳回 HTTP 200 OK 且無回應內文。

> **Note**
>
> Signature 第 AWS 4 版用來簽署請求的服務名稱為：iotdevicegateway。您可以使用 `aws iot describe-endpoint --endpoint-type iot:Data-ATS`命令來尋找您的端點。

所需的許可

若要使用 `DeleteConnection` API，您需要下列 IAM 許可：

```
iot:DeleteConnection
```

您可以使用資源型政策，將此許可範圍限定為特定用戶端 IDs。例如：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
}
```

重要考量

中斷連線的用戶端可以立即嘗試重新連線，除非您已實作其他邏輯來防止重新連線。中斷連線操作只會終止目前的連線，而且不會阻止連線重新連線。如果您需要防止重新連線，請考慮實作用戶端邏輯或停用裝置的登入資料。

速率限制適用於 API 作為標準 AWS IoT Core API 速率限制的一部分。規劃大量中斷連線操作時，請確保您考慮這些限制，並實作適當的重試邏輯和批次策略，以避免限流。如需詳細資訊，請參閱 AWS IoT Core 端點和配額。

錯誤回應

`DeleteConnection` API 可以傳回下列錯誤回應：

InvalidRequestException

請求無效。如果用戶端 ID 格式無效、包含貨幣符號 ($) 字首，或缺少必要的參數，就會發生這種情況。

ResourceNotFoundException

指定的用戶端 ID 不存在或目前未連線，而且沒有持久性工作階段。

UnauthorizedException

未授權您執行此操作。確保您擁有必要的`iot:DeleteConnection`許可。

ForbiddenException

發起人無權提出請求。這可能是由於 IAM 許可不足或資源型政策限制所致。

ThrottlingException

請求頻率超過限制。降低 API 呼叫的頻率，並使用指數退避實作適當的重試邏輯。

InternalFailureException

出現未預期的錯誤。在短暫延遲後重試請求。

ServiceUnavailableException

此服務暫時無法使用。在短暫延遲後重試請求。

檢視連線詳細資訊

使用 `GetConnection` API 擷取作用中 MQTT 用戶端連線的詳細資訊，包括連線狀態、工作階段詳細資訊和網路資訊。

`GetConnection` API 會傳回完整的連線中繼資料，以協助疑難排解連線問題、監控用戶端行為和稽核連線模式。您可以使用此資訊來了解用戶端連線生命週期，並診斷 MQTT 工作階段的問題。無論用戶端是否使用乾淨或持久性工作階段，連線資訊會在中斷連線後提供 30 分鐘。對於超過 30 分鐘的中斷連線中繼資料，請使用需要啟用機群索引的 GetThingConnectivityData API。

使用案例

`GetConnection` API 適用於疑難排解連線問題和監控用戶端行為。您可以驗證用戶端連線狀態、檢查持久性工作階段設定，以及檢閱連線時間戳記，以了解用戶端連線模式。

網路診斷也受益於此 API。連線詳細資訊包括來源和目標 IP 地址和連接埠，可協助識別網路路由問題或連線端點問題。此資訊在偵錯連線問題或最佳化網路組態時非常有用。

API 參數

`GetConnection` API 接受下列參數：

clientId （必要）

要擷取連線詳細資訊之 MQTT 用戶端的唯一識別符。這是在 URL 路徑中指定的。用戶端 ID 不能以貨幣符號 ($) 開頭。

> **Note**
>
> MQTT IDs 可包含在 HTTP 請求中無效的字元。使用 `GetConnection` API 時，您必須對用戶端 ID 中的任何字元進行 URL 編碼（百分比編碼），這些字元在 MQTT 中有效，但在 HTTP 中無效。這包括特殊字元，例如空格、斜線 (/) 和 UTF-8 字元。例如，空格變成 %20，正斜線變成 %2F，而 UTF-8 字元 ü 變成 %C3%BC。適當的編碼可確保 MQTT 用戶端 IDs HTTP 型 API 呼叫中正確傳輸。

路由考量：當用戶端 IDs 同時包含正斜線和字串「訂閱」時，需要 URL 編碼。如果沒有適當的編碼，路徑結構可能會誤譯，您的請求將路由到 ListSubscriptions API，而不是 GetConnection。例如，用戶端 ID 用戶端/訂閱必須編碼為 client%2Fsubscriptions，以確保請求正確路由至 GetConnection，而不是解譯為ListSubscriptions呼叫。

includeSocketInformation （選用）

類型：布林值

預設：false

includeSocketInformation 參數控制通訊端層級網路資訊是否包含在 API 回應中。設定為時true，回應包含下列欄
位：sourceIp、sourcePort、targetIp、targetPort、vpcEndpointId。當 includeSocketInformation 未指定或設定為 時false，這些通訊端欄位會從回應中排除。如果 includeSocketInformation 參數設定為 ，true但發起人沒有檢視通訊端資訊的授權，則整個 API 呼叫會失敗，並顯示 403 禁止。

API 語法

GetConnection API 使用以下 HTTP 請求格式：

```
GET /connections/<client-id>?includeSocketInformation=false HTTP/1.1
```

請求範例：

```
GET /connections/myDevice123?includeSocketInformation=false HTTP/1.1

// Get connection details for a client with special characters (URL encoded)
// Original client ID: "my device/123"
GET /connections/my%20device%2F123?includeSocketInformation=true HTTP/1.1
```

連線用戶端的範例回應：

```
{
    "clientId": "abcd12345fvgbh",
    "connected": true,
    "cleanSession": false,
```

```
    "connectedSince": 1573002340451,
    "thingName": "deviceABC",
    "sourceIp": "203.0.113.1",
    "sourcePort": 8883,
    "targetIp": "198.51.100.1",
    "targetPort": 8883,
    "keepAliveDuration": 60,
    "disconnectReason": null,
    "sessionExpiry": 3600
}
```

回應欄位：

clientId

MQTT 用戶端的用戶端 ID。

connected

用戶端的連線狀態。`true` 如果用戶端目前連線 AWS IoT Core，或`false`如果用戶端未連線，則傳回 。

cleanSession

指出用戶端是否使用乾淨的工作階段。傳回`true`清除工作階段或`false`持久性工作階段。清除工作階段在中斷連線後不會保留狀態，而持久性工作階段會維護訂閱和排入佇列的訊息。

connectedSince

Unix 時間戳記（以毫秒為單位），指出用戶端連線的時間。僅在 `connected`為 時顯示`true`。

disconnectedSince

Unix 時間戳記（以毫秒為單位），指出用戶端何時中斷連線。僅在 `connected`為 時顯示`false`。

thingName

用戶端的物件名稱，只有在有與 clientId 相關聯的物件時才會顯示。

disconnectReason

如果用戶端目前中斷連線，最近中斷連線的原因。此欄位有助於識別中斷連線是用戶端啟動、伺服器啟動還是網路問題。僅在 `connected`為 時顯示`false`。如需中斷連線原因代碼，請參閱 LifeCycleEvents。

keepAliveDuration

用戶端在建立連線時指定的保持連線間隔，以秒為單位。這會決定用戶端傳送保持連線訊息以維持連線的頻率。

sourceIp

啟動連線之用戶端的 IP 地址。只有在 `includeSocketInformation` 設定為 `true`且您獲得擷取此資訊的授權時，才會傳回 。

sourcePort

用戶端用於連線的連接埠號碼。只有在 `includeSocketInformation` 設定為 `true`且您獲得擷取此資訊的授權時，才會傳回 。

targetIp

發出連線請求的 IP 地址。只有在 `includeSocketInformation` 設定為 `true`且您獲得擷取此資訊的授權時，才會傳回 。

targetPort

用戶端連線的 AWS IoT Core 端點連接埠號碼。只有在 `includeSocketInformation` 設定為 `true`且您獲得擷取此資訊的授權時，才會傳回 。

vpcEndpointId

VPC 端點的 ID。適用於 AWS IoT Core 透過 VPC 端點連線至 的用戶端。只有在 `includeSocketInformation` 設定為 `true`且您獲得擷取此資訊的授權時，才會傳回 。

sessionExpiry

用戶端在建立連線時所指定或預設的持久性工作階段到期組態。這會決定工作階段在用戶端中斷連線後保持作用中狀態的時間。

> **Note**
>
> Signature 第 AWS 4 版用來簽署請求的服務名稱為：iotdevicegateway。您可以使用 `aws iot describe-endpoint --endpoint-type iot:Data-ATS`命令來尋找您的端點。

所需的許可

若要使用 `GetConnection` API，您需要下列 IAM 許可：

```
iot:GetConnection
```

您可以使用資源型政策，將此許可範圍限定為特定用戶端 IDs。

> **Tip**
>
> 使用 `iot:IncludeSocketInformation`條件，根據組織的隱私權需求實作精細存取控制。

授權政策範例

範例 1：允許 GetConnection 提供所有用戶端的通訊端資訊

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/*"
        }
    ]
}
```

使用此政策，下列 API 呼叫會傳回連線詳細資訊，包括通訊端資訊：

```
GET /connections/myDevice123?includeSocketInformation=true HTTP/1.1
```

結果：API 會傳回所有連線欄位，包括 `sourceIp`、`targetIp`、 `sourcePort`和 `targetPort`。

範例 2：允許 GetConnection，但拒絕特定用戶端的通訊端資訊

您可以使用條件型方法或明確拒絕方法：

條件型方法：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*",
            "Condition": {
                "Bool": {
                    "iot:IncludeSocketInformation": "false"
                }
            }
        }
    ]
}
```

明確拒絕方法：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*",
            "Condition": {
                "Bool": {
                    "iot:IncludeSocketInformation": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": ["iot:GetConnection"],
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
}
```

使用任一政策時，會拒絕下列 API 呼叫：

```
GET /connections/myDevice123?includeSocketInformation=true HTTP/1.1
```

結果：API 傳回 403 禁止錯誤，因為這些用戶端的通訊端資訊存取受到限制。

錯誤回應

GetConnection API 可以傳回下列錯誤回應：

InvalidRequestException

請求無效。如果用戶端 ID 格式無效、包含貨幣符號 ($) 字首，或缺少必要的參數，就會發生這種情況。HTTP 狀態碼：400

ResourceNotFoundException

指定的用戶端 ID 不存在或目前未連線，而且沒有持久性工作階段。HTTP 狀態碼：404

ForbiddenException

發起人無權提出請求。這可能是由於 IAM 許可不足或資源型政策限制所致。HTTP 狀態碼：403

ThrottlingException

請求頻率超過限制。降低 API 呼叫的頻率，並使用指數退避實作適當的重試邏輯。HTTP 狀態碼：429

InternalFailureException

出現未預期的錯誤。在短暫延遲後重試請求。HTTP 狀態碼：500

管理用戶端訂閱

使用 `ListSubscriptions` API 擷取 MQTT 用戶端工作階段的所有主題訂閱。此 API 會針對連線用戶端和具有持久性工作階段的離線用戶端傳回訂閱，提供用戶端訂閱模式的可見性，並協助疑難排解訂閱相關問題。

`ListSubscriptions` API 提供完整的訂閱資訊，可協助稽核用戶端行為、疑難排解訊息傳遞問題，以及了解裝置機群的訂閱模式。您可以使用此資訊來驗證用戶端是否已訂閱預期的主題，並識別潛在的訂閱衝突。

使用案例

`ListSubscriptions` API 可協助您疑難排解訊息傳遞問題和稽核用戶端訂閱行為。使用此 API 來驗證用戶端訂閱的主題、驗證 QoS 層級，以及識別重疊的訂閱或過度寬鬆的名片。

訂閱管理案例也受益於此 API。當用戶端回報缺少訊息或意外訊息傳遞時，您可以使用此 API 來驗證其目前的訂閱，並確保符合預期的組態。這對於偵錯複雜的訂閱階層和共用訂閱組態特別有用。

API 參數

`ListSubscriptions` API 接受下列參數：

clientId （必要）

要擷取訂閱之 MQTT 用戶端的唯一識別符。這是在 URL 路徑中指定的。用戶端 ID 不能以貨幣符號 ($) 開頭。

> **Note**
>
> MQTT IDs 可包含在 HTTP 請求中無效的字元。使用 `ListSubscriptions` API 時，您必須對用戶端 ID 中的任何字元進行 URL 編碼（百分比編碼），這些字元在 MQTT 中有效，但在 HTTP 中無效。這包括特殊字元，例如空格、斜線 (/) 和 UTF-8 字元。例如，空格變成 %20，正斜線變成 %2F，而 UTF-8 字元 ü 變成 %C3%BC。適當的編碼可確保 MQTT 用戶端 IDs HTTP 型 API 呼叫中正確傳輸。
>
> 路由考量：當用戶端 IDs 同時包含正斜線和字串「訂閱」時，需要 URL 編碼。如果沒有適當的編碼，路徑結構可能會誤譯，您的請求將路由到 `ListSubscriptions` API，而不是 `GetConnection`。例如，用戶端 ID 用戶端/訂閱必須編碼為 client%2Fsubscriptions，以確保請求正確路由至 `GetConnection`，而不是解譯為`ListSubscriptions`呼叫。

nextToken （選用）

用於擷取下一組結果的分頁字符。使用此前一個回應中的字符，以繼續進行大型訂閱清單的分頁。

maxResults （選用）

在單一回應中傳回的訂閱數量上限。有效範圍為 1-200。預設值為 20。

API 語法

`ListSubscriptions` API 使用以下 HTTP 請求格式：

```
GET /connections/<client-id>/subscriptions?nextToken=token&maxResults=max HTTP/1.1
```

請求範例：

```
// Get subscriptions for a specific client
GET /connections/myDevice123/subscriptions HTTP/1.1

// Get subscriptions with custom page size
GET /connections/myDevice123/subscriptions?maxResults=50 HTTP/1.1
```

```
// Get next page of results using pagination token
GET /connections/myDevice123/subscriptions?nextToken=eyJhbGciOiJIUzI1NiJ9&maxResults=50
 HTTP/1.1
```

回應範例：

```
{
    "subscriptions": [
        {"topicFilter": "device/myDevice123/commands", "qos": 1},
        {"topicFilter": "device/myDevice123/shadow", "qos": 0}
    ],
    "nextToken": "eyJhbGciOiJIUzI1NiJ9"
}
```

回應欄位：

訂閱

訂閱物件陣列，每個都包含主題篩選條件和 QoS 層級。

topicFilter

用戶端訂閱的 MQTT 主題篩選條件。這可包含萬用字元 (+ 、#) 以進行模式比對。

qos

訂閱的服務品質等級 (0 或 1)。

nextToken

用於擷取其他結果的分頁字符。只有在要擷取更多訂閱時才會顯示。

> **Note**
>
> Signature 第 AWS 4 版用來簽署請求的服務名稱為：iotdevicegateway。您可以使用 `aws iot describe-endpoint --endpoint-type iot:Data-ATS`命令來尋找您的端點。

所需的許可

若要使用 `ListSubscriptions` API，您需要下列 IAM 許可：

```
iot:ListSubscriptions
```

您可以使用資源型政策，將此許可範圍限定為特定用戶端 IDs。例如：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:ListSubscriptions",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
}
```

錯誤回應

`ListSubscriptions` API 可以傳回下列錯誤回應：

InvalidRequestException

請求無效。HTTP 狀態碼：400

ResourceNotFoundException

指定的資源不存在。HTTP 狀態碼：404

ForbiddenException

發起人無權提出請求。HTTP 狀態碼：403

ThrottlingException

請求頻率超過限制。HTTP 狀態碼：429

InternalFailureException

出現未預期的錯誤。HTTP 狀態碼：500

使用主控台進行 MQTT 連線管理

您可以使用 AWS IoT 主控台中斷 MQTT 裝置的連線，並管理其連線。主控台提供直覺式界面，用於檢視連線裝置和執行中斷連線操作。

檢視連線的 MQTT 用戶端

此程序說明如何在 AWS IoT 主控台中檢視目前連線的 MQTT 裝置。

檢視連線的 MQTT 裝置

1. 開啟 AWS IoT 主控台。
2. 在左側導覽窗格中，選擇管理，然後選擇所有裝置。
3. 選擇用戶端連線以檢視目前連線的 MQTT 裝置清單。
4. 用戶端連線頁面會顯示每個連線裝置的完整資訊。您可以檢視做為 MQTT 裝置唯一識別符的用戶端 ID，以及顯示裝置目前是否連線的連線狀態。此頁面也會顯示連線自時間戳記，指出裝置何時建立其連線、裝置連線的來源 IP 地址，以及為裝置連線設定的保持運作間隔。

檢視 MQTT 用戶端訂閱

此程序說明如何使用 AWS IoT 主控台檢視特定 MQTT 用戶端的主題訂閱。

檢視用戶端訂閱

1. 開啟 AWS IoT 主控台。
2. 在左側導覽窗格中，選擇管理，然後選擇所有裝置。
3. 選擇用戶端連線以檢視 MQTT 裝置清單。
4. 使用搜尋功能搜尋您要檢視 訂閱的特定用戶端 ID。
5. 選取用戶端 ID 以檢視其詳細資訊頁面。
6. 在用戶端詳細資訊頁面中，找到訂閱的主題表格。此資料表會顯示用戶端目前訂閱的所有主題篩選條件，以及其對應的服務品質 (QoS) 層級。
7. 訂閱的主題表格會顯示：
   - 主題篩選條件：用戶端訂閱的 MQTT 主題模式，可能包含萬用字元 (+ 、#)
   - QoS：訂閱的服務品質等級 (0 或 1)

> **Note**
>
> 訂閱資訊適用於具有持久性工作階段的連線用戶端和離線用戶端。如果用戶端沒有作用中的訂閱，則資料表將為空白。

中斷連接 MQTT 裝置

此程序說明如何使用 AWS IoT 主控台中斷連接特定 MQTT 裝置。

中斷 MQTT 裝置的連線

1. 在用戶端連線頁面中，找到您要中斷連線的裝置。
2. 選擇用戶端 ID 旁的選項按鈕來選取裝置。
3. 選擇動作，然後選擇中斷連線用戶端。
4. 在中斷連線用戶端對話方塊中，設定中斷連線選項：
   a. 對於清除工作階段，您可以選擇保留或清除工作階段狀態。選取保留工作階段狀態會保留裝置的工作階段資訊，包括訂閱和佇列訊息，允許裝置在重新連線時繼續其先前的工作階段。或者，選擇清除工作階段狀態會在中斷裝置連線時移除所有工作階段資訊，強制裝置在重新連線時建立新的工作階段。
   b. 對於最後遺囑和測試訊息，您可以控制裝置的 LWT 訊息是否在中斷連線時發佈。選擇允許 LWT 訊息會在中斷連線時發佈裝置的 Last Will and Testament 訊息，這會通知其他裝置意外中斷連線。選取防止 LWT 訊息可防止發佈最後遺囑和試驗訊息，這在計劃的維護或控制中斷連線期間非常有用。
5. 選擇中斷連線以確認動作。
6. 當裝置成功中斷連線時，主控台會顯示確認訊息。裝置不會再出現在連線的裝置清單中。

> **Note**
>
> 中斷連線的裝置可以立即嘗試重新連線，除非您已實作其他邏輯來防止重新連線。中斷連線操作只會終止目前的連線。

AWS IoT 與 MQTT 規格的差異

雖然訊息代理程式實作以 MQTT v3.1.1 規格與 MQTT 5.0 規格為基礎，但有些部分與規格有異，內容如下：

- AWS IoT 不支援 MQTT 3 的下列封包：PUBREC、PUBREL 和 PUBCOMP。
- AWS IoT 不支援 MQTT 5 的下列封包：PUBREC、PUBREL、PUBCOMP 和 AUTH。
- AWS IoT 不支援 MQTT 5 伺服器重新導向。

- AWS IoT 僅支援 MQTT 服務品質 (QoS) 層級 0 和 1。 AWS IoT 不支援發佈或訂閱 QoS 層級 2。請求 QoS 2 時，訊息代理程式不會傳送 PUBACK 或 SUBACK。
- 在 中 AWS IoT，訂閱 QoS 層級 0 的主題表示訊息傳遞零次或多次。一則訊息可能會傳送超過一次。傳送超過一次的訊息在發送時可能會使用不同的封包 ID。在這些情況下，DUP 旗標就不會設置。
- 在回應連線請求時，訊息代理程式會傳送 CONNACK 訊息。此訊息包含了一個旗標，指出連線是否會恢復先前的工作階段。

CloudWatch Logs AWS IoT 日誌項目

當 MQTT 用戶端連線Connect時， AWS IoT 訊息中介裝置會使用 eventType 的 產生日誌項目。

連線日誌項目範例

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "thingName": "myDevice",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "keepAliveDuration": 60,
    "sessionExpiry": 3600,
    "cleanSession": boolean
}
```

除了常見的 CloudWatch Logs 屬性之外，Connect 日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

thingName

用戶端的物件名稱。當物件與用戶端 ID 相關聯時，傳回值。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

keepAliveDuration

用戶端在建立連線時指定的保持連線間隔，以秒為單位。

sessionExpiry

用戶端在建立連線時指定的持久性工作階段到期組態。

cleanSession

指出用戶端是否使用乾淨的工作階段。

HTTPS 發佈

用戶端可以使用 HTTP 1.0 或 1.1 通訊協定向 REST API 發出請求來發佈訊息。如需 HTTP 請求所使用的身分驗證和連接埠對應的相關資訊，請參閱 ???。

> **Note**
>
> HTTPS 不像 MQTT 一樣可支援 `clientId` 值。使用 MQTT 時可使用 `clientId`，但使用 HTTPS 時則無法使用。

HTTPS 訊息 URL

裝置和用戶端會將 POST 請求發佈至用戶端特定端點和特定主題的 URL，以發佈其訊息：

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1
```

- *IoT_data_endpoint* 是 AWS IoT 裝置資料端點。您可以使用 命令，在物件的詳細資訊頁面或用戶端 AWS CLI 的 AWS IoT 主控台中找到端點：

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

此端點看起來如下：a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com

- *url_encoded_topic_name* 是所傳送郵件的完整主題名稱。

HTTPS 訊息程式碼範例

以下是如何將 HTTPS 訊息傳送至 AWS IoT的一些範例。

Python (port 8443)

```
import requests
import argparse

# define command-line parameters
parser = argparse.ArgumentParser(description="Send messages through an HTTPS
 connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom
 endpoint, not including a port. " +
                                                       "Ex: \"abcdEXAMPLExyz-
ats.iot.us-east-1.amazonaws.com\"")
parser.add_argument('--cert', required=True, help="File path to your client
 certificate, in PEM format.")
parser.add_argument('--key', required=True, help="File path to your private key, in
 PEM format.")
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to
 publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Message to publish. "
 +
                                                       "Specify empty string to
 publish nothing.")

# parse and load command-line parameter values
args = parser.parse_args()

# create and format values for HTTPS request
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')
```

```
# make request
publish = requests.request('POST',
            publish_url,
            data=publish_msg,
            cert=[args.cert, args.key])

# print results
print("Response status: ", str(publish.status_code))
if publish.status_code == 200:
        print("Response body:", publish.text)
```

Python (port 443)

```
import requests
import http.client
import json
import ssl

ssl_context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
ssl_context.minimum_version = ssl.TLSVersion.TLSv1_2

# note the use of ALPN
ssl_context.set_alpn_protocols(["x-amzn-http-ca"])
ssl_context.load_verify_locations(cafile="./<root_certificate>")

# update the certificate and the AWS endpoint
ssl_context.load_cert_chain("./<certificate_in_PEM_Format>",
 "<private_key_in_PEM_format>")
connection = http.client.HTTPSConnection('<the ats IoT endpoint>', 443,
 context=ssl_context)
message = {'data': 'Hello, I'm using TLS Client authentication!'}
json_data = json.dumps(message)
connection.request('POST', '/topics/device%2Fmessage?qos=1', json_data)

# make request
response = connection.getresponse()

# print results
print(response.read().decode())
```

CURL

您可以使用 curl 從用戶端或裝置傳送訊息至 AWS IoT。

使用 curl 從 AWS IoT 用戶端裝置傳送訊息

1. 檢查 curl 版本。

   a. 在您的用戶端上，請在命令提示中執行此命令。

      curl --help

      在說明文字中，尋找 TLS 選項。您應該看到 `--tlsv1.2` 選項。

   b. 如果您看到 `--tlsv1.2` 選項，請繼續。

   c. 如果您沒有看到 `--tlsv1.2` 選項或您收到 `command not found` 錯誤訊息，您可能需要在用戶端上更新或安裝 curl 或安裝 `openssl`，然後才能繼續進行。

2. 在用戶端上安裝憑證。

   複製您在 AWS IoT 主控台中註冊用戶端 （ 物件 ） 時建立的憑證檔案。在繼續之前，請確定您的用戶端上有這三個憑證檔案。

   - 憑證授權機構憑證檔案 (此範例中的 *Amazon-root-CA-1.pem*)。
   - 用戶端的憑證檔案 (此範例中的 *device.pem.crt*)。
   - 用戶端的私密金鑰檔案 (此範例中的 *private.pem.key*)。

3. 建立 curl 命令列，取代您帳戶和系統的可取代值。

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://IoT_data_endpoint:8443/topics/topic?qos=1"
```

   --tlsv1.2

   使用 TLS 1.2 (SSL)。

   --cacert *Amazon-root-CA-1.pem*

   驗證對等的憑證授權機構憑證檔案名稱和路徑 (如有必要)。

   --cert *device.pem.crt*

   用戶端的憑證檔案名稱和路徑 (如有必要)。

--key *private.pem.key*

用戶端的私密金鑰檔案名稱和路徑 (如有必要)。

--request POST

HTTP 請求的類型 (此處指 POST)。

--data "*{ \"message\": \"Hello, world\" }*"

您要發佈的 HTTP POST 資料。在這種情況下，這是一個 JSON 字串，其內部引號用反斜線字元 (\) 逸出。

"https://*IoT_data_endpoint*:8443/topics/*topic*?qos=1"

用戶端 AWS IoT 裝置資料端點的 URL，後面接著 HTTPS 連接埠 `:8443`，接著關鍵字 `/topics/`和主題名稱 `topic`，在此情況下為 。指定服務品質作為查詢參數 `?qos=1`。

4. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。

   遵循使用 MQTT 用戶端檢視 AWS IoT MQTT 訊息中的指示並設定主控台，以訂閱用於 curl 命令中且主題名稱為 *topic* 的訊息，或使用 # 的萬用字元主題篩選條件。

5. 測試命令。

   監控 AWS IoT 主控台測試用戶端中的主題時，請前往您的用戶端並發出您在步驟 3 中建立的 curl 命令列。您應該會在主控台中看到用戶端的訊息。

直接傳訊

AWS IoT Core 現在支援直接傳訊。您可以依 MQTT 用戶端 ID 傳送訊息至單一連線的裝置，而不需要裝置訂閱主題。

先前，傳送訊息至需要發佈至裝置訂閱之主題的特定裝置，且沒有確認交付的內建方式。寄件者呼叫 SendDirectMessage HTTP API，指定接收者的用戶端 ID 和目標主題。當 時`confirmation=true`，會在 QoS 1 AWS IoT Core 傳送並等待接收者的 PUBACK，然後再傳回成功的回應。這可為您提供 end-to-end交付確認。API 回應和 Amazon CloudWatch Logs 提供交付狀態和失敗原因的完整可見性。

AWS IoT 規則不會處理規則執行的直接訊息，也不會為離線裝置排入佇列，也不支援保留的訊息。

在本主題中：

- 先決條件

- SendDirectMessage API
- 接收者用戶端行為

先決條件

寄件者和接收者都需要特定的政策動作，才能使用直接傳訊。寄件者必須具有 `iot:SendDirectMessage` 許可。目標用戶端 ID 指定為 資源，條件`iot:Topic`索引鍵 （選用） 會限制寄件者可以傳送直接訊息的主題。接收者必須具有目標主題的`iot:Receive`許可。接收者不需要`iot:Subscribe`許可 - 無需主題訂閱即可 AWS IoT Core 傳遞直接訊息。如需詳細資訊和範例政策，請參閱 直接傳訊政策範例。

如需 HTTP 請求所使用的身分驗證和連接埠對應的相關資訊，請參閱 ???。

SendDirectMessage API

寄件者可以透過向用戶端特定的 URL 提出 HTTP POST 請求來傳送直接訊息：

```
https://IoT_data_endpoint/connections/client_id/messages?
topic=topic_name&confirmation=true&timeout=10
```

- *`IoT_data_endpoint`* 是 AWS IoT 裝置資料端點。請參閱 the section called “AWS IoT 裝置資料和服務端點” 尋找您的端點。
- *`client_id`* 是要傳送訊息之 MQTT 用戶端的唯一識別符。客戶 IDs 不得超過 128 個字元，且開頭不得為貨幣符號 ($)。當 MQTT 用戶端 IDs 包含 HTTP 請求中無效的字元時，例如空格、正斜線 (/) 和 UTF-8 字元，則必須使用 URL 編碼 （百分比編碼）。如需詳細資訊，請參閱AWS IoT Core 訊息中介裝置和通訊協定限制和配額。
- *`topic_name`* 是接收者收到 URL 編碼訊息的主題。開頭不得為 $。不得為 AWS IoT Core 預留主題。如需主題長度和深度限制，請參閱 AWS IoT Core 服務配額頁面。如需詳細資訊，請參閱AWS IoT Core 訊息中介裝置和通訊協定限制和配額。
- *`confirmation`* 是布林值。設為 時`true`，API 會在 QoS 1 傳送訊息，並等待 MQTT 用戶端傳送交付確認 (PUBACK)，再傳回成功回應。如果未在指定的逾時期間內收到交付確認，API 會傳回 HTTP 504。
- *`##`*是整數，代表訊息交付後等待來自接收用戶端的交付確認 (PUBACK) 的最長時間，以秒為單位。只有在 `confirmation` 設定為 時，才會使用此參數`true`。如果 `confirmation`是 `false`，則會忽略此參數。由於內部處理，總 API 回應時間可能高於此值。將您的 HTTP 用戶端逾時設定為大於此參數的值。

API 回應狀態碼

下表列出 SendDirectMessage API 傳回的 HTTP 狀態碼，以及每個 API 的建議動作。啟用 AWS IoT Core CloudWatch 日誌以查看詳細的 SendDirectMessage 事件日誌，包括程式設計錯誤處理的原因欄位。

SendDirectMessage API 回應狀態碼

HTTP 代碼	建議的動作
200 OK	如果使用 請求交付確認`confirmation=true` ，這表示接收者已確認收到訊息。否則，這表示訊息已成功分派。
400 錯誤的請求	這表示其中一個參數無效。檢閱 HTTP 回應訊息或 CloudWatch 日誌，以識別特定失敗和修正。確保主題名稱和 Client-id 有效且 URL 編碼正確。
403 禁止	這表示寄件者的政策不會在目標用戶端和主題`iot:SendDirectMessage` 上授予 ，或接收者的政策不會在主題`iot:Receive` 上授予 。檢閱 HTTP 回應訊息或 CloudWatch 日誌以識別特定失敗，並更新對應的政策。請參閱直接傳訊政策範例。
404 找不到	這表示目標用戶端 ID 未連線 AWS IoT Core。基於特定原因檢閱 HTTP 回應訊息或 CloudWatch 日誌，確認接收器已連線，然後再試一次。如果回應訊息指出「目標用戶端 ID 未連線，但其具有作用中的持久性工作階段」，則目標用戶端具有未過期的持久性工作階段，但目前處於離線狀態。
413 承載太大	承載超過允許的大小上限。減少承載大小並重試。請參閱AWS IoT Core 服務配額。
429 請求過多	這表示帳戶已超過 SendDirectMessage requests-per-second數限制，或接收者連線已超過傳出發佈限制。檢閱 HTTP 回應訊息或

HTTP 代碼	建議的動作
	CloudWatch 日誌以了解特定原因、降低請求率並實作指數退避。請參閱AWS IoT Core 服務配額。
500 內部伺服器錯誤	這表示未預期的伺服器端錯誤。以指數退避重試請求。如果問題仍然存在，請聯絡 AWS Support with the traceId from the response。
504 閘道逾時	這表示接收者未在指定的逾時期間內傳送 PUBACK。增加逾時值、確認接收者的 MQTT 用戶端為 QoS 1 訊息傳送 PUBACK，或檢查接收者是否處理訊息緩慢。

範例

AWS CLI

```
aws iot-data send-direct-message \
    --client-id myDevice \
    --topic commands/reboot \
    --confirmation \
    --timeout 10 \
    --payload '{"action": "reboot"}' \
    --cli-binary-format raw-in-base64-out \
    --region us-west-2 \
    --endpoint-url https://IoT_data_endpoint
```

如果您使用的是第 2 AWS Command Line Interface 版，則需要 `--cli-binary-format`選項。若要讓此成為預設的設定，請執行 `aws configure set cli-binary-format raw-in-base64-out`。若要取得更多資訊，請參閱《AWS Command Line Interface 使用者指南第 2 版》中 AWS CLI 支援的全域命令列選項。

curl (X.509 client certificate, port 8443)

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
```

```
    --request POST \
    --data '{"action": "reboot"}' \
    "https://IoT_data_endpoint:8443/connections/myDevice/messages?topic=commands
%2Freboot&confirmation=true&timeout=10"
```

接收者用戶端行為

直接傳訊會傳送訊息給 MQTT 用戶端 （接收者），而不需要主題訂閱。若要完全受益於直接傳訊，接收者必須支援下列行為：

- 接收未明確訂閱主題的訊息 — 接收者的直接傳訊可以將訊息傳遞至接收者未明確訂閱的主題。不過，有些 MQTT 用戶端實作會篩選或捨棄未訂閱主題的訊息。如果您的用戶端捨棄這些訊息，直接傳訊僅適用於接收者也訂閱的主題。若要接收任何主題的直接訊息，請確認用戶端的訊息處理常式處理訊息，無論訂閱狀態為何。
- 處理 API 決定的 QoS — 交付訊息的 QoS 層級是由寄件者的 API 請求中的 `confirmation` 參數設定，而不是由接收者的訂閱設定。當 時`confirmation=true`，訊息到達 QoS 1，接收者的用戶端必須傳送 PUBACK 以確認傳遞。當 時`confirmation=false`，訊息會到達 QoS 0，而不需要確認。確保您的用戶端的 MQTT 實作正確處理 QoS 0 和 QoS 1 傳入訊息。

MQTT 主題

MQTT 主題識別 AWS IoT message. AWS IoT clients 透過提供訊息主題名稱來識別他們發佈的訊息。用戶端能透過用 AWS IoT Core註冊主題篩選條件，來識別想要訂閱 (接收) 的簡訊。訊息代理程式使用主題名稱和主題篩選條件，將訊息從發佈用戶端路由至訂閱用戶端。

訊息代理程式會使用主題來識別使用 MQTT 傳送和使用 HTTP 傳送至 HTTPS 訊息 URL 的訊息

雖然 AWS IoT 支援某些預留系統主題，但大多數 MQTT 主題都是由您建立和管理，System designer. AWS IoT uses 主題可用來識別從發佈用戶端接收的訊息，並選取要傳送給訂閱用戶端的訊息，如以下章節所述。為您的系統建立主題命名空間之前，請檢閱 MQTT 主題的特性，以建立最適合您 IoT 系統的主題名稱階層。

主題名稱

主題名稱和主題篩選條件均為 UTF-8 編碼的字串。它們可以使用正斜線 (/) 字元來表示資訊層級，以分離層次的層級。例如，本主題名稱會提及房間 1 中的溫度感應器：

- `sensor/temperature/room1`

在此範例中，其他房間中可能也有其他類型的感應器，例如主題名稱為：

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

> **Note**
>
> 考慮系統中的訊息主題名稱時，請謹記：
>
> - 主題名稱和主題篩選條件會區分大小寫。
> - 主題名稱不得包含個人識別資訊。
> - 以 $ 開頭的主題名稱為僅由 AWS IoT Core使用的預留主題。
> - AWS IoT Core 無法在 AWS 帳戶或 區域之間傳送或接收訊息。

如需有關設計主題名稱和命名空間的詳細資訊，請參閱我們的白皮書，針對 AWS IoT Core設計 MQTT 主題。

如需應用程式如何發佈和訂閱訊息的範例，請從 AWS IoT Core 教學課程入門 和 AWS IoT 裝置 SDKs、行動SDKs和 AWS IoT 裝置用戶端 開始。

> **Important**
>
> 主題命名空間僅限於 AWS 帳戶 和 區域。例如， AWS 帳戶 一個區域中的 所使用的`sensor/temp/room1`主題與另一個區域中的相同 AWS 帳戶所使用的主題，或任何 AWS 帳戶 區域中的任何其他帳戶所使用的`sensor/temp/room1`主題不同。

ARN 主題

所有主題 ARN (Amazon Resource Names) 都具有以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例如，`arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor` 是主題 `application/topic/device/sensor` 的 ARN。

主題名稱篩選條件

訂閱用戶端向訊息中介裝置註冊主題名稱篩選條件，以指定訊息中介裝置應傳送給他們的訊息主題。主題名稱篩選條件可以是單一主題名稱來訂閱單一主題名稱，也可以包含萬用字元來同時訂閱多個主題名稱。

發佈用戶端無法在其發佈的主題名稱中使用萬用字元。

以下表格列出可用於主題篩選條件的萬用字元。

主題萬用字元

萬用字元	相符	備註
#	位於主題階層中及低於其層級的所有字串。	必須是主題篩選條件中的最後一個字元。 必須是其主題階層的層級中唯一的字元。 可以在也包含 + 萬用字元的主題篩選條件中使用。
+	包含該字元的層級中的任何字串。	必須是其主題階層的層級中唯一的字元。 可以在主題篩選條件的多個層級中使用。

對上一個感應器主題名稱範例使用萬用字元：

- 訂閱 `sensor/#` 會收到發佈至 `sensor/`、`sensor/temperature`、`sensor/temperature/room1` 的訊息，但不會收到發佈至 `sensor` 的訊息。
- 訂閱 `sensor/+/room1` 會收到發佈至 `sensor/temperature/room1` 和 `sensor/humidity/room1` 的訊息，但不會收到 `sensor/temperature/room2` 和 `sensor/humidity/room2` 的訊息。

主題篩選條件 ARN

所有主題篩選條件 ARN (Amazon Resource Names) 都具有以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

例如，arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/+/sensor 是主題篩選條件 application/topic/+/sensor 的 ARN。

MQTT 訊息承載

MQTT 訊息中傳送的訊息承載不會由 指定 AWS IoT，除非是針對其中一個 the section called “預留主題”。為了滿足您的應用程式的需求，我們建議您在通訊協定的AWS IoT Core 服務配額限制範圍內定義主題的訊息承載。

為您的訊息承載使用 JSON 格式，可讓 AWS IoT 規則引擎剖析您的訊息，並將 SQL 查詢套用至其中。如果您的應用程式不需要規則引擎將 SQL 查詢套用至訊息承載，您可以使用應用程式需要的任何資料格式。如需 JSON 文件 (使用於 SQL 查詢) 的限制和預留字元的相關資訊，請參閱 JSON Extensions。

如需設計 MQTT 主題及其對應訊息承載的詳細資訊，請參閱針對 AWS IoT Core設計 MQTT 主題。

如果訊息大小限制超過了服務配額，則會導致 CLIENT_ERROR，原因為 PAYLOAD_LIMIT_EXCEEDED，並顯示 Message payload exceeds size limit for message type (訊息承載超過訊息類型的大小限制)。如需訊息大小限制的詳細資訊，請參閱 AWS IoT Core 訊息代理程式限制和配額。

預留主題

以美元符號 ($) 開頭的主題保留供 使用 AWS IoT。在允許的情況下，您可以訂閱並發佈至這些預留主題；然而您無法建立以貨幣符號開頭的新主題。不支援的發佈或訂閱預留主題作業可能會導致連線終止。

資產模型主題

主題	允許的用戶端操作	說明
$aws/sitewise/asset-models/*assetModelId* /assets/*assetId*/properties/*propertyId*	訂閱	AWS IoT SiteWise 會將資產屬性通知發佈至此主題。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的與其他 AWS 服務互動。

AWS IoT Device Defender 主題

這些訊息支援簡潔二進位物件表示法 (CBOR) 格式的回應緩衝區和 JavaScript 物件表示法 (JSON)，具體取決於 topic. AWS IoT Device Defender topics *#####*，僅支援 MQTT 發佈。

####	回應格式資料類型
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript 物件標記法 (JSON)

如需詳細資訊，請參閱從裝置傳送指標。

主題	允許操作	說明
$aws/things/*thingName* /defender/metrics/*payload-format*	發布	AWS IoT Device Defender 代理程式會將指標發佈至此主題。如需詳細資訊，請參閱從裝置傳送指標。
$aws/things/*thingName* /defender/metrics/*payload-format* /accepted	訂閱	AWS IoT 在 AWS IoT Device Defender 客服人員成功將訊息發佈至 $aws/things/*thingName* /defender/metrics/*payload-format* 之後，會發佈至此主題。如需詳細資訊，請參閱從裝置傳送指標。
$aws/things/*thingName* /defender/metrics/*payload-format* /rejected	訂閱	AWS IoT 在 AWS IoT Device Defender 代理程式將失敗訊息發佈至 $aws/things/*thingName* /defender/metrics/*payload-format* 之後，會發佈至此主題。如需詳細資訊，請參閱從裝置傳送指標。

AWS IoT Core 裝置位置主題

AWS IoT Core Device Location 可以從您的裝置解析測量資料，並提供 IoT 裝置的預估位置。來自裝置的測量資料可以包含 GNSS、Wi-Fi、行動和 IP 地址。然後，AWS IoT Core Device Location 會選

擇提供最佳準確度的測量類型，並解決裝置位置資訊的問題。如需詳細資訊，請參閱AWS IoT Core 裝置位置及使用裝置位置 MQTT 主題解決 AWS IoT Core 裝置位置。

主題	允許操作	說明
$aws/device_location/*customer_device_id*/get_position_estimate	發布	裝置會發佈至此主題，以取得 AWS IoT Core 裝置位置要解析的掃描原始測量資料。
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	訂閱	AWS IoT Core 裝置位置成功解析裝置位置後，會發佈至此主題。
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	訂閱	AWS IoT Core 當裝置位置因 4xx 錯誤而無法成功解析裝置位置時，裝置位置會發佈至此主題。

事件主題

當發生特定事件時，會發佈事件訊息。例如，在新增、更新或刪除事物時，由登錄檔產生的事件。資料表顯示各種 AWS IoT 事件及其預留主題。

主題	允許的用戶端操作	說明
$aws/events/certificates/registered/*caCertificateId*	訂閱	AWS IoT 當 AWS IoT 自動註冊憑證，以及當用戶端呈現PENDING_ACTIVATION 狀態為 的憑證時， 會發佈此訊息。如需詳細資訊，請參閱the section called “設定用戶端的首次連線進行自動註冊”。
$aws/events/job/*jobID*/canceled	訂閱	AWS IoT 會在任務取消時發佈此訊息。如需詳細資訊，請參閱任務事件。

主題	允許的用戶端操作	說明
$aws/events/job/*jobID*/cancellation_in_progress	訂閱	AWS IoT 會在任務取消進行中時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobID*/completed	訂閱	AWS IoT 當任務完成時， 會發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobID*/deleted	訂閱	AWS IoT 會在刪除任務時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobID*/deletion_in_progress	訂閱	AWS IoT 會在任務刪除進行中時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/canceled	訂閱	AWS IoT 會在任務執行取消時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/deleted	訂閱	AWS IoT 會在刪除任務執行時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/failed	訂閱	AWS IoT 當任務執行失敗時， 會發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/rejected	訂閱	AWS IoT 當任務執行遭拒時， 會發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/removed	訂閱	AWS IoT 會在任務執行移除時發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/succeeded	訂閱	AWS IoT 當任務執行成功時， 會發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobID*/timed_out	訂閱	AWS IoT 當任務執行逾時時， 會發佈此訊息。如需詳細資訊，請參閱任務事件。
$aws/events/presence/connected/*clientId*	訂閱	AWS IoT 當具有指定用戶端 ID 的 MQTT 用戶端連線到 時， 會發佈至此主題 AWS IoT。如需詳細資訊，請參閱連線/中斷連線事件。

主題	允許的用戶端操作	說明
$aws/events/presence/disconnected/*clientId*	訂閱	AWS IoT 當具有指定用戶端 ID 的 MQTT 用戶端中斷連線時，會發佈至此主題 AWS IoT。如需詳細資訊，請參閱連線/中斷連線事件。
$aws/events/subscriptions/subscribed/*clientId*	訂閱	AWS IoT 當具有指定用戶端 ID 的 MQTT 用戶端訂閱 MQTT 主題時，會發佈至此主題。如需詳細資訊，請參閱訂閱/取消訂閱事件。
$aws/events/subscriptions/unsubscribed/*clientId*	訂閱	AWS IoT 當具有指定用戶端 ID 的 MQTT 用戶端取消訂閱 MQTT 主題時，會發佈至此主題。如需詳細資訊，請參閱訂閱/取消訂閱事件。
$aws/events/thing/*thingName* /created	訂閱	AWS IoT 會在建立 *thingName* 物件時發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thing/*thingName* /updated	訂閱	AWS IoT 當 *thingName* 物件更新時，會發佈至此主題。如需詳細資訊，請參閱 the section called “登錄檔事件”。
$aws/events/thing/*thingName* /deleted	訂閱	AWS IoT 刪除 *thingName* 物件時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingGroup/*thingGroupName* /created	訂閱	AWS IoT 會在物件群組 *thingGroupName* 建立時發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingGroup/*thingGroupName* /updated	訂閱	AWS IoT 當物件群組 *thingGroupName* 更新時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。

主題	允許的用戶端操作	說明
$aws/events/thingGroup/*thingGroupName*/deleted	訂閱	AWS IoT 當刪除物件群組 *thingGroupName* 時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingType/*thingTypeName*/created	訂閱	AWS IoT 會在建立 *thingTypeName* 物件類型時發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingType/*thingTypeName*/updated	訂閱	AWS IoT 當 *thingTypeName* 物件類型更新時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingType/*thingTypeName*/deleted	訂閱	AWS IoT 刪除 *thingTypeName* 物件類型時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingTypeAssociation/thing/*thingName*/*thingTypeName*	訂閱	AWS IoT 當 thing *thingName* 與 物件類型 *thingTypeName* 相關聯或取消關聯時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/added	訂閱	AWS IoT 當 物件 *thingName* 新增至物件群組 *thingGroupName* 時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/removed	訂閱	AWS IoT 從物件群組 *thingName* *thingGroupName* 時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。

主題	允許的用戶端操作	說明
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /added	訂閱	AWS IoT 當物件群組 *childThingGroupName* 新增至物件群組 *parentThingGroupName* 時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /removed	訂閱	AWS IoT 當物件群組 *childThingGroupName* 從物件群組 *parentThingGroupName* 中移除時，會發佈至此主題。如需詳細資訊，請參閱the section called “登錄檔事件”。

機群佈建主題

> **Note**
>
> 此表格中標記為接收的用戶端操作指出直接 AWS IoT 發佈到請求它的用戶端的主題，無論用戶端是否已訂閱主題。即使用戶端尚未訂閱回應訊息，也應該預期會收到這些訊息。這些回應訊息不會透過訊息代理程式傳遞，而且其他用戶端或規則無法訂閱這些訊息。

這些訊息支援 Concise Binary Object Representation (CBOR) 格式和 JavaScript 物件標記法 (JSON) 的回應緩衝區，具體取決於主題的*####*。

####	回應格式資料類型
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript 物件標記法 (JSON)

如需詳細資訊，請參閱裝置佈建 MQTT API。

主題	允許的用戶端操作	說明
$aws/certificates/create/*payload-format*	發布	發佈到這個主題，以從憑證簽署要求 (CSR) 建立憑證。
$aws/certificates/create/*payload-format* /accepted	訂閱、接收	AWS IoT 在成功呼叫 $aws/certificates/create/*payload-format* 之後，會發佈至此主題。
$aws/certificates/create/*payload-format* /rejected	訂閱、接收	AWS IoT 在呼叫 $aws/certificates/create/*payload-format* 失敗之後，會發佈至此主題。
$aws/certificates/create-from-csr/*payload-format*	發布	發佈至本主題，以從 CSR 建立憑證。
$aws/certificates/create-from-csr/*payload-format* /accepted	訂閱、接收	AWS IoT 成功呼叫 $aws/certificates/create-from-csr/*payload-format* ，發佈至此主題。
$aws/certificates/create-from-csr/*payload-format* /rejected	訂閱、接收	AWS IoT 會將呼叫失敗的 $aws/certificates/create-from-csr/*payload-format* 發佈至此主題。
$aws/provisioning-templates/*templateName* /provision/*payload-format*	發布	發佈至此主題以註冊實物。
$aws/provisioning-templates/*templateName* /provision/*payload-format* /accepted	訂閱、接收	AWS IoT 在成功呼叫 $aws/provisioning-templates/*templateName* /provision/*payload-format* 之後，會發佈至此主題。
$aws/provisioning-templates/*templateName* /provision/*payload-format* /rejected	訂閱、接收	AWS IoT 在呼叫 $aws/provisioning-templates/*templateName* /provision/*payload-format* 失敗之後，會發佈至此主題。

任務主題

> **Note**
>
> 此表格中標記為接收的用戶端操作指出直接 AWS IoT 發佈到請求它的用戶端的主題，無論用戶端是否已訂閱主題。即使用戶端尚未訂閱回應訊息，也應該預期會收到這些訊息。
> 這些回應訊息不會透過訊息代理程式傳遞，而且其他用戶端或規則無法訂閱這些訊息。若要訂閱任務活動相關訊息，請使用 `notify` 和 `notify-next` 主題。
> 為機群監控解決方案訂閱任務和 `jobExecution` 事件主題時，您必須先啟用任務和任務執行事件來接收雲端上的任何事件。
> 如需詳細資訊，請參閱任務裝置 MQTT API 操作。

主題	允許的用戶端操作	說明
$aws/things/*thingName*/jobs/get	發布	裝置發佈訊息到這個主題來提出 `GetPendingJobExecutions` 請求。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName*/jobs/get/accepted	訂閱、接收	裝置訂閱此主題，以接收來自 `GetPendingJobExecutions` 請求的成功回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName*/jobs/get/rejected	訂閱、接收	當 `GetPendingJobExecutions` 請求被拒時，裝置會訂閱此主題以接收回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName*/jobs/start-next	發布	裝置發佈訊息到這個主題來提出 `StartNextPendingJobExecution` 請求。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName*/jobs/start-next/accepted	訂閱、接收	裝置訂閱此主題，以接收送往 `StartNextPendingJobExecutio`

主題	允許的用戶端操作	說明
		n 請求的成功回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName* / jobs/start-next/rejected	訂閱、接收	當 StartNextPendingJobExecution 請求被拒時，裝置會訂閱此主題以接收回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get	發布	裝置發佈訊息到這個主題來提出 DescribeJobExecution 請求。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get/accepted	訂閱、接收	裝置訂閱此主題，以接收送往 DescribeJobExecution 請求的成功回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get/rejected	訂閱、接收	當 DescribeJobExecution 請求被拒時，裝置會訂閱此主題以接收回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/update	發布	裝置發佈訊息到這個主題來提出 UpdateJobExecution 請求。如需詳細資訊，請參閱任務裝置 MQTT API 操作。

主題	允許的用戶端操作	說明
$aws/things/*thingName*/jobs/*jobId*/update/accepted	訂閱、接收	裝置訂閱此主題，以接收送往 `UpdateJobExecution` 請求的成功回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。 **注意** 只有發佈到 $aws/things/*thingName*/jobs/*jobId*/update 的設備才會收到此主題的訊息。
$aws/things/*thingName*/jobs/*jobId*/update/rejected	訂閱、接收	當 `UpdateJobExecution` 請求被拒時，裝置會訂閱此主題以接收回應。如需詳細資訊，請參閱任務裝置 MQTT API 操作。 **注意** 只有發佈到 $aws/things/*thingName*/jobs/*jobId*/update 的設備才會收到此主題的訊息。
$aws/things/*thingName*/jobs/notify	訂閱、接收	裝置訂閱此主題，以在物件的待處理執行清單新增或移除任務執行時接收通知。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/things/*thingName*/jobs/notify-next	訂閱、接收	裝置訂閱此主題，以在物件的下一個待處理任務執行變更時接收通知。如需詳細資訊，請參閱任務裝置 MQTT API 操作。
$aws/events/job/*jobId*/completed	訂閱	任務完成時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。

主題	允許的用戶端操作	說明
$aws/events/job/*jobId*/canceled	訂閱	任務取消時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobId*/deleted	訂閱	任務刪除時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobId*/cancellation_in_progress	訂閱	開始取消任務時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/job/*jobId*/deletion_in_progress	訂閱	開始刪除任務時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/succeeded	訂閱	任務執行成功時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/failed	訂閱	任務執行失敗時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/rejected	訂閱	任務執行遭拒時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/canceled	訂閱	任務執行取消時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/timed_out	訂閱	任務執行逾時時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。
$aws/events/jobExecution/*jobId*/removed	訂閱	任務執行移除時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。

主題	允許的用戶端操作	說明
$aws/events/jobExecution/*jobId*/deleted	訂閱	任務執行刪除時，任務服務會發佈這個主題的事件。如需詳細資訊，請參閱任務事件。

命令主題

> **Note**
>
> 此表格中標記為接收的用戶端操作指出直接 AWS IoT 發佈到請求它的用戶端的主題，無論用戶端是否已訂閱主題。即使用戶端尚未訂閱回應訊息，也應該預期會收到這些訊息。
> 這些回應訊息不會透過訊息代理程式傳遞，而且其他用戶端或規則無法訂閱這些訊息。

主題	允許的用戶端操作	說明
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/request/*<PayloadFormat>* $aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/request	訂閱、接收	當發出從主控台或使用 `StartCommandExecution` API 啟動命令執行的請求時，裝置會收到有關此主題的訊息。在此情況下，*<devices>* 可以是 IoT 物件或 MQTT 用戶端，而 *<DeviceID>* 可以是 IoT 物件名稱或 MQTT 用戶端 ID。
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/response/*<PayloadFormat>*	發布	裝置使用 `UpdateCommandExecution` MQTT API 將有關命令執行的訊息發佈至此主題。訊息會發佈為從主控台或使用 `StartCommandExecution` API 啟動命令執行之請求的回應。

主題	允許的用戶端操作	說明
		發佈的訊息將使用 JSON 或 CBOR 做為 *<PayloadFormat>* 。
$aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/accepted /*<PayloadFormat>* $aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/accepted	訂閱、接收	如果雲端服務成功處理命令執行結果，會將回應 AWS IoT Device Management 發佈至 /接受的主題。
$aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/rejected /*<PayloadFormat>* $aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/rejected	發布	如果雲端服務無法處理命令執行結果，會將回應 AWS IoT Device Management 發佈至 /拒絕主題。

規則主題

主題	允許的用戶端操作	說明
$aws/rules/*ruleName*	發布	裝置或應用程式會發佈至此主題，以直接觸發規則。如需詳細資訊，請參閱使用基本擷取減少簡訊費用。

安全通道主題

主題	允許的用戶端操作	說明
$aws/things/*thing-name* /tunnels/notify	訂閱	AWS IoT 會發佈此訊息，讓 IoT 代理程式在遠端裝置上啟動本機代理。如需詳細資訊，請參閱the section called “IoT Agent Snippet”。

影子主題

已命名的影子和未命名的影子會使用本節中的主題。各影子所使用的主題只有在主題字首中有所不同。此表格會顯示每種影子類型所使用的主題字首。

ShadowTopicPrefix 值	影子類型
$aws/things/*thingName* /shadow	未命名 (經典) 影子
$aws/things/*thingName* /shadow/name/*shadowName*	已命名影子

若要建立完整的主題，請為您想要參照的影子類型選取 *ShadowTopicPrefix*，取代 *thingName* 和 *shadowName* (如果適用)，並使用其對應的值，然後將其附加到主題 stub，如下表所示。請記住，主題會區分大小寫。

主題	允許的用戶端操作	說明
ShadowTopicPrefix /delete	發佈/訂閱	裝置或應用程式會發佈至此主題，以刪除影子。如需詳細資訊，請參閱 /delete。
ShadowTopicPrefix /delete/accepted	訂閱	當影子刪除時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /delete/accepted。
ShadowTopicPrefix /delete/rejected	訂閱	拒絕刪除影子的要求時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /delete/rejected。
ShadowTopicPrefix /get	發佈/訂閱	應用程式或物件會發佈空白訊息至此主題，以取得影子。如需詳細資訊，請參閱Device Shadow MQTT 主題。
ShadowTopicPrefix /get/accepted	訂閱	當要求影子成功時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /get/accepted。
ShadowTopicPrefix /get/rejected	訂閱	當要求影子被拒絕時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /get/rejected。
ShadowTopicPrefix /update	發佈/訂閱	物件或應用程式會發佈至此主題，以更新影子。如需詳細資訊，請參閱 /update。
ShadowTopicPrefix /update/accepted	訂閱	當更新影子成功時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /update/accepted。
ShadowTopicPrefix /update/rejected	訂閱	當更新影子被拒絕時，Device Shadow 服務會傳送訊息至此主題。如需詳細資訊，請參閱 /update/rejected。
ShadowTopicPrefix /update/delta	訂閱	當偵測到回報的區段與所需的區段之間發生差異時，Device Shadow 服務會傳送

主題	允許的用戶端操作	說明
		訊息至此主題。如需詳細資訊，請參閱 /update/delta。
ShadowTopicPrefix /update/documents	訂閱	AWS IoT 每當成功執行影子更新時，就會發佈狀態文件至此主題。如需詳細資訊，請參閱 /update/documents。

MQTT 型檔案交付主題

> **Note**
>
> 此表格中標記為接收的用戶端操作指出直接 AWS IoT 發佈到請求它的用戶端的主題，無論用戶端是否已訂閱主題。即使用戶端尚未訂閱回應訊息，也應該預期會收到這些訊息。這些回應訊息不會透過訊息代理程式傳遞，而且其他用戶端或規則無法訂閱這些訊息。

這些訊息支援 Concise Binary Object Representation (CBOR) 格式和 JavaScript 物件標記法 (JSON) 的回應緩衝區，具體取決於主題的*####*。

####	回應格式資料類型
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript 物件標記法 (JSON)

主題	允許的用戶端操作	說明
$aws/things/*ThingName*/streams/*StreamId*/data/*payload-format*	訂閱、接收	AWS 如果接受來自裝置的「GetStream」請求，則 MQTT 型檔案交付會發佈至此主題。承載包含串流資料。如需詳細資訊，請參閱在裝置中使用 AWS IoT MQTT 型檔案交付。

主題	允許的用戶端操作	說明
$aws/things/*ThingName* /streams/*StreamId*/get/*payload-format*	發布	裝置會發佈至本主題，以執行 "GetStream" 請求。如需詳細資訊，請參閱在裝置中使用 AWS IoT MQTT 型檔案交付。
$aws/things/*ThingName* /streams/*StreamId*/description/*payload-format*	訂閱、接收	AWS 如果接受來自裝置的「DescribeStream」請求，則 MQTT 型檔案交付會發佈至此主題。承載包含串流描述。如需詳細資訊，請參閱在裝置中使用 AWS IoT MQTT 型檔案交付。
$aws/things/*ThingName* /streams/*StreamId*/describe/*payload-format*	發布	裝置會發佈至本主題，以執行 "DescribeStream" 請求。如需詳細資訊，請參閱在裝置中使用 AWS IoT MQTT 型檔案交付。
$aws/things/*ThingName* /streams/*StreamId*/rejected/*payload-format*	訂閱、接收	AWS 如果來自裝置的「DescribeStream」或「GetStream」請求遭到拒絕，則 MQTT 型檔案交付會發佈至此主題。如需詳細資訊，請參閱在裝置中使用 AWS IoT MQTT 型檔案交付。

預留的主題 ARN

所有預留的主題 ARN (Amazon Resource Names) 都具有以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例如，`arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted` 是預留主題 `$aws/things/thingName/jobs/get/accepted` 的 ARN。

網域組態

在 中 AWS IoT Core，您可以使用網域組態來設定和管理資料端點的行為。使用網域組態，您可以產生多個 AWS IoT Core 資料端點、使用自己的完整網域名稱 (FQDN) 和相關聯的伺服器憑證來自訂它們，以及建立自訂授權方的關聯。如需詳細資訊，請參閱自訂身分驗證和授權。

在本章中：

- 什麼是網域組態？
- 建立和設定 AWS 受管網域
- 建立和設定客戶受管網域
- 管理網域組態
- 在網域組態中設定 TLS 設定
- OCSP 裝訂的伺服器憑證組態

什麼是網域組態？

在 中 AWS IoT Core，網域組態是指 AWS IoT Core 資料端點的網域（AWS 受管網域或客戶受管網域）的設定和組態。 AWS IoT Core 也為 AWS 您的帳戶 (`iot:Data-ATS`) 提供預設端點，供裝置與之通訊 AWS IoT Core。

在本主題中：

- 使用案例
- 重要概念
- 重要說明

使用案例

您可以使用網域組態來簡化任務，如下所示。

- 將裝置遷移至 AWS IoT Core。
- 維護個別裝置類型的不同網域組態，以支援異質裝置機群。
- 將應用程式基礎設施遷移至 時，維護品牌身分（例如透過網域名稱） AWS IoT Core。

重要概念

下列概念提供有關網域組態和相關概念的詳細資訊。

- 網域組態

  AWS IoT Core 端點網域的設定和組態。

- 預設端點網域

  AWS IoT 提供預設端點的網域，例如 `iot:Data-ATS`。若要尋找預設端點，請執行 describe-endpoint 或 describe-domain-configuration CLI 命令。或者，前往 AWS IoT Core 主控台，從左側導覽的 Connect 選擇網域組態。預設端點會以名稱 列出`iot:Data-ATS`。

- AWS 受管網域

  AWS 將管理的網域。選擇 AWS 受管網域表示您的裝置將使用 提供的資料端點進行連線 AWS。AWS 將管理網域和憑證。

- 客戶受管網域

  您將管理的網域。也稱為自訂網域。選擇客戶受管網域表示您的裝置將使用自訂網域資料端點進行連線。您將會管理網域和憑證。客戶受管網域可讓您量身打造端點 URLs 以符合您的需求。例如，您可以使用自訂網域名稱 (`your-domain-name.com`) 或套用特定存取政策。

- 身分驗證類型

  您在連線時選擇對裝置進行身分驗證的身分驗證類型 AWS IoT Core。建立網域組態時，您必須指定身分驗證類型。如需詳細資訊，請參閱???。

- 應用程式通訊協定

  您的裝置連線至 時所使用的應用程式層通訊協定 AWS IoT Core。建立網域組態時，您必須指定應用程式通訊協定。如需詳細資訊，請參閱???。

重要說明

AWS IoT Core 使用伺服器名稱指示 (SNI) TLS 延伸來套用網域組態。將裝置連線至 時 AWS IoT Core，用戶端可以傳送伺服器名稱指示 (SNI) 延伸，這是多帳戶註冊、可設定端點、自訂網域和 VPC 端點等功能的必要項目。它們也必須傳遞與您在網域組態中指定之網域名稱相同的伺服器名稱。若要測試此服務，請使用 GitHub 中的 AWS IoT 裝置 SDK。

如果您在 中建立多個資料端點 AWS 帳戶，它們將共用 AWS IoT Core 資源，例如 MQTT 主題、裝置影子和規則。

當您提供 AWS IoT Core 自訂網域組態的伺服器憑證時，憑證最多有四個網域名稱。如需詳細資訊，請參閱 AWS IoT Core 端點和配額。

建立和設定 AWS 受管網域

您可以使用 CreateDomainConfiguration API 在 AWS 受管網域上建立可設定的端點。 AWS 受管網域的網域組態包含下列項目：

- `domainConfigurationName`

  識別網域組態和 值的使用者定義名稱對您的 必須是唯一的 AWS 區域。您無法使用以 `IoT:` 開頭的網域組態名稱，因為它們是保留給預設端點。
- `defaultAuthorizerName` (選用)

  要在端點上使用的自訂授權方名稱。
- `allowAuthorizerOverride` (選用)

  布林值，用來指定裝置是否可以在請求的 HTTP 標頭中指定不同的授權方來覆寫預設授權方。如果指定了 `defaultAuthorizerName` 的值，則需要此值。
- `serviceType` (選用)

  端點交付的服務類型。 AWS IoT Core 僅支援 DATA服務類型。當您指定 DATA 時， AWS IoT Core 會傳回端點類型為 `iot:Data-ATS` 的端點。您無法建立可設定的 `iot:Data` (VeriSign) 端點。
- `TlsConfig` (選用)

  為網域指定 TLS 組態的物件。如需詳細資訊，請參閱???。

下列範例 AWS CLI 命令會建立Data端點的網域組態。

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
```

命令的輸出可能如下所示。

```
{
    "domainConfigurationName": "myDomainConfigurationName",
    "domainConfigurationArn": "arn:aws:iot:us-east-1:123456789012:domainconfiguration/
myDomainConfigurationName/itihw"
}
```

建立和設定客戶受管網域

網域組態可讓您指定自訂完整網域名稱 (FQDN) 來連線至 AWS IoT Core。使用客戶受管網域 （ 也稱為自訂網域 ） 有許多好處： 您可以公開自有網域或公司自有網域給客戶用於品牌用途；您可以輕鬆變更自有網域以指向新的中介裝置；您可以支援多租用戶，為擁有相同網域的客戶提供服務 AWS 帳戶；而且您可以管理自己的伺服器憑證詳細資訊，例如用來簽署憑證的根憑證授權機構 (CA)、簽章演算法、憑證鏈深度，以及憑證的生命週期。

使用自訂網域來設定網域組態的工作流程包含下列三個階段。

1. 在 中註冊伺服器憑證 AWS Certificate Manager
2. 建立網域組態
3. 建立 DNS 記錄

在憑證管理器中註冊伺服器 AWS 憑證

使用自訂網域建立網域組態之前，您必須在 AWS Certificate Manager (ACM) 中註冊伺服器憑證鏈。您可以使用下列三種類型的伺服器憑證。

- ACM 產生的公有憑證
- 由公有 CA 簽署的外部憑證
- 由私有 CA 簽署的外部憑證

> **Note**
>
> AWS IoT Core 如果憑證包含在 Mozilla 的信任 ca-bundle 中，則 會將憑證視為由公有 CA 簽署。

憑證需求

如需將憑證匯入至 ACM 的需求，請參閱匯入憑證的先決條件。除了這些需求之外， AWS IoT Core 會新增下列需求。

- 分葉憑證必須包含值為 serverAuth (TLS Web Server Authentication) 的擴充金鑰用量 x509 v3 延伸。如果您從 ACM 請求憑證，則會自動新增此延伸。
- 憑證鏈深度上限為 5 個憑證。

- 憑證鏈大小上限為 16KB。
- 支援的加密演算法和金鑰大小包括 RSA 2048 位元 (RSA_2048) 和 ECDSA 256 位元 (EC_prime256v1)。

針對多個網域使用一個憑證

如果您打算使用一個憑證來涵蓋多個子網域，請在通用名稱 (CN) 或主體替代名稱 (SAN) 欄位中使用萬用字元網域。例如，使用 **`*.iot.example.com`** 來涵蓋 dev.iot.example.com、qa.iot.example.com 和 prod.iot.example.com。每個 FQDN 都需要自己的網域組態，但多個網域組態可以使用相同的萬用字元值。CN 或 SAN 必須涵蓋您要用作自訂網域的 FQDN。如果 SAN 存在，CN 會遭忽略，且 SAN 必須涵蓋您要用作自訂網域的 FQDN。此涵蓋範圍可以是完全相符或萬用字元相符。驗證萬用字元憑證並向帳戶註冊後，該區域中的其他帳戶將無法建立與憑證重疊的自訂網域。

下列各節說明如何取得每種類型的憑證。每個憑證資源都需要向 ACM 註冊的 Amazon 資源名稱 (ARN)，您在建立網域組態時會使用此 ARN。

ACM 產生的公有憑證

您可以使用 RequestCertificate API，為您的自訂網域產生公有憑證。當您以這種方式產生憑證時，ACM 會驗證您對自訂網域的擁有權。如需詳細資訊，請參閱《AWS Certificate Manager 使用者指南》中的請求公有憑證。

由公有 CA 簽署的外部憑證

如果您已經擁有由公有 CA (此 CA 包含在 Mozilla 信任的 ca-bundle 中) 簽署的伺服器憑證，您可以使用 ImportCertificate API，將憑證鏈直接匯入至 ACM。若要進一步了解此任務以及必要條件和憑證格式需求，請參閱匯入憑證。

由私有 CA 簽署的外部憑證

如果您已經擁有由私有 CA 簽署或自我簽署的伺服器憑證，則可以使用憑證來建立您的網域組態，但也必須在 ACM 中建立額外的公有憑證，以驗證網域的擁有權。若要這樣做，請在 ACM 使用 ImportCertificate API 來註冊您的伺服器憑證鏈。若要進一步了解此任務以及必要條件和憑證格式需求，請參閱匯入憑證。

建立驗證憑證

將您的憑證匯入至 ACM 之後，請使用 RequestCertificate API，為您的自訂網域產生公有憑證。當您以這種方式產生憑證時，ACM 會驗證您對自訂網域的擁有權。如需詳細資訊，請參閱請求公有憑證。建立網域組態時，請使用此公有憑證作為您的驗證憑證。

建立網域組態

您可以使用 CreateDomainConfiguration API，在自訂網域上建立可設定端點。自訂網域的網域組態包含下列項目：

- `domainConfigurationName`

  識別網域組態的使用者定義名稱。以 `IoT:` 開頭的網域組態名稱會保留給預設端點，且無法使用。此外，此值對於您的 必須是唯一的 AWS 區域。

- `domainName`

  裝置用來連線的 FQDN AWS IoT Core。 AWS IoT Core 會利用伺服器名稱指示 (SNI) TLS 延伸來套用網域組態。裝置在連線並傳遞與網域組態中指定之網域名稱相同的伺服器名稱時，必須使用此延伸。

- `serverCertificateArns`

  您向 ACM 註冊的伺服器憑證鏈 ARN。 AWS IoT Core 目前僅支援一個伺服器憑證。

- `validationCertificateArn`

  您在 ACM 中產生之公有憑證的 ARN，此憑證用驗證自訂網域的擁有權。如果您使用公開簽署或 ACM 產生的伺服器憑證，則不需要此引數。

- `defaultAuthorizerName (optional)`

  要在端點上使用的自訂授權方名稱。

- `allowAuthorizerOverride`

  布林值，用來指定裝置是否可以在請求的 HTTP 標頭中指定不同的授權方來覆寫預設授權方。如果指定了 `defaultAuthorizerName` 的值，則需要此值。

- `serviceType`

  AWS IoT Core 目前僅支援 DATA服務類型。當您指定 時DATA， 會 AWS IoT 傳回端點類型為 的端點`iot:Data-ATS`。

- `TlsConfig` (選用)

  為網域指定 TLS 組態的物件。如需詳細資訊，請參閱???。

- `serverCertificateConfig` (選用)

  指定網域之伺服器憑證組態的物件。如需詳細資訊，請參閱???。

下列 AWS CLI 命令會建立 iot.example.com 的網域組態。

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
--domain-name "iot.example.com" --server-certificate-arns serverCertARN --validation-
certificate-arn validationCertArn
```

> **Note**
>
> 建立網域組態後，最多可能需要 60 分鐘，直到 AWS IoT Core 提供您的自訂伺服器憑證為止。

如需詳細資訊，請參閱???。

建立 DNS 記錄

在註冊您的伺服器憑證鏈並建立您的網域組態之後，請建立 DNS 記錄，讓您的自訂網域可以指向 AWS IoT 網域。此記錄必須指向 類型的 AWS IoT 端點`iot:Data-ATS`。您可以使用 DescribeEndpoint API 來取得端點。

下列 AWS CLI 命令顯示如何取得您的端點。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

取得`iot:Data-ATS`端點後，請建立從自訂網域到此 AWS IoT 端點`CNAME`的記錄。如果您在相同的中建立多個自訂網域 AWS 帳戶， 會將它們別名為這個相同的`iot:Data-ATS`端點。

疑難排解

如果您無法將裝置連線至自訂網域，請確定 AWS IoT Core 已接受並套用您的伺服器憑證。您可以使用 AWS IoT Core 主控台或 驗證 AWS IoT Core 是否已接受您的憑證 AWS CLI。

若要使用 AWS IoT Core 主控台，請導覽至網域組態頁面，然後選取網域組態名稱。在 Server certificate details (伺服器憑證詳細資訊) 區段中，檢查狀態和狀態詳細資訊。如果憑證無效，請在 ACM 中將其取代為符合上節所列憑證需求的憑證。如果憑證具有相同的 ARN， AWS IoT Core 會自動將其提取並套用。

若要使用 檢查憑證狀態 AWS CLI，請呼叫 DescribeDomainConfiguration API 並指定您的網域組態名稱。

> **Note**
>
> 如果您的憑證無效，AWS IoT Core 將繼續提供最後一個有效的憑證。

您可以使用下列 openssl 命令，檢查正在端點上為哪個憑證提供服務。

```
openssl s_client -connect custom-domain-name:8883 -showcerts -servername
custom-domain-name
```

管理網域組態

本主題涵蓋管理網域組態資源的關鍵操作。您也可以使用下列 APIs 來管理現有組態的生命週期：ListDomainConfigurations、DescribeDomainConfiguration、UpdateDomainConfiguration 和 DeleteDomainConfiguration。

在本主題中：

- 檢視網域組態
- 更新網域組態
- 刪除網域組態
- 在自訂網域中輪換憑證

檢視網域組態

若要傳回 中所有網域組態的分頁清單 AWS 帳戶，請使用 ListDomainConfigurations API 。您可以使用 DescribeDomainConfiguration API 來查看特定網域組態的詳細資訊。此 API 採用單一 domainConfigurationName 參數，並傳回所指定組態的詳細資訊。

範例

更新網域組態

若要更新網域組態的狀態或自訂授權方，請使用更 UpdateDomainConfiguration API。您可以將狀態設為 ENABLED 或 DISABLED。如果您停用網域組態，連線至該網域的裝置會收到驗證錯誤。目前您無法在網域組態中更新伺服器憑證。若要變更網域組態的憑證，您必須刪除並重新建立它。

範例

刪除網域組態

刪除網域設定之前，請使用 UpdateDomainConfiguration API，將狀態設為 DISABLED。這可協助您避免意外刪除端點。停用網域組態之後，請使用 DeleteDomainConfiguration API 將其刪除。您必須將AWS受管網域置於 DISABLED 狀態 7 天，才能將其刪除。您可以讓自訂網域處於 DISABLED 狀態，然後一次刪除它們。

範例

刪除網域組態後， AWS IoT Core 不再提供與該自訂網域相關聯的伺服器憑證。

在自訂網域中輪換憑證

您可能需要定期將伺服器憑證取代為更新的憑證。您執行此動作的速率取決於憑證的有效期間。如果您已使用 AWS Certificate Manager (ACM) 產生伺服器憑證，則可以將憑證設定為自動更新。當 ACM 續約您的憑證時， AWS IoT Core 會自動挑選新的憑證。您不必執行任何額外的動作。如果您已從不同來源匯入伺服器憑證，則可以將其重新匯入至 ACM 來輪換憑證。如需重新匯入憑證的詳細資訊，請參閱重新匯入憑證。

> **Note**
>
> AWS IoT Core 只會在下列情況下取得憑證更新。
>
> - 新憑證具有與舊憑證相同的 ARN。
> - 新憑證具有與舊憑證相同的簽署演算法、通用名稱或主體別名。

在網域組態中設定 TLS 設定

AWS IoT Core 為您提供預先定義的安全政策，以自訂網域組態中 TLS 1.2 和 TLS 1.3 的 Transport Layer Security (TLS) 設定。安全政策是 TLS 通訊協定及其密碼的組合，用來決定在用戶端與伺服器之間進行 TLS 交涉期間支援的通訊協定和密碼。使用支援的安全政策，您可以更靈活地管理裝置的 TLS 設定、在連線新裝置時套用最新的安全性措施，以及將現有裝置的 TLS 組態保持一致。

下表說明安全政策、其 TLS 版本及支援的區域：

安全政策名稱	支援的 AWS 區域
IoTSecurityPolicy_TLS13_1_3_2022_10	全部 AWS 區域，不包括 AWS GovCloud (US)

安全政策名稱	支援的 AWS 區域
IoTSecurityPolicy_TLS13_1_2_2022_10	全部 AWS 區域，不包括 AWS GovCloud (US)
IoTSecurityPolicy_TLS12_1_2_2022_10	全部 AWS 區域
IoTSecurityPolicy_TLS12_1_0_2016_01	ap-east-1、ap-northeast-2、ap-south-1、ap-southeast-2、ca-central-1、cn-north-1、cn-northwest-1、eu-north-1、eu-west-2、eu-west-3、me-south-1、sa-east-1、us-east-2、us-west-1
IoTSecurityPolicy_TLS12_1_0_2015_01	ap-northeast-1、ap-southeast-1、eu-central-1、eu-west-1、us-east-1、us-west-2
IoTSecurityPolicy_TLS13_1_2_2022_01	AWS GovCloud (US) 僅限 (FIPS 政策)

中的安全政策名稱 AWS IoT Core 包含根據發行月份和年份的版本資訊。如果您建立新的網域組態，則安全政策會針對 以外的`IoTSecurityPolicy_TLS13_1_2_2022_10`所有區域預設為 AWS GovCloud (US)。對於 AWS GovCloud (US) 區域，安全政策預設為`IoTSecurityPolicy_TLS13_1_2_2022_01`，即 FIPS 政策。對於 AWS GovCloud (US) 區域中的非 FIPS 端點，您可以使用 `IoTSecurityPolicy_TLS12_1_2_2022_10`。如需包含通訊協定、TCP 連接埠和密碼詳細資訊的完整安全政策資料表，請參閱安全政策。 AWS IoT Core 不支援自訂安全政策。如需詳細資訊，請參閱???。

若要在網域組態中設定 TLS 設定，您可以使用 AWS IoT 主控台或 AWS CLI。

目錄

- 在網域組態中設定 TLS 設定 (主控台)
- 在網域組態中設定 TLS 設定 (CLI)

在網域組態中設定 TLS 設定 (主控台)

使用 AWS IoT 主控台設定 TLS 設定

1. 登入 AWS 管理主控台 並開啟 AWS IoT 主控台。

2. 若要在建立新網域組態時設定 TLS 設定，請遵循下列步驟。

   1. 在左側導覽窗格中，選擇網域組態，然後選擇建立網域組態。
   2. 在建立網域組態頁面的自訂網域設定 - 選用區段中，從選取安全政策中選擇安全政策。
   3. 遵循小工具並完成其餘步驟。選擇建立網域組態。

3. 若要更新現有網域組態中的 TLS 設定，請遵循下列步驟。

   1. 在左側導覽窗格中，選擇網域組態，然後選擇網域組態。
   2. 在網域組態詳細資訊頁面中，選擇編輯。然後，在自訂網域設定 - 選用區段的選取安全政策下，選擇安全政策。
   3. 選擇更新網域組態。

如需詳細資訊，請參閱建立網域組態和管理網域組態。

在網域組態中設定 TLS 設定 (CLI)

您可以使用 create-domain-configuration 或 update-domain-configuration CLI 命令，在網域組態中設定 TLS 設定。

1. 若要使用 create-domain-configuration CLI 命令指定 TLS 設定：

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

   此令命的輸出如下所示：

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

   如果您建立新的網域組態，而未指定安全政策，則值將預設為 `IoTSecurityPolicy_TLS13_1_2_2022_10`。

2. 若要使用 describe-domain-configuration CLI 命令說明 TLS 設定：

```
aws iot describe-domain-configuration \
```

```
    --domain-configuration-name domainConfigurationName
```

此命令可以傳回網域組態詳細資訊，其中包含 TLS 設定，如下所示：

```
{
 "tlsConfig": {
 "securityPolicy": "IoTSecurityPolicy_TLS13_1_2_2022_10"
 },
 "domainConfigurationStatus": "ENABLED",
 "serviceType": "DATA",
 "domainType": "AWS_MANAGED",
 "domainName": "d1234567890abcdefghij-ats.iot.us-west-2.amazonaws.com",
 "serverCertificates": [],
 "lastStatusChangeDate": 1678750928.997,
 "domainConfigurationName": "test",
 "domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

3. 若要使用 update-domain-configuration CLI 命令更新 TLS 設定：

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

此令命的輸出如下所示：

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

4. 若要更新 ATS 端點的 TLS 設定，請執行 update-domain-configuration CLI 命令。ATS 端點的網域組態名稱為 `iot:Data-ATS`。

```
aws iot update-domain-configuration \
    --domain-configuration-name "iot:Data-ATS" \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

此命令的輸出如下所示：

```
{
"domainConfigurationName": "iot:Data-ATS",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
iot:Data-ATS"
}
```

如需詳細資訊，請參閱《AWS API 參考》中的 CreateDomainConfiguration 和 UpdateDomainConfiguration。

OCSP 裝訂的伺服器憑證組態

AWS IoT Core 支援伺服器憑證的線上憑證狀態通訊協定 (OCSP) 堆疊，也稱為伺服器憑證 OCSP 堆疊或 OCSP 堆疊。它是一種安全機制，用於在 Transport Layer Security (TLS) 交握中檢查伺服器憑證上的撤銷狀態。中的 OCSP 堆疊 AWS IoT Core 可讓您將額外的驗證層新增至自訂網域的伺服器憑證有效性。

您可以定期查詢 OCSP 回應程式，在 中啟用伺服器憑證 OCSP 堆疊 AWS IoT Core ，以檢查憑證的有效性。OCSP 堆疊設定是使用自訂網域建立或更新網域組態程序的一部分。OCSP 堆疊會持續檢查伺服器憑證上的撤銷狀態。這有助於驗證連線至自訂網域的用戶端不再信任 CA 撤銷的任何憑證。如需詳細資訊，請參閱???。

伺服器憑證 OCSP 堆疊提供即時撤銷狀態檢查、減少與檢查撤銷狀態相關的延遲，並改善安全連線的隱私權和可靠性。如需使用 OCSP 堆疊之優點的詳細資訊，請參閱 ???。

> **Note**
>
> 此功能不適用於 AWS GovCloud (US) Regions。

在本主題中：

- 什麼是 OCSP？
- OCSP 堆疊的運作方式
- 在 中啟用伺服器憑證 OCSP AWS IoT Core
- 在 中為私有端點設定伺服器憑證 OCSP AWS IoT Core
- 在 中使用伺服器憑證 OCSP 堆疊的重要注意事項 AWS IoT Core
- 對 中的伺服器憑證 OCSP 堆疊進行故障診斷 AWS IoT Core

什麼是 OCSP？

線上憑證狀態通訊協定 (OCSP) 可協助為 Transport Layer Security (TLS) 交握提供伺服器憑證的撤銷狀態。

重要概念

下列重要概念提供有關線上憑證狀態通訊協定 (OCSP) 的詳細資訊。

OCSP

OCSP 用於在 Transport Layer Security (TLS) 交握期間檢查憑證撤銷狀態。OCSP 允許即時驗證憑證。這會確認憑證自發出以來尚未撤銷或過期。相較於傳統憑證撤銷清單 (CRLs)，OCSP 也更具可擴展性。OCSP 回應較小且可有效率地產生，使其更適合大規模的私有金鑰基礎設施 (PKIs)。

OCSP 回應程式

OCSP 回應程式（也稱為 OCSP 伺服器）會接收並回應來自用戶端的 OCSP 請求，以驗證憑證的撤銷狀態。

用戶端 OCSP

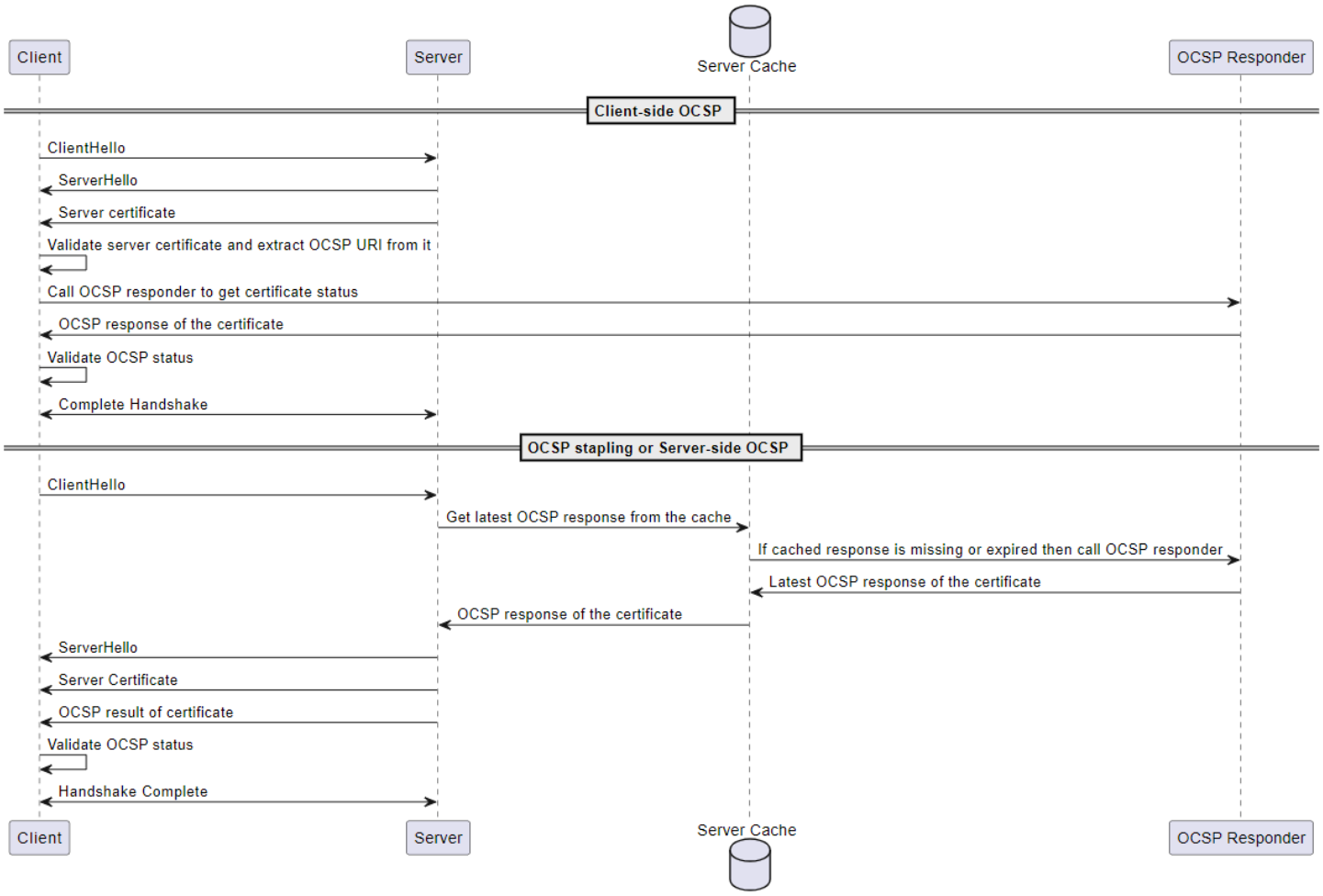
在用戶端 OCSP 中，用戶端會使用 OCSP 聯絡 OCSP 回應者，在 TLS 交握期間檢查憑證的撤銷狀態。

伺服器端 OCSP

在伺服器端 OCSP（也稱為 OCSP 堆疊）中，伺服器已啟用（而不是用戶端），以向 OCSP 回應程式提出請求。伺服器會將 OCSP 回應綁定到憑證，並在 TLS 交握期間將其傳回給用戶端。

OCSP 圖表

下圖說明用戶端 OCSP 和伺服器端 OCSP 的運作方式。



用戶端 OCSP

1. 用戶端會`ClientHello`傳送訊息，以啟動與伺服器的 TLS 交握。
2. 伺服器會收到訊息，並以`ServerHello`訊息回應。伺服器也會將伺服器憑證傳送至用戶端。
3. 用戶端會驗證伺服器憑證，並從中擷取 OCSP URI。
4. 用戶端會將憑證撤銷檢查請求傳送至 OCSP 回應程式。
5. OCSP 回應程式會傳送 OCSP 回應。
6. 用戶端會從 OCSP 回應驗證憑證狀態。
7. TLS 交握已完成。

伺服器端 OCSP

1. 用戶端會`ClientHello`傳送訊息，以啟動與伺服器的 TLS 交握。

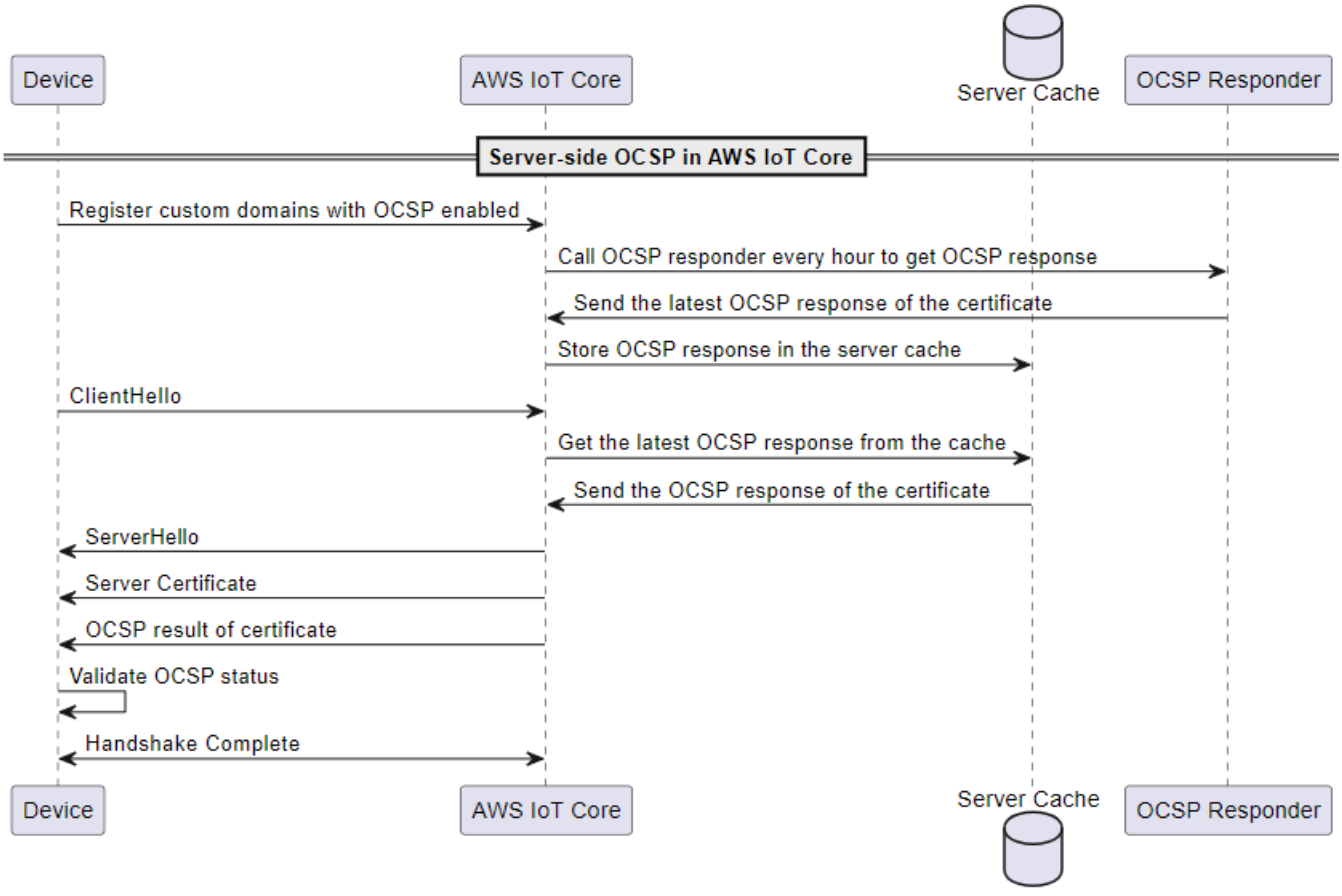
2. 伺服器會收到訊息並取得最新的快取 OCSP 回應。如果快取的回應遺失或過期，伺服器會呼叫 OCSP 回應程式以取得憑證狀態。
3. OCSP 回應程式會將 OCSP 回應傳送至伺服器。
4. 伺服器傳送訊息`ServerHello`。伺服器也會將伺服器憑證和憑證狀態傳送至用戶端。
5. 用戶端會驗證 OCSP 憑證狀態。
6. TLS 交握已完成。

OCSP 堆疊的運作方式

在用戶端和伺服器之間的 TLS 交握期間，會使用 OCSP 堆疊來檢查伺服器憑證撤銷狀態。伺服器向 OCSP 回應程式提出 OCSP 請求，並將 OCSP 回應與傳回給用戶端的憑證整合在一起。透過讓伺服器向 OCSP 回應程式提出請求，可以快取回應，然後對許多用戶端多次使用回應。

OCSP 堆疊如何在 中運作 AWS IoT Core

下圖顯示伺服器端 OCSP 堆疊如何在 中運作 AWS IoT Core。



1. 裝置需要向啟用 OCSP 堆疊的自訂網域註冊。

2. AWS IoT Core 每小時呼叫 OCSP 回應程式以取得憑證狀態。
3. OCSP 回應程式會接收請求、傳送最新的 OCSP 回應，以及存放快取的 OCSP 回應。
4. 裝置會`ClientHello`傳送訊息以啟動 TLS 交握 AWS IoT Core。
5. AWS IoT Core 從伺服器快取取得最新的 OCSP 回應，這會回應憑證的 OCSP 回應。
6. 伺服器會`ServerHello`傳送訊息至裝置。伺服器也會將伺服器憑證和憑證狀態傳送至用戶端。
7. 裝置會驗證 OCSP 憑證狀態。
8. TLS 交握已完成。

相較於用戶端 OCSP 檢查，使用 OCSP 堆疊的優勢

使用伺服器憑證 OCSP 堆疊的一些優點包括：

改善隱私權

如果沒有 OCSP 堆疊，用戶端的裝置可能會向第三方 OCSP 回應者公開資訊，進而可能危及使用者隱私權。OCSP 堆疊可讓伺服器取得 OCSP 回應並將其直接交付給用戶端，以緩解此問題。

改善可靠性

OCSP 堆疊可以提高安全連線的可靠性，因為它可以降低 OCSP 伺服器中斷的風險。當 OCSP 回應為綁定時，伺服器會將最新的回應與憑證一起包含在內。這可讓用戶端存取撤銷狀態，即使 OCSP 回應程式暫時無法使用。OCSP 堆疊有助於緩解這些問題，因為伺服器會定期擷取 OCSP 回應，並在 TLS 交握中包含快取的回應。這可減少對 OCSP 回應者即時可用性的依賴。

減少伺服器負載

OCSP 堆疊會將回應 OCSP 請求的負擔從 OCSP 回應程式卸載至伺服器。這有助於更平均地分配負載，使憑證驗證程序更有效率且更具可擴展性。

降低延遲

OCSP 堆疊可減少在 TLS 交握期間檢查憑證撤銷狀態的相關延遲。伺服器傳送請求，並在交握期間將 OCSP 回應與伺服器憑證連接，而不是用戶端單獨查詢 OCSP 伺服器。

在 中啟用伺服器憑證 OCSP AWS IoT Core

若要在 中啟用伺服器憑證 OCSP 堆疊 AWS IoT Core，請為自訂網域建立網域組態，或更新現有的自訂網域組態。如需使用自訂網域建立網域組態的一般資訊，請參閱 ???。

使用以下指示，使用 AWS 管理主控台 或 啟用 OCSP 伺服器堆疊 AWS CLI。

主控台

若要使用 AWS IoT 主控台啟用伺服器憑證 OCSP 堆疊：

1. 在導覽功能表中，選擇設定，然後選擇建立網域組態，或選擇自訂網域的現有網域組態。
2. 如果您選擇在上一個步驟中建立新的網域組態，您會看到建立網域組態頁面。在網域組態屬性區段中，選擇自訂網域。輸入資訊以建立網域組態。

   如果您選擇更新自訂網域的現有網域組態，您會看到網域組態詳細資訊頁面。選擇編輯。
3. 若要啟用 OCSP 伺服器堆疊，請在伺服器憑證組態子區段中選擇啟用伺服器憑證 OCSP 堆疊。
4. 選擇建立網域組態或更新網域組態。

AWS CLI

若要使用 啟用伺服器憑證 OCSP 堆疊 AWS CLI：

1. 如果您為自訂網域建立新的網域組態，啟用 OCSP 伺服器堆疊的命令可能如下所示：

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

2. 如果您更新自訂網域的現有網域組態，啟用 OCSP 伺服器堆疊的命令可能如下所示：

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

如需詳細資訊，請參閱 AWS IoT API 參考中的 CreateDomainConfiguration 和 UpdateDomainConfiguration。

在 中為私有端點設定伺服器憑證 OCSP AWS IoT Core

適用於私有端點的 OCSP 可讓您將 Amazon Virtual Private Cloud (Amazon VPC) 中的私有 OCSP 資源用於 AWS IoT Core 操作。程序涉及設定做為 OCSP 回應程式的 Lambda 函數。Lambda 函數可能會使用您的私有 OCSP 資源來製作 AWS IoT Core 將使用的 OCSP 回應。

Lambda 函式

為私有端點設定伺服器 OCSP 之前，請建立 Lambda 函數，做為評論請求 (RFC) 6960 相容線上憑證狀態通訊協定 (OCSP) 回應程式，以支援基本 OCSP 回應。Lambda 函數接受區分編碼規則 (DER) 格式的 OCSP 請求 base64 編碼。Lambda 函數的回應也是 DER 格式的 base64 編碼 OCSP 回應。回應大小不得超過 4 KB (KiB)。Lambda 函數必須與網域組態位於相同 AWS 帳戶 和 AWS 區域 。以下是 Lambda 函數的範例。

Lambda 函數範例

JavaScript

```
import * as pkijs from 'pkijs';
console.log('Loading function');

export const handler = async (event, context) => {
    const requestBytes = decodeBase64(event);
    const ocspRequest = pkijs.OCSPRequest.fromBER(requestBytes);

    console.log("Here is a better look at the OCSP request");
    console.log(ocspRequest.toJSON());

    const ocspResponse = getOcspResponse();

    console.log("Here is a better look at the OCSP response");
    console.log(ocspResponse.toJSON());

   const responseBytes = ocspResponse.toSchema().toBER();
   return encodeBase64(responseBytes);
};

function getOcspResponse() {
    const responseString = "MIIC/
woBAKCCAvgwggL0BgkrBgEFBQcwAQEEggLlMIIC4TCByqFkMGIxJzAlBgNVBAoMHlJpY2hhcmQncyBEaXNjb3VudCBMY
p5w7WOtPjp3otNtVgIBAYAAGA8yMDI0MDQyMzE4NTMyNVowDQYJKoZIhvcNAQELBQADggIBAJFRyjDAHfazNejo704Ra
+s82R1spDarr3k7Pzkod9jJhwsZ2YgushlS4Npfe4lHCdwFyZR75WXrW55aXFddy03KLz01ZLNYyxkleW3f5dgrUcRU3
DEBiyS7ZsyhKo6igWU/SY7YMSKgwBvFsqSDcOa/hRYQkxWKWJ19gcz8CIkWN7NvfIxCs6VrAdzEJwmE7y3v
```

```
+jdfhxW9JmI4xStE4K0tAR9vVOOfKs7NvxXj7oc9pCSG60xl96kaEE6PaY1YsfNTsKQ7pyCJ0s7/2q
+ieZ4AtNyzw1XBadPzPJNv6E0LvI24yQZqN5wACvtut5prMMRxAHbOy
+abLZR58wloFSELtGJ7UD96LFv1GgtC5s
+2QlzPc4bEEof7Lo1EISt3j2ibNch8LxhqTQ4ufrbhsMkpSOTFYEJVMJF6aKj/OGXBUUqgc0Jx6jjJXNQd
+l5KCY9pQFeb/wVUYC6mYqZOkNNMMJxPbHHbFnqb68yO+g5BE9011N44YXoPVJYoXxBLFX+OpRu9cqPkT9/
vlkKd+SYXQknwZ81agKzhf1HsBKabtJwNVMlBKaI8g5UGa7Bxi6ewH3ezdWiERRUK7F56OM53wto/";
    const responseBytes = decodeBase64(responseString);
    return pkijs.OCSPResponse.fromBER(responseBytes);
}

function decodeBase64(input) {
    const binaryString = atob(input);

    const byteArray = new Uint8Array(binaryString.length);
    for (var i = 0; i < binaryString.length; i++) {
        byteArray[i] = binaryString.charCodeAt(i);
    }

    return byteArray.buffer;
}

function encodeBase64(buffer) {
    var binary = '';
    const bytes = new Uint8Array( buffer );
    const len = bytes.byteLength;

    for (var i = 0; i < len; i++) {
        binary += String.fromCharCode( bytes[ i ] );
    }

    return btoa(binary);
}
```

Java

```
package com.example.ocsp.responder;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import org.bouncycastle.cert.ocsp.OCSPReq;
import org.bouncycastle.cert.ocsp.OCSPResp;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.OutputStream;
import java.util.Base64;

public class LambdaResponderApplication implements RequestHandler<String, String> {
    @Override
    public String handleRequest(final String input, final Context context) {
        LambdaLogger logger = context.getLogger();

        byte[] decodedInput = Base64.getDecoder().decode(input);

        OCSPReq req;
        try {
            req = new OCSPReq(decodedInput);
        } catch (IOException e) {
            logger.log("Got an IOException creating the OCSP request: " +
 e.getMessage());
            throw new RuntimeException(e);
        }

        try {
            OCSPResp response = businessLogic.getMyResponse();
            String toReturn =
 Base64.getEncoder().encodeToString(response.getEncoded());
            return toReturn;
        } catch (Exception e) {
            logger.log("Got an exception creating the response: " + e.getMessage());
            return "";
        }
    }
}
```

授權 AWS IoT 叫用您的 Lambda 函數

在使用 Lambda OCSP 回應程式建立網域組態的過程中，您必須授予在建立函數之後叫用 Lambda 函數的 AWS IoT 許可。若要授與許可，您可以使用 add-permission CLI 命令。

使用 將許可授予 Lambda 函數 AWS CLI

1. 在插入您的值之後，輸入以下命令。請注意，`statement-id` 值必須是唯一的。*Id-1234* 將 取代為您擁有的確切值，否則您可能會發生錯誤`ResourceConflictException`。

```
aws lambda add-permission  \
```

```
--function-name "ocsp-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn arn:aws:iot:us-east-1:123456789012:domainconfiguration/<domain-config-name>/*
--source-account 123456789012
```

IoT 網域組態 ARNs 將遵循下列模式。建立時間之前不會知道服務產生的尾碼，因此您必須以 取代尾碼*。您可以在建立網域組態並知道確切的 ARN 之後更新許可。

arn:*aws*:iot:*use-east-1:123456789012*:domainconfiguration/*domain-config-name/service-generated-suffix*

2. 如果命令成功，它會傳回許可陳述式，例如此範例。您可以繼續前往下一節，為私有端點設定 OCSP 堆疊。

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-east-1:123456789012:function:ocsp-
function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:iot:us-
east-1:123456789012:domainconfiguration/domain-config-name/*\"}}}"
}
```

如果命令未成功，它會傳回錯誤，例如此範例。您必須先檢閱並更正錯誤，然後才能繼續進行。

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:us-east-1:123456789012:function:ocsp-function
```

設定私有端點的伺服器 OCSP 堆疊

主控台

若要使用 AWS IoT 主控台設定伺服器憑證 OCSP 堆疊：

1. 從導覽功能表中，選擇設定，然後選擇建立網域組態，或選擇自訂網域的現有網域組態。

2. 如果您選擇在上一個步驟中建立新的網域組態，您會看到建立網域組態頁面。在網域組態屬性區段中，選擇自訂網域。輸入資訊以建立網域組態。

   如果您選擇更新自訂網域的現有網域組態，您會看到網域組態詳細資訊頁面。選擇編輯。

3. 若要啟用 OCSP 伺服器堆疊，請在伺服器憑證組態子區段中選擇啟用伺服器憑證 OCSP 堆疊。
4. 選擇建立網域組態或更新網域組態。

AWS CLI

若要使用 設定伺服器憑證 OCSP 堆疊 AWS CLI：

1. 如果您為自訂網域建立新的網域組態，為私有端點設定伺服器憑證 OCSP 的命令可能如下所示：

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

2. 如果您更新自訂網域的現有網域組態，為私有端點設定伺服器憑證 OCSP 的命令可能如下所示：

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

enableOCSPCheck

這是布林值，指出伺服器 OCSP 堆疊檢查是否已啟用。若要啟用伺服器憑證 OCSP 堆疊，此值必須是 true。

ocspAuthorizedResponderArn

這是存放在 (ACM) 中 X.509 憑證的 Amazon Resource Name AWS Certificate Manager (ARN) 字串值。如果提供， AWS IoT Core 將使用此憑證來驗證收到的 OCSP 回應的簽章。如果未提供， AWS IoT Core 將使用發行憑證來驗證回應。憑證必須與網域組態位於相同 AWS 帳戶 和 AWS 區域 。如需如何註冊授權回應者憑證的詳細資訊，請參閱將憑證匯入 AWS Certificate Manager。

ocspLambdaArn

這是 Lambda 函數的 Amazon Resource Name (ARN) 字串值，可做為符合評論請求 (RFC) 6960 標準 (OCSP) 回應程式，支援基本 OCSP 回應。Lambda 函數接受使用 DER 格式編碼之 OCSP 請求的 base64 編碼。Lambda 函數的回應也是 DER 格式的 base64 編碼 OCSP 回應。回應大小不得超過 4 KB (KiB)。Lambda 函數必須與網域組態位於相同 AWS 帳戶 和 AWS 區域 。

如需詳細資訊，請參閱 AWS IoT API 參考中的 CreateDomainConfiguration 和 UpdateDomainConfiguration。

在 中使用伺服器憑證 OCSP 堆疊的重要注意事項 AWS IoT Core

當您在 中使用伺服器憑證 OCSP 時 AWS IoT Core，請記住下列事項：

1. AWS IoT Core 僅支援可透過公有 IPv4 地址連線的 OCSP 回應程式。
2. 中的 OCSP 堆疊功能 AWS IoT Core 不支援授權的回應程式。所有 OCSP 回應都必須由簽署憑證的 CA 簽署，而且 CA 必須是自訂網域憑證鏈的一部分。
3. 中的 OCSP 堆疊功能 AWS IoT Core 不支援使用自我簽署憑證建立的自訂網域。
4. AWS IoT Core 每小時呼叫 OCSP 回應程式並快取回應。如果呼叫回應者失敗， AWS IoT Core 會將最新的有效回應綁定在一起。
5. 如果 `nextUpdateTime` 不再有效， AWS IoT Core 將從快取中移除回應，且 TLS 交握不會包含 OCSP 回應資料，直到下一次成功呼叫 OCSP 回應程式為止。當快取的回應在伺服器從 OCSP 回應程式取得有效的回應之前已過期時，就會發生這種情況。的值`nextUpdateTime`表示 OCSP 回應在此之前都是有效的。如需 `nextUpdateTime` 的相關資訊，請參閱 ???。
6. 有時， AWS IoT Core 無法接收 OCSP 回應或移除現有的 OCSP 回應，因為它已過期。如果發生這類情況， AWS IoT Core 將繼續使用自訂網域提供的伺服器憑證，而無需 OCSP 回應。
7. OCSP 回應的大小不能超過 4 KiB。

對 中的伺服器憑證 OCSP 堆疊進行故障診斷 AWS IoT Core

AWS IoT Core 會向 CloudWatch 發出`RetrieveOCSPStapleData.Success`指標和`RetrieveOCSPStapleData`日誌項目。指標和日誌項目有助於偵測與擷取 OCSP 回應相關的問題。如需詳細資訊，請參閱???及???。

連線至 AWS IoT FIPS 端點

AWS IoT 提供支援聯邦資訊處理標準 (FIPS) 140-2 的端點。FIPS 合規端點與標準 AWS 端點不同。若要以 FIPS 合規方式與 AWS IoT 互動，您必須使用下述端點搭配 FIPS 合規用戶端。 AWS IoT 主控台不符合 FIPS 規範。

下列各節說明如何使用 REST API、 SDK 或 存取 FIPS 相容 AWS IoT 端點 AWS CLI。

主題

- AWS IoT Core：控制平面端點
- AWS IoT Core：資料平面端點
- AWS IoT Core - 登入資料 提供者端點
- AWS IoT Device Management：任務資料端點
- AWS IoT Device Management：Fleet Hub 端點
- AWS IoT Device Management：安全通道端點
- AWS IoT Device Management - 受管整合端點

AWS IoT Core：控制平面端點

支援 AWS IoT 操作及其相關 CLI 命令的 FIPS 合規 AWS IoT Core：控制平面端點，都列於依服務列出的 FIPS 端點。在依服務列出的 FIPS 端點中，尋找 AWS IoT Core：控制平面服務，並查詢適合 AWS 區域的端點。

若要在存取AWS IoT操作時使用 FIPS 相容端點，請使用 AWS SDK 或 REST API 搭配適合您的端點 AWS 區域。

若要在執行 aws iot CLI 命令時使用 FIPS 合規端點，請將具有適合 AWS 區域 端點的 --endpoint 參數新增至命令。

AWS IoT Core：資料平面端點

FIPS 合規 AWS IoT Core：資料平面端點列於依服務列出的 FIPS 端點。在依服務列出的 FIPS 端點中，尋找 AWS IoT Core：資料平面服務，並查詢適合 AWS 區域的端點。

您可以使用 AWS IoT 裝置 SDK，並將端點提供給 SDK 的連線函數，以取代您帳戶的預設 AWS IoT Core 資料 平面端點，藉此將 FIPS 相容端點用於 AWS 區域 與 FIPS 相容用戶端。連線函數專屬於 AWS IoT 裝置 SDK。如需連線函數的範例，請參閱適用於 Python 的 AWS IoT 裝置 SDK 中的連線函數。

> **Note**
>
> AWS IoT 不支援 AWS 帳戶符合 FIPS 的特定 AWS IoT Core 資料 平面端點。無法使用伺服器名稱指示 (SNI) 中需要 AWS 帳戶特定端點的服務功能。FIPS 合規 AWS IoT Core- 資料平面端點無法支援「多帳戶註冊憑證」、「自訂網域」、「自訂授權者」和「可設定的端點」(包括支援的 TLS 政策)。

AWS IoT Core - 登入資料 提供者端點

符合 FIPS 的AWS IoT Core - 登入資料 提供者端點會列在依服務分類的 FIPS 端點中。在 FIPS Endpoints by Service 中，尋找AWS IoT Core - 憑證 提供者服務，並查詢 的端點 AWS 區域。

> **Note**
>
> AWS IoT 不支援 AWS 帳戶特定AWS IoT Core - 符合 FIPS 規範的 登入資料 提供者端點。無法使用伺服器名稱指示 (SNI) 中需要 AWS 帳戶特定端點的服務功能。FIPS 合規AWS IoT Core - 登入資料 提供者端點不支援多帳戶註冊憑證、自訂網域、自訂授權方和可設定的端點（包括支援的 TLS 政策)。

AWS IoT Device Management：任務資料端點

FIPS 合規 AWS IoT Device Management：任務資料端點列於依服務列出的 FIPS 端點。在依服務列出的 FIPS 端點中，尋找 AWS IoT Device Management：任務資料服務，並查詢適合 AWS 區域的端點。

若要在執行 aws iot-jobs-data CLI 命令,時，使用 FIPS 合規AWS IoT Device Management - 任務資料端點，請將具有適合 AWS 區域 端點的 --endpoint 參數新增至命令。您也可以將 REST API 與此端點搭配起來使用。

建議使用 Data-ATS而非 iot:Jobs。 iot:Data-ATS支援雙堆疊端點 (IPv4 和 IPv6)，同時僅iot:Jobs支援 IPv4。

您可以使用 AWS IoT 裝置 SDK，並將端點提供給 SDK 的連線函數，以取代您帳戶的預設AWS IoT Device Management 任務 資料端點，將 FIPS 相容端點用於 AWS 區域 與 FIPS 相容的用戶端。連線函數是 AWS IoT 裝置 SDK 的專用函數。如需連線函數的範例，請參閱適用於 Python 的 AWS IoT 裝置 SDK 中的連線函數。

AWS IoT Device Management：Fleet Hub 端點

與 FIPS 相容的AWS IoT Device Management - 與 Fleet Hub for AWS IoT Device Management CLI 命令搭配使用的 Fleet Hub 端點會列在 FIPS Endpoints by Service 中。 https://docs.aws.amazon.com/cli/latest/reference/iotfleethub/index.html在依服務列出的 FIPS 端點中，尋找 AWS IoT Device Management：Fleet Hub 服務，並查詢適合 AWS 區域的端點。

若要在執行 aws iotfleethubCLI 命令時使用符合 FIPS 標準的 AWS IoT Device Management Fleet Hub 端點，請將 --endpoint 參數與適用於 的端點新增至 AWS 區域 命令。您也可以將 REST API 與此端點搭配起來使用。

AWS IoT Device Management：安全通道端點

適用於 AWS IoT 安全通道 API的 FIPS 合規 AWS IoT Device Management：安全通道端點以及對應的 CLI 命令，都列於依服務列出的 FIPS 端點。在依服務列出的 FIPS 端點中，尋找 AWS IoT Device Management：安全通道服務，並查詢適合 AWS 區域的端點。

若要在執行 aws iotsecuretunneling CLI 命令時，使用 AWS IoT Device Management FIPS 合規 - 安全通道端點，請將具有適合 AWS 區域 端點的 --endpoint 參數新增至命令。您也可以將 REST API 與此端點搭配起來使用。

AWS IoT Device Management - 受管整合端點

支援受管整合操作及其相關 AWS CLI 命令的 FIPS 相容控制平面端點，會列在 FIPS Endpoints by Service 中。在 FIPS Endpoints by Service 中，尋找 AWS IoT Device Management 受管整合服務，並查詢 的端點 AWS 區域。

若要在存取受管整合操作時使用 FIPS 相容端點，請使用 AWS SDK 或 REST API 搭配適合您的端點 AWS 區域。

若要在執行受管整合 CLI 命令時使用 FIPS 相容端點，請將 --endpoint 參數與適用於 的端點新增至 AWS 區域 命令。

使用 管理裝置 AWS IoT

AWS IoT 提供的登錄檔可協助您管理物件。物件是特定裝置或邏輯實體的表示法。它可以是實體裝置或感應器 (例如燈泡或牆上的開關)。它也可以是邏輯實體，例如未連線至 但與其他裝置 （ 例如具有引擎感應器或控制面板的汽車 ） AWS IoT 相關的應用程式執行個體或實體實體。

物件的相關資訊會以 JSON 資料的形式存放於登錄檔中。以下為物件的範例：

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

物件以名稱識別。物件可以具有屬性，亦即名稱/值對，可讓您用來存放關於物件的資訊，例如其序號或製造商。

典型的裝置使用案例包括使用物件名稱做為預設 MQTT 用戶端 ID。雖然我們不會強制對物件登錄檔名稱及其使用的 MQTT 用戶端 ID、憑證或影子狀態進行映射，但仍建議您選擇物件名稱，並用其作為登錄檔與 Device Shadow 服務的 MQTT 用戶端 ID。這能為您的 IoT 裝置機群帶來秩序與便利，同時仍能保有基礎裝置憑證模型或影子的彈性。

您不必在登錄檔中建立物件，即可將裝置連線至 AWS IoT。將物件新增至登錄檔，可更輕鬆地管理及搜尋裝置。

物件登錄檔資料 （ 包括屬性、物件類型和群組成員資格 ） 也可以在規則引擎中動態擷取，以用於訊息處理和路由。如需詳細資訊，請參閱[get_registry_data]。

使用登錄檔管理物件

您可以使用 AWS IoT 主控台、 AWS IoT API 或 AWS CLI 與登錄檔互動。下列章節說明如何利用 CLI 使用登錄檔。

命名您的物件物件時：

- 請勿在物件名稱中使用個人識別資訊。物件名稱可以出現在未加密的通訊和報告中。

主題

- 建立物件
- 列出物件
- 說明物件
- 更新物件
- 刪除物件
- 將主體連接至物件
- 列出與委託人相關聯的物件
- 列出與物件相關聯的主體
- 列出與委託人 V2 相關聯的物件
- 列出與物件 V2 相關聯的主體
- 將主體與物件分離

建立物件

下列命令顯示如何使用來自 CLI 的 AWS IoT CreateThing命令來建立物件。建立物件之後，就無法變更物件的名稱。若要變更物件的名稱，請建立新的物件、提供新名稱，然後刪除舊物件。

```
$ aws iot create-thing \
    --thing-type-name "MyLightBulb" \
    --attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

該 CreateThing 命令會顯示新物件的名稱和 Amazon Resource Name (ARN)：

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

> **Note**
>
> 我們不建議在物件名稱中使用個人識別資訊。

如需詳細資訊，請參閱《 AWS CLI 命令參考》中的 create-thing。

列出物件

您可以使用 ListThings 命令列出您帳戶中的所有物件：

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
            "thingName": "MyWallSwitch"
        }
    ]
}
```

您可以使用 ListThings 命令搜尋特定物件類型的所有物件：

```
$  aws iot list-things --thing-type-name "LightBulb"
```

```
{
    "things": [
        {
```

```
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

您可以使用 ListThings 命令搜尋具特定值之屬性的所有物件。此命令最多可搜尋三個屬性。

```
$  aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```
{
    "things": [
        {
            "thingTypeName": "StopLight",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3,
            "thingName": "MyLightBulb"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
```

```
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

如需詳細資訊，請參閱《 AWS CLI 命令參考》中的 list-things。

說明物件

您可使用 DescribeThing 命令顯示物件的相關詳細資訊：

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "version": 3,
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "StopLight",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

您也可以使用內嵌函數 在規則引擎內存取此 APIget_registry_data()。您可以使用此函數，直接在 AWS IoT 規則內呼叫 和 DescribeThing ListThingGroupsForThing APIs，根據裝置登錄檔資料啟用即時訊息處理和路由，以動態存取和利用物件登錄檔資訊 （包括屬性、物件類型和群組成員資格）。如需詳細資訊，請參閱get_registry_data。

如需詳細資訊，請參閱 AWS CLI 命令參考中的 describe-thing。

更新物件

您可以使用 UpdateThing 命令更新物件。此命令只會更新物件的屬性。您不能改變物件的名字。若要變更物件的名稱，請建立新的物件、提供新名稱，然後刪除舊物件。

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

UpdateThing 命令不會產生輸出。您可以使用 DescribeThing 命令來查看結果：

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

如需詳細資訊，請參閱《 AWS CLI 命令參考》中的 update-thing。

刪除物件

您可以使用 DeleteThing 命令來刪除物件：

```
$ aws iot delete-thing --thing-name "MyThing"
```

如果刪除成功或您指定的物件不存在，則此命令會成功傳回且不含錯誤。

如需詳細資訊，請參閱《 AWS CLI 命令參考》中的 delete-thing。

將主體連接至物件

實體裝置可以使用委託人與之通訊 AWS IoT。委託人可以是 X.509 憑證或 Amazon Cognito ID。您可以透過執行 attach-thing-principal 命令，將憑證或 Amazon Cognito ID 與代表裝置的登錄檔中的物件建立關聯。

若要將憑證或 Amazon Cognito ID 連接至您的物件，請使用 attach-thing-principal 命令：

```
$ aws iot attach-thing-principal \
```

```
    --thing-name "MyLightBulb1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

若要使用附件類型（專屬附件或非專屬附件）將憑證連接至您的物件，請使用 attach-thing-principal 命令，並在 --thing-principal-type 欄位中指定類型。專屬附件表示您的 IoT 物件是唯一連接到憑證的物件，而且此憑證無法與任何其他物件建立關聯。非專屬附件表示您的 IoT 物件已連接至憑證，而且此憑證可以與其他物件建立關聯。如需詳細資訊，請參閱???。

> **Note**
>
> 對於???此功能，您只能使用 X.509 憑證做為委託人。

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb2" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --thing-principal-type "EXCLUSIVE_THING"
```

如果連接成功，AttachThingPrincipal命令不會產生任何輸出。若要描述附件，請使用 list-thing-principals-v2 CLI 命令。

如需詳細資訊，請參閱 AWS IoT Core API 參考中的 AttachThingPrincipal。

列出與委託人相關聯的物件

若要列出與指定委託人相關聯的物件，請執行 list-principal-things命令。請注意，此命令不會列出物件與憑證之間的連接類型。若要列出附件類型，請使用 list-principal-things-v2命令。如需詳細資訊，請參閱???。

```
$ aws iot list-principal-things \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

輸出可能如下所示。

```
{
    "things": [
        "MyLightBulb1",
        "MyLightBulb2"
    ]
}
```

如需詳細資訊，請參閱 AWS IoT Core API 參考中的 ListPrincipalThings。

列出與物件相關聯的主體

若要列出與指定物件相關聯的委託人，請執行 list-thing-principals命令。請注意，此命令不會列出物件與憑證之間的連接類型。若要列出附件類型，請使用 list-thing-principals-v2命令。如需詳細資訊，請參閱???。

```
$ aws iot list-thing-principals \
    --thing-name "MyLightBulb1"
```

輸出可能如下所示。

```
{
    "principals": [
         "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8",
         "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
    ]
}
```

如需詳細資訊，請參閱 AWS IoT Core API 參考中的 ListThingPrincipals。

列出與委託人 V2 相關聯的物件

若要列出與指定憑證相關聯的物件，以及附件類型，請執行 list-principal-things-v2命令。連接類型是指憑證如何連接到物件。

```
$ aws iot list-principal-things-v2 \
```

```
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

輸出可能如下所示。

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        },
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_2"
        }

    ]
}
```

如需詳細資訊，請參閱 AWS IoT Core API 參考中的 ListPrincipalThingsV2。

列出與物件 V2 相關聯的主體

若要列出與指定物件相關聯的憑證，以及附件類型，請執行 list-thing-principals-V2命令。連接類型是指憑證如何連接到物件。

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

輸出可能如下所示。

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        },
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
```

```
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
        }
    ]
}
```

如需詳細資訊，請參閱 AWS IoT Core API 參考中的 ListThingsPrincipalV2。

將主體與物件分離

您可以使用 DetachThingPrincipal 命令從物件分離出憑證：

```
$ aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

DetachThingPrincipal 命令不會產生任何輸出。

如需詳細資訊，請參閱 API AWS IoT Core 參考中的 detach-thing-principal。

物件類型

物件類型能讓您存放與相同物件類型關聯之所有物件通用的描述和組態資訊。如此可簡化登錄檔中物件的管理。例如，您可以定義 LightBulb 物件類型。所有與 LightBulb 物件類型關聯的物件具有同一組屬性：序號、製造商、瓦特數。當您建立 LightBulb 類型的物件 (或是變更現有物件的類型為 LightBulb)，可以指定在 LightBulb 物件類型中所定義的每個屬性之值。

雖然物件類型為選用，但使用時可讓您探索物件更輕鬆。

- 有物件類型的物件最多可具有 50 個屬性。
- 無物件類型的物件最多可具有三個屬性。
- 一個物件只能與一個物件類型建立關聯。
- 在您的帳戶中，可以建立的物件類型數量不限。

物件類型建立之後，您就無法變更其名稱。您可以隨時棄用物件類型，避免新的物件與其建立關聯。您也可以刪除未和任何物件關聯的物件類型。

主題：

- 建立物件類型
- 列出物件類型
- 描述物件類型
- 將物件類型與物件建立關聯
- 更新物件類型
- 棄用物件類型
- 刪除物件類型

建立物件類型

您可以使用 CreateThingType 命令來建立物件類型：

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

CreateThingType 命令的回應會包含物件類型及其 ARN：

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

列出物件類型

您可以使用 ListThingTypes 命令列出物件類型：

```
$ aws iot list-thing-types
```

ListThingTypes 命令會傳回 中定義的物件類型清單 AWS 帳戶：

```
{
```

```
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
                "thingTypeDescription": "light bulb type"
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

描述物件類型

您可以使用 DescribeThingType 命令獲得關於物件類型的資訊：

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

DescribeThingType 命令會傳回關於指定類型的資訊：

```
{
    "thingTypeProperties": {
        "searchableAttributes": [
            "model",
            "wattage"
        ],
        "thingTypeDescription": "light bulb type"
    },
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
    "thingTypeName": "LightBulb",
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1544466338.399
    }
}
```

將物件類型與物件建立關聯

在建立物件時，您可以使用 CreateThing 命令指定物件類型：

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您可以隨時使用 UpdateThing 命令變更與物件關聯的物件類型：

```
$ aws iot update-thing --thing-name "MyLightBulb"
               --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您也可以使用 UpdateThing 命令取消物件與物件類型的關聯。

更新物件類型

您可以在建立物件時使用 UpdateThingType命令來更新物件類型：

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您可以隨時使用 UpdateThing 命令變更與物件關聯的物件類型：

```
$ aws iot update-thing --thing-name "MyLightBulb"
               --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您也可以使用 UpdateThing 命令取消物件與物件類型的關聯。

棄用物件類型

物件類型不可變。物件類型定義之後就無法變更。不過，您可以棄用物件類型，避免使用者將其與任何新物件建立關聯。所有與該物件類型關聯的現有物件都不會變更。

若要棄用物件類型，請使用 DeprecateThingType 命令：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

您可以使用 DescribeThingType 命令來查看結果：

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
    },
    "thingTypeMetadata": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "deprecationDate": 1468446026349
    }
}
```

棄用物件類型是可逆的操作。您可以使用 `--undo-deprecate` 旗標以及 DeprecateThingType CLI 命令來取消棄用：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

您可以使用 DescribeThingType CLI 命令查看結果：

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/StopLight",
    "thingTypeId": "12345678abcdefgh12345678ijklmnop12345678"
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type"
```

```
    },
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1468425854308,
    }
}
```

刪除物件類型

物件類型只有在棄用後才能刪除。若要刪除物件類型，請使用 DeleteThingType 命令：

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

> **Note**
>
> 刪除物件類型之前，請等待五分鐘後將其棄用。

靜態物件群組

物件群組可讓您將多個物件分類成群組以方便同時管理。靜態物件群組包含一組使用主控台、CLI 或 API 來管理的物件。另一方面，動態物件群組會包含與指定查詢相符的物件。靜態物件群組還可包含其他靜態物件群組，您可建置一個群組階層。您可以將政策連接至父群組，而此政策會被其子群組以及父、子群組內的所有物件繼承。這可讓您輕易控制大量物件的許可。

> **Note**
>
> 物件群組政策不允許存取 AWS IoT Greengrass 資料平面操作。若要允許物件存取 AWS IoT Greengrass 資料平面操作，請將 許可新增至您連接至物件憑證 AWS IoT 的政策。如需詳細資訊，請參閱《AWS IoT Greengrass 開發人員指南》中的裝置身分驗證和授權。

您可以對靜態物件群組執行下列操作：

- 建立、描述或刪除群組。
- 新增物件至一個多個群組。
- 從群組移除物件。
- 列出您建立的群組。

- 列出群組的所有子群組 (直系與旁系的後代)。
- 列出群組裡的物件，包括其子群組裡的所有物件。
- 列出群組的所有上階群組 (直系與旁系的父群組)。
- 新增、刪除或更新群組的屬性。(屬性是您可用於存放群組相關資訊的名稱/值對。)
- 讓政策與群組連接或分離。
- 列出連接至群組的政策。
- 列出物件所繼承的政策 (透過連接至其群組或其父群組的政策)。
- 設定群組中物件的記錄選項。請參閱 設定 AWS IoT 記錄。
- 建立傳送至群組及其子群組中所有物件並執行的任務。請參閱 AWS IoT 任務。

> **Note**
>
> 當物件連接到 AWS IoT Core 政策所連接的靜態物件群組時，物件名稱必須符合用戶端 ID。

以下為靜態物件群組的限制：

- 一個群組最多只能有一個直系的父群組。
- 如果群組是另一個群組的子群組，請在建立時指定此項目。
- 您之後便無法變更群組的父群組，因此請務必在建立群組所包含的任何子群組之前，先規劃群組階層並建立父群組。
- 物件所屬群組數有限。
- 在相同的階層內，您無法讓一個物件加入超過一個群組。(亦即，您無法讓物件加入父群組相同的兩個群組。)
- 您無法重新命名群組。
- 物件群組名稱無法包含國際字元，例如 û、é 和 ñ。
- 請勿在物件群組名稱中使用個人識別資訊。物件群組名稱可顯示於未加密的通訊和報告中。

讓政策跟群組連接和分離，可以在許多重要方面強化您的 AWS IoT 操作安全。透過裝置將政策連接至憑證再連接到物件的方式相當耗時，而且在裝置機群中要迅速更新或變更政策也很困難。若政策連接至物件的群組，在輪換物件的憑證時就可以省卻許多步驟。此外，物件變更群組成員資格時，政策就會動態套用至物件，所以每當群組中有裝置變更成員資格，您就不必重新建立一組複雜的許可。

建立靜態物件群組

使用 CreateThingGroup 命令建立動態物件群組：

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

該 CreateThingGroup 命令返回一個包含靜態物件群組的名稱、ID 和 ARN 的響應：

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

> **Note**
>
> 我們不建議在物件群組名稱中使用個人識別資訊。

這個範例是在建立靜態物件群組時，為其指定一個父群組：

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name
 LightBulbs
```

如先前所述，CreateThingGroup 命令傳回的回應會包含物件群組、ID 及 ARN：

```
{
    "thingGroupName": "RedLights",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

> **Important**
>
> 在建立物件群組階層時，請注意下列限制：
>
> - 物件群組只能有一個直接的父系群組。
> - 物件群組可以擁有的直接子系群組數量有限。
> - 群組階層的最大深度有限。

- 物件群組可以擁有的屬性數量有限。(屬性是您可用於存放群組相關資訊的名稱/值對。) 每個屬性名稱和每個值的長度也有限。

描述物件群組

您可以使用 DescribeThingGroup 命令獲得關於物件群組的資訊：

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

DescribeThingGroup 命令會傳回關於指定群組的資訊：

```
{
    "thingGroupName": "RedLights",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
    "thingGroupId": "12345678abcdefgh12345678ijklmnop12345678",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1478299948.882
        "parentGroupName": "Lights",
        "rootToParentThingGroups": [
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ShinyObjects",
                "groupName": "ShinyObjects"
            },
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs",
                "groupName": "LightBulbs"
            }
        ]
    },
    "thingGroupProperties": {
        "attributePayload": {
            "attributes": {
                "brightness": "3400_lumens"
            },
        },
        "thingGroupDescription": "string"
    },
}
```

新增一個物件到一個靜態的物件群組

您可以使用 AddThingToThingGroup 命令將物件新增至靜態物件群組：

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

AddThingToThingGroup 命令不會產生任何輸出。

> **Important**
>
> 一個物件最多可加入 10 個群組。但是在相同的階層內，您無法讓一個物件加入超過一個群組。(亦即，您無法讓物件加入父群組相同的兩個群組。)
> 如果物件屬於 10 個物件群組，而其中有一或多個群組為動態物件群組，則您可以使用 overrideDynamicGroups 旗標讓靜態群組的優先順序高於動態群組。

從靜態物件群組中移除一個物件

您可以使用 RemoveThingFromThingGroup 命令從群組移除物件：

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

RemoveThingFromThingGroup 命令不會產生任何輸出。

列出物件群組中的物件

您可以使用 ListThingsInThingGroup 命令列出屬於群組的物件：

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

ListThingsInThingGroup 命令會傳回該群組的物件清單：

```
{
    "things":[
        "TestThingA"
    ]
}
```

使用 `--recursive` 參數，您可以列出屬於某個群組及其任一子群組中的物件：

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

> **Note**
>
> 此操作為最終一致。換句話說，可能不會一次反映物件群組的變更。

列出物件群組

您可以使用 ListThingGroups 命令來列出帳戶的物件群組：

```
$ aws iot list-thing-groups
```

ListThingGroups 命令會傳回 中物件群組的清單 AWS 帳戶：

```
{
    "thingGroups": [
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
```

```
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

使用選用的篩選條件列出以特定群組為父群組的群組 (--parent-group) 或是名稱開頭為特定字首 (--name-prefix-filter) 的群組。--recursive 參數可讓您列出所有子群組，不只是物件群組的直系子群組：

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

在此情況下， ListThingGroups命令會傳回 中定義之物件群組的直接子群組清單 AWS 帳戶：

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

使用 --recursive 參數與 ListThingGroups 命令來列出物件群組的所有子群組，不只是直系子群組：

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

ListThingGroups 命令會傳回一份清單，列出物件群組的所有子群組：

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
```

```
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
    ]
}
```

> **Note**
>
> 此操作為最終一致。換句話說，可能不會一次反映物件群組的變更。

列出物件的群組

您可以使用 ListThingGroupsForThing 命令列出物件所屬的群組：

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

ListThingGroupsForThing 命令會傳回一份清單，其中會列出此物件所屬的直接物件群組：

```
{
    "thingGroups":[
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
```

```
}
```

您也可以使用內嵌函數 在規則引擎內存取此 APIget_registry_data()。您可以使用此函數，直接在 AWS IoT 規則內呼叫 和 DescribeThing ListThingGroupsForThing APIs，根據裝置登錄檔資料啟用即時訊息處理和路由，以動態存取和利用物件登錄檔資訊（包括屬性、物件類型和群組成員資格）。如需詳細資訊，請參閱get_registry_data。

更新靜態物件群組

您可以使用 UpdateThingGroup 命令更新靜態物件群組的屬性：

```
$ aws iot update-thing-group --thing-group-name "LightBulbs" --thing-group-properties
 "thingGroupDescription=\"this is a test group\", attributePayload=\"{\"attributes
\"={\"Owner\"=\"150\",\"modelNames\"=\"456\"}}"
```

UpdateThingGroup 命令的回應會包含更新之後的群組版本編號：

```
{
    "version": 4
}
```

> **Note**
>
> 一個物件可以擁有的屬性數量有限。

刪除物件群組

若要刪除物件群組，請使用 DeleteThingGroup 命令：

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

DeleteThingGroup 命令不會產生任何輸出。

> **Important**
>
> 如果您嘗試刪除具有子物件群組的物件群組，就會收到錯誤：
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> ```

```
operation: Cannot delete thing group : RedLights when there are still child
 groups attached to it.
```

刪除群組之前，請先刪除任何子群組。

您可以刪除具有子物件的群組，但由群組成員資格授予物件的任何許可將不再適用。在刪除已連接政策的群組之前，請確認移除這些許可不會使群組中的物件無法正常運作。此外，顯示物件所屬群組的命令（例如 ListGroupsForThing) 可能會繼續顯示群組，同時更新雲端中的記錄。

將政策附加到靜態物件群組

您可以使用 AttachPolicy 命令將政策連線至靜態物件群組，進而連線至該群組及其子群組中的所有物件。

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

AttachPolicy 命令不會產生任何輸出。

Important

您可以附加至群組的政策數上限為兩個。

Note

我們不建議在政策名稱中使用個人識別資訊。

`--target` 參數可為物件群組 ARN (如上所述)、憑證 ARN 或 Amazon Cognito Identity。如需關於政策、憑證與身分驗證的詳細資訊，請參閱 身分驗證。

如需詳細資訊，請參閱 AWS IoT Core 政策。

將政策與靜態物件群組分離

您可以使用 DetachPolicy 命令從物件群組分離政策，進而讓政策與該群組及其子群組中的所有物件分離。

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
LightBulbs" --policy-name "myLightBulbPolicy"
```

DetachPolicy 命令不會產生任何輸出。

列出連線至靜態物件群組的政策

您可以使用 ListAttachedPolicies 命令列出連線至群組的政策：

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-
west-2:123456789012:thinggroup/RedLights"
```

`--target` 參數可為物件群組 ARN (如上所述)、憑證 ARN 或 Amazon Cognito Identity。

加入選用的 `--recursive` 參數，納入與該群組之父群組連接的所有政策。

ListAttachedPolicies 命令會傳回政策清單：

```
{
    "policies": [
        "MyLightBulbPolicy"
        ...
    ]
}
```

列出政策所在的群組

您可以使用 ListTargetsForPolicy 命令列出政策連接的目標，包括任何群組：

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

新增選用的 `--page-size` *`number`* 參數來指定每次查詢所回傳的結果數量上限，並在後續呼叫使用 `--marker` *`string`* 參數以擷取下一組結果。

ListTargetsForPolicy 命令會傳回一份目標清單以及用於擷取更多結果的符記：

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
}
```

為物件取得有效政策

您可以使用 GetEffectivePolicies 命令列出物件的有效政策，包括連接至該物件所屬任何群組的政策 (無論此群組是直系的父群組或旁系的上階群組)：

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

使用 `--principal` 參數指定連接物件之憑證的 ARN。如果您使用 Amazon Cognito 身分驗證，請使用 `--cognito-identity-pool-id` 參數並新增選用的 `--principal` 參數來指定一個 Amazon Cognito 身分。如果您僅指定 `--cognito-identity-pool-id`，則會傳回該身分集區角色對於未授權使用者的相關政策。如果您兩者都使用，則會傳回該 identity pool 角色對於授權使用者的相關政策。

`--thing-name` 參數為選用，可用於取代 `--principal` 參數。若使用此參數，則會傳回連接物件所屬群組的政策，以及連接這些群組之父群組 (最高到階層裡之根群組) 的政策。

GetEffectivePolicies 命令會傳回政策清單：

```
{
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

MQTT 動作的測試授權

您可以使用 TestAuthorization 命令測試 MQTT 動作 (`Publish`、`Subscribe`) 是否可用於物件：

```
aws iot test-authorization \
    --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --auth-infos "{\"actionType\": \"PUBLISH\", \"resources\": [ \"arn:aws:iot:us-
east-1:123456789012:topic/my/topic\"]}"
```

使用 `--principal` 參數指定連接物件之憑證的 ARN。若使用 Amazon Cognito Identity 驗證，請指定一個 Cognito Identity 為 `--principal` 或使用 `--cognito-identity-pool-id` 參數，或兩者並用。(如果您僅指定 `--cognito-identity-pool-id`，則需考量該 identity pool 角色對於未授權使用者的相關政策。如果您兩者都使用，則需考量該 identity pool 角色對於授權使用者的相關政策。

若要指定一個或多個您想要測試的 MQTT 動作，請列出資源以及執行 `--auth-infos` 參數後的動作類型。`actionType` 欄位應包含 "PUBLISH"、"SUBSCRIBE"、"RECEIVE" 或 "CONNECT"。`resources` 欄位應包含一份資源 ARN 的清單。如需詳細資訊，請參閱 AWS IoT Core 政策。

若要測試新增政策的效果，請為其指定 `--policy-names-to-add` 參數。若要測試移除政策的效果，請為其指定 `--policy-names-to-skip` 參數。

您可以使用選用的 `--client-id` 參數進一步加強結果。

TestAuthorization 命令會傳回您指定的每筆 `--auth-infos` 查詢所允許或拒絕的動作詳細資訊：

```
{
    "authResults": [
        {
            "allowed": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "authDecision": "string",
            "authInfo": {
                "actionType": "string",
                "resources": [ "string" ]
            },
            "denied": {
                "explicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                },
                "implicitDeny": {
```

```
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                }
            },
            "missingContextValues": [ "string" ]
        }
    ]
}
```

動態物件群組

動態物件群組是從登錄檔中的特定搜尋查詢建立的。搜尋查詢參數，例如裝置連線能力、裝置影子建立和 AWS IoT Device Defender 違規資料都支援此參數。動態物件群組需要啟用機群索引，才能索引、搜尋和彙總裝置的資料。您可以在建立動態物件群組之前，使用機群索引搜尋查詢來預覽動態物件群組中的物件。如需詳細資訊，請參閱機群索引及查詢語法。

> **Note**
>
> 動態物件群組操作是在登錄檔操作下計量。如需詳細資訊，請參閱AWS IoT Core 其他計量詳細資訊。

動態物件群組與靜態物件群組的差異如下：

- 物件成員資格沒有明確定義。若要建立動態物件群組，請定義搜尋查詢字串來判斷群組成員資格。
- 動態物件群組不能是階層的一部分。
- 動態物件群組無法套用政策。
- 您會使用不同的命令集來建立、更新和刪除動態物件群組。對於所有其他操作，您可以對這兩種類型的物件群組使用相同的命令。
- 每個 的動態群組數量 AWS 帳戶 有限。

- 請勿在物件群組名稱中使用個人識別資訊。物件群組名稱可顯示於未加密的通訊和報告中。

如需靜態物件群組的詳細資訊，請參閱 靜態物件群組。

動態物件群組的使用案例

您可以針對下列使用案例使用動態物件群組：

指定動態物件群組做為任務的目標

建立以動態物件群組做為目標的連續任務，可讓您在裝置符合所需條件時自動將裝置設為目標。條件可以是連線狀態或存放在登錄檔或影子中的任何條件，例如軟體版本或模型。如果物件未出現在動態物件群組中，則不會從任務接收任務文件。

例如，如果您的裝置機群需要韌體更新，才能將更新程序期間中斷的風險降至最低，而且您只想要在電池壽命大於 80% 的裝置上更新韌體。您可以建立名為 80PercentBatteryLife，只包含電池壽命超過 80% 的裝置，並將其用作任務的目標。只有符合您電池壽命條件的裝置才會收到韌體更新。當裝置達到 80% 的電池壽命條件時，它們會自動新增至動態物件群組，並會收到韌體更新。

您也可以擁有多個具有不同韌體或作業系統的裝置模型，因此需要不同版本的新軟體更新。這是具有連續任務的動態群組最常見的使用案例，您可以在其中為每個裝置模型、韌體和作業系統組合建立動態群組。然後，您可以為每個這些動態群組設定連續任務，以在裝置根據定義的條件自動成為這些群組的成員時推送軟體更新。

如需將物件群組指定為任務目標的詳細資訊，請參閱 CreateJob。

使用動態群組成員資格變更來執行所需的動作

每次在動態物件群組中新增或移除裝置時，都會傳送通知至 MQTT 主題，做為登錄事件更新的一部分。您可以設定AWS IoT Core 規則，根據動態群組成員資格更新與 AWS 服務互動，並採取所需的動作。範例動作包括寫入 Amazon DynamoDB、叫用 Lambda 函數，或傳送通知至 Amazon SNS。

將裝置新增至動態物件群組以進行自動違規偵測

AWS IoT Device Defender Detect 客戶可以在動態物件群組上定義安全性設定檔。動態物件群組的裝置會自動透過群組上定義的安全性描述檔偵測違規。

在動態物件群組上設定日誌層級，以觀察具有精細記錄的裝置

您可以在動態物件群組上指定日誌層級。如果您只想要為符合特定條件的裝置自訂記錄層級和詳細資訊，這會很有用。例如，如果您懷疑具有特定韌體版本的裝置在特定規則發佈的主題上造成錯誤，建議您設定詳細記錄以偵錯這些問題。在此情況下，您可以為具有此韌體版本的所有裝置建立動態群組，我們假設該群組會儲存為登錄檔屬性或裝置影子。然後，您可以設定偵錯層級，並將記錄目標定義為此動

態物件群組。如需精細記錄的詳細資訊，請參閱AWS IoT 使用 CloudWatch Logs 進行監控。如需如何為特定物件群組指定記錄層級的詳細資訊，請參閱設定資源特定的登入 AWS IoT。

建立動態物件群組

使用 CreateDynamicThingGroup 命令建立動態物件群組。若要為 80PercentBatteryLife 案例建立動態物件群組，請使用 create-dynamic-thing-group CLI 命令：

```
$ aws iot create-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --query-string "attributes.batterylife80"
```

> **Note**
>
> 請勿在動態物件群組名稱中使用個人身分識別資訊。

CreateDynamicThingGroup 命令會傳回回應。回應包含索引名稱、查詢字串、查詢版本、物件群組名稱、物件群組 ID 和物件群組的 Amazon Resource Name (ARN)：

```
{
    "indexName": "AWS_Things",
    "queryVersion": "2017-09-30",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx"
}
```

動態物件群組的建立不會同時發生。動態物件群組回填需要一些時間才能完成。當您建立動態物件群組時，群組的狀態會設為 BUILDING。回填處理完成時，狀態會變更為 ACTIVE。若要查看動態物件群組的狀態，您可以使用 DescribeThingGroup 命令。

描述動態物件群組

使用 DescribeThingGroup 命令以取得動態物件群組的相關資訊：

```
$ aws iot describe-thing-group --thing-group-name "80PercentBatteryLife"
```

DescribeThingGroup 命令會傳回關於指定群組的資訊：

```
{
    "status": "ACTIVE",
    "indexName": "AWS_Things",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1548716921.289
    },
    "thingGroupProperties": {},
    "queryVersion": "2017-09-30",
    "thingGroupId": "84dd9b5b-2b98-4c65-84e4-be0e1ecf4fd8"
}
```

在動態物件群組DescribeThingGroup上執行會傳回動態物件群組特有的屬性。傳回屬性範例為 queryString 和 狀態。

動態物件群組的狀態可能會需要以下的值：

`ACTIVE`

動態物件群組已準備就緒。

`BUILDING`

正在建立動態群組，且正在處理物件成員資格。

`REBUILDING`

正在更新動態物件群組的成員資格，接著是調整該群組搜尋查詢。

> **Note**
>
> 建立動態物件群組之後，無論其狀態為何，都可以使用它。只有狀態為 `ACTIVE` 的動態物件群組，才會包含所有與該動態物件群組搜尋查詢相符的物件。狀態為 `BUILDING` 和 `REBUILDING` 的動態物件群組，則可能不會包含所有符合搜尋查詢的物件。

更新動態物件群組

使用 UpdateDynamicThingGroup 命令來更新動態物件群組的屬性，包括該群組的搜尋查詢。下列命令會更新兩個屬性。一個是物件群組描述，另一個是將成員資格條件變更為電池壽命 > 85 的查詢字串：

```
$ aws iot update-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --thing-
group-properties "thingGroupDescription=\"This thing group contains devices with a
 battery life greater than 85 percent.\"" --query-string "attributes.batterylife85"
```

UpdateDynamicThingGroup 命令的回應會包含更新之後的群組版本編號：

```
{
    "version": 2
}
```

動態物件群組的更新不會同時發生。動態物件群組回填需要一些時間才能完成。當您更新動態物件群組時，群組的狀態會變更為 ，REBUILDING同時群組會更新其成員資格。回填處理完成時，狀態會變更為 ACTIVE。若要查看動態物件群組的狀態，您可以使用 DescribeThingGroup 命令。

刪除動態物件群組

使用 DeleteDynamicThingGroup 命令來刪除動態物件群組：

```
$ aws iot delete-dynamic-thing-group --thing-group-name "80PercentBatteryLife"
```

DeleteDynamicThingGroup 命令不會產生任何輸出。

雲端中的記錄正在更新時，顯示物件所屬群組的命令 (例如 ListGroupsForThing) 可能仍會顯示出該群組。

動態和靜態物件群組限制

動態物件群組和靜態物件群組共用下列限制：

- 物件群組可以擁有的屬性數量有限。
- 物件所屬群組數有限。
- 您無法重新命名物件群組。

- 物件群組名稱無法包含國際字元，例如 û、é 和 ñ。

動態物件群組限制

動態物件群組有下列限制：

機群索引

啟用機群索引服務後，您可以在裝置機群上執行搜尋查詢。您可以在機群索引回填完成後建立和管理動態物件群組。回填程序的完成時間會直接受到向 註冊的裝置機群大小影響 AWS 雲端。在您為動態物件群組啟用機群索引服務後，除非您刪除所有動態物件群組，否則無法停用服務。

> **Note**
>
> 如果您有查詢機群索引的許可，您就可以存取整個機群的物件資料。

動態物件群組的數量有限

動態物件群組的數量有限。

成功的命令可以記錄錯誤

當您建立或更新動態物件群組時，有些物件可能符合包含在動態物件群組中的資格，但不會新增至其中。該案例會在記錄錯誤和產生`AddThingToDynamicThingGroupsFailed`指標時，導致成功的建立或更新命令。單一指標可以代表多個日誌項目。

發生下列情況時，會建立 CloudWatch 日誌中的錯誤日誌項目：

- 符合資格的物件無法新增至動態物件群組。
- 物件會從動態物件群組中移除，以將其新增至另一個群組。

當物件符合新增至動態物件群組的資格時，請考慮下列事項：

- 該物件是否已經在盡可能多的群組中存在？ (請參閱限制數量)。
  - 否：該物件被新增到動態物件組。
  - 是：事情是否是任何動態物件群組的成員？

- 否：無法將物件新增至動態物件群組，系統已記錄錯誤，並產生 AddThingToDynamicThingGroupsFailed 指標。
- 是：要加入的動態物件群組是否比已是成員的任何動態物件群組還舊？
  - 否：無法將物件新增至動態物件群組，系統已記錄錯誤，並產生 AddThingToDynamicThingGroupsFailed 指標。
  - 是：從最新的動態物件群組中移除物件、記錄錯誤，並將物件新增至動態物件群組。這會產生錯誤和移除物件之動態物件群組的 AddThingToDynamicThingGroupsFailed 指標。

當動態物件群組中的物件不再符合搜尋查詢時，該物件會從動態物件群組中移除。同樣地，當物件更新為符合動態物件群組的搜尋查詢時，該物件會如先前所述新增至群組。這些新增和移除都是正常的，不會產生錯誤日誌項目。

啟用 overrideDynamicGroups 時，靜態群組會優先於動態群組

物件所屬群組數有限。當您使用 AddThingToThingGroup 或 UpdateThingGroupsForThing 命令更新物件成員資格時，新增 `--overrideDynamicGroups` 參數會提供靜態物件群組優先順序，而非動態物件群組。

當您將物件新增至靜態物件群組時，請考慮下列事項：

- 物件是否已屬於群組的最大數量？
  - 否：該物件會新增到靜態物件群組。
  - 是：該物件是否在任何動態群組中？
    - 否：不能將該物件新增到物件群組。命令會引發例外狀況。
    - 是：是否已啟用 --overrideDynamicGroups？
      - 否：不能將該物件新增到物件群組。命令會引發例外狀況。
      - 是：該物件會從最近建立的動態物件群組中移除、記錄錯誤，並針對移除該物件的動態物件群組產生 AddThingToDynamicThingGroupsFailed 指標。然後，事情會新增到靜態物件群組中。

較舊的動態物件群組優先於較新的動態物件群組

物件所屬群組數有限。當建立或更新操作為物件建立額外的群組資格，且物件已達到其群組限制時，可能會從另一個動態物件群組移除以啟用此新增。如需有關如何發生這種情況的詳細資訊，請參閱 成功的命令可以記錄錯誤 和 啟用 overrideDynamicGroups 時，靜態群組會優先於動態群組 以取得範例。

從動態物件群組移除物件時，會記錄錯誤並引發事件。

您無法將政策套用至動態物件群組

嘗試將政策套用至動態物件群組會產生例外狀況。

動態物件群組成員資格為最終一致

只會針對登錄檔對物件的最終狀態進行評估。如果狀態快速更新，可能會略過中繼狀態。避免將規則或任務與成員資格取決於中間狀態的動態物件群組建立關聯。

將 AWS IoT 物件與 MQTT 用戶端連線建立關聯

當您將 X.509 憑證連接至單一物件時，即為專屬 AWS IoT 物件關聯。在此情況下，憑證無法與其他物件搭配使用。透過確保憑證僅供單一 IoT 物件使用，有助於防止安全漏洞。

在 中 AWS IoT，用戶端 ID 是物件或裝置連線到 AWS IoT Core MQTT 代理程式時的唯一識別符。如果您使用非專屬關聯，則可以將多個物件連接到相同的憑證。建立非獨佔物件關聯時，若要維持明確的關聯並避免潛在的衝突，您必須比對用戶端 ID 與物件名稱。

在本主題中

- 使用案例
- 如何將物件與連線建立關聯

使用案例

將物件與連線建立關聯可提供下列功能。

> **Note**
>
> 請注意，如果您的 IoT 物件和用戶端連線具有非專屬關聯，您可以使用生命週期事件功能以外的所有下列功能。若要在生命週期事件訊息中包含您的物件名稱，您的 IoT 物件和用戶端連線必須具有專屬關聯。

物件政策變數 - 您可以使用物件政策變數來授權裝置存取 AWS IoT API 操作。這些變數可讓您撰寫 AWS IoT Core 政策，根據名稱、類型和屬性值等物件屬性來授予或拒絕許可。透過使用物件政策變

數，您可以套用相同的政策來控制多個 AWS IoT Core 裝置。這可讓您簡化政策管理並減少資源重複。如需詳細資訊，請參閱物件政策變數。

生命週期事件 - 您可以在生命週期事件中接收物件名稱（例如，連線、中斷連線和訂閱，以及取消訂閱）。這允許處理訊息中包含的物件名稱，例如規則。如需詳細資訊，請參閱生命週期事件。

資源特定的記錄 - 您可以為物件群組設定資源特定的記錄，並輕鬆為所定義物件群組中的所有物件套用所需的記錄組態。如需詳細資訊，請參閱???。

成本分配 - 您可以使用成本分配的自訂標籤建立帳單群組，並將物件新增至這些群組。如需詳細資訊，請參閱帳單群組。

如何將物件與連線建立關聯

如果您的用戶端 ID 符合登錄檔中的物件名稱，則在您將 X.509 憑證連接至該 IoT 物件之後， AWS IoT Core 會將用戶端連線與物件建立關聯。如果您的用戶端 ID 與登錄檔中的物件名稱不符，您可以只將 X.509 憑證連接至物件來建立此關聯。具有此獨佔附件的物件稱為獨佔物件。否則，它稱為非獨佔物件。當憑證與獨家實物相關聯時，只有在您將憑證與獨家實物分離時，此憑證才能與其他實物相關聯。在本節中，選擇 AWS 管理主控台 或 AWS CLI 將物件與連線建立關聯。

AWS 管理主控台

僅使用 將憑證連接至物件 AWS 管理主控台。

1. 在 AWS IoT 主控台中開啟 AWS IoT 首頁。在左側導覽中，從安全性中選擇憑證。
2. 在憑證頁面上，選擇您要附加物件的憑證。然後從頁面右上角的動作中選擇連接至物件。

   或者，選擇憑證並導覽至憑證詳細資訊頁面。選擇實物索引標籤，然後選擇連接至實物。
3. 在將憑證連接至 物件（附加）頁面上，勾選將物件與連線建立關聯核取方塊。然後從物件下拉式清單中選擇要將此憑證連接至 的物件。
4. 選擇連接物件 (s)。如果動作成功，您會看到一個橫幅，指出「成功將物件連接到您的憑證」，而該物件會新增到物件索引標籤。

使用 從獨佔物件分離憑證 AWS 管理主控台

1. 在 AWS IoT 主控台中開啟 AWS IoT 首頁。在左側導覽中，從安全性中選擇憑證。
2. 在憑證頁面上，選擇憑證並導覽至憑證詳細資訊頁面。
3. 在憑證詳細資訊頁面上，選擇實物索引標籤。然後選擇您要分離憑證的物件。選擇分離物件。

4. 在分離物件視窗中，確認您的動作。請選擇分離。如果動作成功，您會看到橫幅，指出「成功從憑證分離物件」，而且物件不會再出現在物件索引標籤中。

AWS CLI

1. 若要使用 將憑證連接至物件 AWS CLI，請執行 attach-thing-principal 命令。若要指定專屬certificate-to-thing附件，您必須在 `--thing-principal-type EXCLUSIVE_THING`欄位中指定 。範例命令可以是下列項目。

```
aws iot attach-thing-principal \
    --thing-name "thing_1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-principal-type "EXCLUSIVE_THING"
```

此命令不會產生任何輸出。如需詳細資訊，請參閱???。

2. 若要列出與指定憑證相關的物件以及附件類型，請執行 `list-principal-things-v2`命令。連接類型是指憑證如何連接到物件。範例命令可以是下列項目。

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

輸出可能如下所示。

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        }
    ]
}
```

如需詳細資訊，請參閱???。

3. 若要列出與指定物件相關聯的主體以及連接類型，請執行`list-thing-principals-v2`命令 。連接類型是指憑證如何連接到物件。範例命令可以是下列項目。

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

輸出可能如下所示。

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        }
    ]
}
```

如需詳細資訊，請參閱???。

4. 若要從物件分離憑證，請執行 detach-thing-principal 命令。

```
aws iot detach-thing-principal \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-name "thing_1"
```

此命令不會產生任何輸出。如需詳細資訊，請參閱???。

新增傳播屬性讓訊息內容更豐富

在 中 AWS IoT Core，您可以透過新增傳播屬性來豐富來自裝置的 MQTT 訊息，這些屬性是來自物件屬性或連線詳細資訊的內容中繼資料。此程序稱為訊息擴充，在各種情況下都很有幫助。例如，您可以為每個傳入發佈操作擴充訊息，而無需進行任何裝置端變更或需要使用規則。透過利用傳播屬性，您可以受益於更有效率且符合成本效益的方式，以富集 IoT 資料，而無需設定規則或管理重新發佈組態的複雜性。

訊息擴充功能可供使用基本擷取和訊息代理程式 AWS IoT Core 的客戶使用。請注意，發佈裝置可以使用任何 MQTT 版本時，訂閱者 （使用訊息的應用程式或服務） 必須支援 MQTT 5，才能接收具有傳播

屬性的豐富訊息。富集的訊息會以 MQTT 5 使用者屬性新增至從裝置發佈的每個訊息。如果您使用規則，則可以利用 get_user_properties 函數來擷取富集的資料，以便根據資料進行訊息路由或處理。

在 中 AWS IoT Core，您可以在建立或更新物件類型時，使用 AWS 管理主控台 或 來新增傳播屬性 AWS CLI。

> **Important**
>
> 新增傳播屬性時，您必須確定發佈訊息的用戶端已使用憑證進行身分驗證。如需詳細資訊，請參閱用戶端身分驗證。

> **Note**
>
> 如果您嘗試在主控台中使用 MQTT 測試用戶端來測試此功能，則可能無法運作，因為此功能需要使用相關聯的憑證進行 MQTT 用戶端驗證。

AWS 管理主控台

使用 新增訊息擴充的傳播屬性 AWS 管理主控台

1. 在 AWS IoT 主控台中開啟 AWS IoT 首頁。在左側導覽中，從管理中選擇所有裝置。然後選擇物件類型。
2. 在物件類型頁面上，選擇建立物件類型。

   若要透過更新物件類型來設定訊息擴充，請選擇物件類型。然後在物件類型詳細資訊頁面上，選擇更新。
3. 在建立物件類型頁面上，選擇或在物件類型屬性中輸入物件類型資訊。

   如果您選擇更新物件類型，則在上一個步驟中選擇更新後，您會看到物件類型屬性。
4. 在其他組態中，展開傳播屬性。然後選擇物件屬性，然後輸入您要填入已發佈 MQTT5 訊息的物件屬性。使用 主控台，您最多可以新增三個物件屬性。

   在傳播屬性區段中，選擇連線屬性，然後輸入屬性類型和選擇性的屬性名稱。
5. 或者，新增標籤。然後選擇建立物件類型。

   如果您選擇更新物件類型，請選擇更新物件類型。

AWS CLI

1. 若要使用 新增物件類型來新增訊息擴充的傳播屬性 AWS CLI，請執行 create-thing-type 命令。範例命令可以是下列項目。

```
aws iot create-thing-type \
    --thing-type-name "LightBulb" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

   命令的輸出可能如下所示。

```
{
 "thingTypeName": "LightBulb",
 "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
 "thingTypeId": "ce3573b0-0a3c-45a7-ac93-4e0ce14cd190"
}
```

2. 若要使用 更新物件類型來設定訊息擴充 AWS CLI，請執行 update-thing-type 命令。請注意，您只能在執行此命令mqtt5Configuration時更新 。範例命令可以是下列項目。

```
aws iot update-thing-type \
    --thing-type-name "MyThingType" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

   此命令不會產生任何輸出。

3. 若要描述物件類型，請執行 describe-thing-type命令。此命令會在 thing-type-properties 欄位中產生具有訊息擴充組態資訊的輸出。範例命令可以是下列項目。

```
aws iot describe-thing-type \
    --thing-type-name "LightBulb"
```

   輸出可能如下所示。

```
{
 "thingTypeName": "LightBulb",
```

```
 "thingTypeId": "bdf72512-0116-4392-8d79-bf39b17ef73d",
 "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/LightBulb",
 "thingTypeProperties": {
  "mqtt5Configuration": {
   "propagatingAttributes": [
    {
     "userPropertyKey": "iot:ClientId",
     "connectionAttribute": "iot:ClientId"
    },
    {
     "userPropertyKey": "test",
     "thingAttribute": "attribute"
    }
   ]
  }
 },
 "thingTypeMetadata": {
  "deprecated": false,
  "creationDate": "2024-10-18T17:37:46.656000+00:00"
 }
}
```

如需詳細資訊，請參閱???。

標記您的 AWS IoT 資源

為了協助您管理和組織物件群組、物件類型、主題規則、任務、排程稽核及安全性描述檔，您可以選擇性地以標籤形式將您自己的中繼資料指派給其中每個資源。本節說明標籤並示範如何建立它們。

為了協助您管理與物件相關的成本，您可以建立包含物件的帳單群組。您可以將包括中繼資料的標籤指派到每個帳單群組。本節也會討論可用來建立和管理的帳單群組和命令。

標籤基本概念

您可以使用標籤，以不同的方式（例如，依用途、擁有者或環境）分類 AWS IoT 資源。當您有許多相同類型的資源時，這會很有用，因為您可以依據先前指派的標籤快速識別資源。每個標籤皆包含由您定義的一個「索引鍵」與選擇性的「值」。例如，您可以為您的物件類型定義一組標籤，協助您以類型追蹤裝置。我們建議您為每種資源類型建立符合您需求的一組標籤金鑰。使用一致的標籤金鑰組可讓您更輕鬆管理您的 資源。

您可以根據新增或套用的標籤來搜尋與篩選資源。您也可以使用帳單群組標籤來分類和追蹤您的成本。您也可以使用標籤來控制對您資源的存取，如 搭配 IAM 政策使用標籤 所述。

為了方便使用， AWS 管理主控台中的標籤編輯器提供集中、統一的方式來建立和管理標籤。如需詳細資訊，請參閱使用標籤編輯器使用 AWS 管理主控台。

您也可以使用 AWS CLI 和 AWS IoT API 處理標籤。當您使用下列命令在 Tags 欄位中建立標籤時，您可將標籤與物件群組、物件類型、主題規則、任務、安全性設定檔、政策、帳單群組、物件相關套件和版本建立關聯：

- CreateBillingGroup
- CreateDestination
- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate
- CreatePolicy
- CreateScheduledAudit
- CreateSecurityProfile

- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway
- CreateWirelessDevice

您可以使用下列命令新增、修改或刪除支援標記功能的現有資源標籤：

- TagResource
- ListTagsForResource
- UntagResource

您可以編輯標籤索引鍵和值，也可以隨時從資源中移除標籤。您可以將標籤的值設為空白字串，但您無法將標籤的值設為 Null。如果您新增的標籤具有與該資源上現有標籤相同的索引鍵，則新值會覆寫舊值。如果您刪除資源，也會刪除與該資源相關聯的任何標籤。

標籤的限制與上限

以下基本限制適用於 標籤：

- 每個資源的標籤數上限：50
- 金鑰長度上限：127 個 UTF-8 Unicode 字元
- 值長度上限：255 個 UTF-8 Unicode 字元
- 標籤鍵與值皆區分大小寫。
- 請勿於標籤名稱或值中使用 `aws:` 字首。它保留供 AWS 使用。您不可編輯或刪除具此字首的標籤名稱或值。具此字首的標籤，不算在受資源限制的標籤計數內。
- 如果您的標記結構描述是跨多項服務和資源使用，請記得其他服務可能會有字元使用限制。允許使用的字元包括可用 UTF-8 表示的英文字母、空格和數字，以及以下特殊字元：+ - = . _ : / @。

搭配 IAM 政策使用標籤

您可以在用於 AWS IoT API 動作的 IAM 政策中，套用以標籤為基礎的資源層級許可。這可讓您更有效地控制使用者可以建立、修改或使用哪些資源。您可以使用 Condition 元素 (也稱為 Condition 區塊)，以及 IAM 政策中的以下條件內容金鑰和值，來根據資源標籤控制使用者存取 (許可)：

- 使用 aws:ResourceTag/*tag-key*: *tag-value* 以允許或拒絕資源上具有特定標籤的使用者動作。
- 使用 aws:RequestTag/*tag-key*: *tag-value* 以在提出 API 請求時，要求使用 (或不使用) 特定標籤，以建立或修改允許標籤的資源。
- 使用 aws:TagKeys: [*tag-key*, ...] 以在提出 API 請求時，要求使用 (或不使用) 特定標籤金鑰集，以建立或修改允許標籤的資源。

> **Note**
>
> IAM 政策中的條件內容索引鍵和值僅適用於資源識別符為必要參數的那些 AWS IoT 動作。例如，根據條件內容金鑰和值，不允許或拒絕使用 DescribeEndpoint，因為在此請求中所參照的項目沒有可標記資源 (物件群組、物件類型、主題規則、任務或安全性描述檔)。如需可標記 AWS IoT 的資源及其支援的條件索引鍵的詳細資訊，請參閱 動作、資源和條件索引鍵 AWS IoT。

如需使用標籤的詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的使用標籤控制。該指南的 IAM JSON 政策參考章節有詳細的語法、說明，還有元素、變數範例，以及在 IAM 中的 JSON 政策評估邏輯。

以下範例政策會針對 ThingGroup 動作套用兩個以標籤為基礎的限制。受到此政策限制的 IAM 使用者：

- 無法建立包含 "env=prod" 標籤的物件群組 (在範例中，請參閱行 "aws:RequestTag/env" : "prod")。
- 無法修改或存取具有現有標籤 "env = prod" 的物件群組 (在範例中，請參閱此行 "aws:ResourceTag/env" : "prod")。

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iot:CreateThingGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

您也可以透過將其包含在清單中，為特定標籤金鑰指定多個標籤值，如下所示：

```
            "StringEquals" : {
              "aws:ResourceTag/env" : ["dev", "test"]
            }
```

> **Note**
>
> 如果您允許或拒絕使用者根據標籤存取資源，請務必考慮明確拒絕使用者將這些標籤新增至相同資源或從中移除的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

帳單群組

AWS IoT 不允許您將標籤直接套用至個別物件，但可讓您將物件放置在帳單群組中，並將標籤套用至這些物件。對於 AWS IoT，根據標籤配置的成本和用量資料僅限於帳單群組。

AWS IoT Core for LoRaWAN 資源，例如無線裝置和閘道，無法新增至帳單群組。不過，它們可以與物件相關聯，這些 AWS IoT 物件可以新增到帳單群組。

下列命令可供使用：

- AddThingToBillingGroup 可將物件新增至帳單群組。
- CreateBillingGroup 會建立帳單群組。
- DeleteBillingGroup 會刪除帳單群組。
- DescribeBillingGroup 會傳回關於帳單群組的資訊。
- ListBillingGroups 會列出您建立的帳單群組。
- ListThingsInBillingGroup 會列出您已新增至特定帳單群組的物件。
- RemoveThingFromBillingGroup 會將特定物件從帳單群組中移除。
- UpdateBillingGroup 會更新關於帳單群組的資訊。
- CreateThing 可讓您在建立物件時，為其指定帳單群組。
- DescribeThing 會傳回物件的描述，包括該物件所屬的帳單群組 (若有的話)。

AWS IoT 無線 API 提供這些動作，將無線裝置和閘道與 AWS IoT 物件建立關聯。

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

檢視成本分配和用量資料

您可以使用帳單群組標籤來分類和追蹤您的成本。當您將標籤套用到帳單群組 （因此套用到它們包含的物件） 時， 會以逗號分隔值 (CSV) 檔案的形式 AWS 產生成本分配報告，其中包含標籤彙總的用量和成本。您可以套用代表業務類別 (例如成本中心、應用程式名稱或擁有者) 的標籤，來整理多個服務中的成本。如需使用標籤進行成本分配的詳細資訊，請參閱《AWS 帳單和成本管理使用者指南》中的使用成本分配標籤。

> **Note**
>
> 若要準確地將用量和成本資料與您已置於帳單群組的物件建立關聯，每個裝置或應用程式皆必須：
>
> - 註冊為 中的物件 AWS IoT。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。
> - 僅使用物件名稱做為用戶端 ID，透過 MQTT 連線至 AWS IoT 訊息中介裝置。如需詳細資訊，請參閱the section called “裝置通訊協定”。如果您的用戶端 ID 與物件名稱不符，您可以啟用獨佔物件附件來建立關聯。如需詳細資訊，請參閱???。
> - 使用與物件關聯的用戶端憑證驗證。

帳單群組可使用以下定價維度 (根據與帳單群組關聯的物件活動)：

- 連線能力 (根據做為用戶端 ID 以用於連線的物件名稱)。
- 簡訊 (根據物件傳入和傳出的訊息；僅限 MQTT)。
- 影子操作 (根據其訊息觸發影子更新的物件)。
- 觸發的規則 (根據其傳入訊息已觸發規則的物件；不適用於 MQTT 生命週期事件所觸發的那些規則)。
- 物件索引更新 (根據新增到索引的物件)。
- 遠端動作 (根據更新的物件)。
- AWS IoT Device Defender 偵測報告 （根據報告其活動的物件）。

根據標籤的成本和用量資料 (以及帳單群組的報告) 不會反映以下活動：

- 裝置登錄操作 (包括物件、物件群組和物件類型的更新)。如需詳細資訊，請參閱 使用 管理裝置 AWS IoT。
- 物件群組索引更新 (在新增物件群組時)。

- 索引搜尋查詢。
- 裝置佈建.
- AWS IoT Device Defender 稽核報告。

中的安全性 AWS IoT

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，該架構專為符合最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。若要了解適用的合規計劃 AWS IoT，請參閱AWS 合規計劃範圍內的服務。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您了解如何在使用 時套用共同責任模型 AWS IoT。下列主題說明如何設定 AWS IoT 以符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 AWS IoT 資源。

主題

- AWS IoT 安全性
- 身分驗證
- Authorization
- 中的資料保護 AWS IoT Core
- 的身分和存取管理 AWS IoT
- 記錄和監控
- AWS IoT 核心的合規驗證
- AWS IoT Core 中的彈性
- AWS IoT Core 搭配界面 VPC 端點使用
- 中的基礎設施安全 AWS IoT
- 使用 AWS IoT Core 對生產機群或裝置進行安全監控
- 中的安全最佳實務 AWS IoT Core
- AWS 訓練和認證

AWS IoT 安全性

每個連線的裝置或用戶端都必須具有憑證才能與 AWS IoT互動。所有往返 的流量 AWS IoT 都會透過 Transport Layer Security (TLS) 安全地傳送。 AWS 雲端安全機制會在資料在 AWS IoT 和其他 AWS 服務之間移動時保護資料。



- 您需負責管理 AWS IoT中的裝置憑證 (X.509 憑證、 AWS 憑證、Amazon Cognito 身分、聯合身分或自訂身分驗證字符) 以及政策。您也必須負責為每個裝置指派唯一身分，並管理每個裝置或裝置群組的許可。
- 您的裝置會透過安全的 TLS 連線 AWS IoT ，使用 X.509 憑證或 Amazon Cognito 身分連線至 。在研究和開發期間，以及對於進行 API 呼叫或使用 WebSocket 的某些應用程式，您也可以使用 IAM 使用者和群組或自訂身分驗證字符來進行驗證。如需詳細資訊，請參閱IAM 使用者、群組和角色。
- 使用 AWS IoT 身分驗證時，訊息中介裝置會負責驗證您的裝置、安全地擷取裝置資料，以及使用 AWS IoT 政策授予或拒絕您為裝置指定的存取許可。
- 使用自訂身分驗證時，自訂授權方會負責驗證您的裝置，並授予或拒絕您使用 AWS IoT 或 IAM 政策為裝置指定的存取許可。
- AWS IoT 規則引擎會根據您定義的規則，將裝置資料轉送至其他裝置或其他 AWS 服務。它使用 AWS Identity and Access Management 安全地將資料傳輸到其最終目的地。如需詳細資訊，請參閱的身分和存取管理 AWS IoT。

身分驗證

身分驗證是一種機制，您可以用來驗證用戶端或伺服器的身分。伺服器身分驗證是裝置或其他用戶端確保與實際 AWS IoT 端點通訊的程序。用戶端身分驗證是裝置或其他用戶端自行進行身分驗證的程序 AWS IoT。

X.509 憑證概觀

X.509 憑證為數位憑證，其依據 X.509 公有金鑰基礎設施標準，將公有金鑰與憑證內含的身分建立關聯。X.509 憑證由稱為認證機構 (CA) 的可信任實體發行。CA 負責維護一個或多個稱為憑證授權機構憑證的特殊憑證，用以發行 X.509 憑證。僅認證機構可存取憑證授權機構憑證。X.509 憑證鏈同時用於用戶端的伺服器身分驗證，和伺服器的用戶端身分驗證。

伺服器驗證

當您的裝置或其他用戶端嘗試連線時 AWS IoT Core AWS IoT Core ，伺服器會傳送 X.509 憑證，供您的裝置用來驗證伺服器。身分驗證是透過在 TLS 層驗證 X.509 憑證鏈進行。這與您造訪 HTTPS URL 時瀏覽器所使用的方法相同。如果您要使用自己的憑證授權單位的憑證，請參閱 管理您的憑證授權機構憑證。

當您的裝置或其他用戶端建立與端點的 AWS IoT Core TLS 連線時， 會 AWS IoT Core 呈現憑證鏈，供裝置用來驗證其是否正在與 通訊 AWS IoT Core ，而非其他伺服器模擬 AWS IoT Core。顯示的鏈取決於裝置所連接端點的類型，以及用戶端在 TLS 交握期間交 AWS IoT Core 涉的密碼套件的組合。

端點類型

AWS IoT Core 支援 `iot:Data-ATS`。 `iot:Data-ATS`端點提供由 Amazon Trust Services CA 簽署的伺服器憑證。

ATS 端點呈現的憑證由 Starfield 交叉簽署。某些 TLS 用戶端實作需要驗證信任根，並要求在用戶端的信任存放區中安裝 Starfield 憑證授權機構憑證。

> **Warning**
>
> 不建議使用雜湊整個憑證 (包括發行者名稱等) 的憑證關聯方法，因為這會導致憑證驗證失敗，因為我們提供的 ATS 憑證是由 Starfield 交叉簽署，且具有不同的發行者名稱。

> **Important**
>
> 使用iot:Data-ATS端點。Symantec 和 Verisign 憑證已棄用，不再受 支援 AWS IoT Core。

您可以使用 describe-endpoint 命令建立 ATS 端點。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

describe-endpoint 命令會傳回下列格式的端點。

```
account-specific-prefix.iot.your-region.amazonaws.com
```

> **Note**
>
> 第一次呼叫 describe-endpoint 時，會建立一個端點。所有後續呼叫 describe-endpoint 會傳回相同的端點。

> **Note**
>
> 若要在 AWS IoT Core 主控台中查看您的iot:Data-ATS端點，請選擇設定。主控台只會顯示 iot:Data-ATS 端點。

IotDataPlaneClient 使用適用於 Java 的 AWS 開發套件建立

若要建立IotDataPlaneClient使用 iot:Data-ATS端點的 ，您必須執行下列動作。

- 使用 DescribeEndpoint API 建立 iot:Data-ATS 端點。
- 在建立 IotDataPlaneClient 時指定該端點。

下列範例會同時執行這些操作。

```
public void setup() throws Exception {
        IotClient client =
 IotClient.builder().credentialsProvider(CREDENTIALS_PROVIDER_CHAIN).region(Region.US_EAST_1).b
```

```
        String endpoint = client.describeEndpoint(r -> r.endpointType("iot:Data-
ATS")).endpointAddress();
        iot = IotDataPlaneClient.builder()
                                .credentialsProvider(CREDENTIALS_PROVIDER_CHAIN)
                                .endpointOverride(URI.create("https://" + endpoint))
                                .region(Region.US_EAST_1)
                                .build();
}
```

伺服器身分驗證的憑證授權機構憑證

根據您正在使用的資料端點類型，以及您已交涉的密碼套件， AWS IoT Core 伺服器身分驗證憑證會由下列其中一個根 CA 憑證簽署：

Amazon Trust Services 端點 (首選)

> **Note**
>
> 您可能需要在這些連結上按一下滑鼠右鍵，然後選取 Save link as... (另存連結為...) 以將這些憑證儲存為檔案。

- RSA 2048 位元金鑰：Amazon Root CA 1。
- RSA 4096 位元金鑰：Amazon Root CA 2。保留以供日後使用。
- ECC 256 位元金鑰：Amazon Root CA 3。
- ECC 384 位元金鑰：Amazon Root CA 4。保留以供日後使用。

這些憑證都是由 Starfield 根憑證授權機構憑證交叉簽署的。所有新的 AWS IoT Core 區域，從 2018 年 5 月 9 日 AWS IoT Core 在亞太區域 （孟買） 區域推出 開始，僅提供 ATS 憑證。

VeriSign 端點 (舊版)

- RSA 2048 位元金鑰：VeriSign 類別 3 公開主要 G5 根憑證授權機構憑證

伺服器身分驗證準則

有許多變數會影響裝置驗證 AWS IoT Core 伺服器驗證憑證的能力。例如，裝置的記憶體可能受限，無法保存所有可能的根憑證授權機構憑證，或者裝置可能實作非標準的憑證驗證方法。基於這些原因，我們建議您遵循以下準則：

- 我們建議您使用 ATS 端點並安裝所有支援的 Amazon Root CA 憑證。
- 如果您無法在裝置上存放全部這些憑證，而且您的裝置未使用 ECC 式驗證，則可以略過 Amazon Root CA 3 和 Amazon Root CA 4 4 ECC 憑證。如果您的裝置未實作 RSA 式憑證驗證，則可以略過 Amazon Root CA 1 和 Amazon Root CA 2 RSA 憑證。您可能需要在這些連結上按一下滑鼠右鍵，然後選取 Save link as... (另存連結為...) 以將這些憑證儲存為檔案。
- 如果您在連線至 ATS 端點時遇到伺服器憑證驗證問題，請嘗試將相關的交叉簽署 Amazon Root 憑證授權機構憑證新增至信任存放區。您可能需要在這些連結上按一下滑鼠右鍵，然後選取 Save link as... (另存連結為...) 以將這些憑證儲存為檔案。
  - 交叉簽署 Amazon Root CA 1
  - 交叉簽署的 Amazon Root CA 2 - 預留以供日後使用。
  - 交叉簽署 Amazon Root CA 3
  - 交叉簽署的 Amazon Root CA 4 - 預留以供日後使用。
- 如果您遇到伺服器憑證驗證問題，則您的裝置可能需要明確地信任根 CA。嘗試將 Starfield Root CA Certificate 新增到您的信任存放區。
- 如果您在執行上述步驟後仍然遇到問題，請聯絡 AWS 開發人員支援。

> **Note**
>
> 憑證授權機構憑證在過期日期之後，即無法用於驗證伺服器的憑證。憑證授權機構憑證過期日期之前，可能需要進行替代。請確認您能夠於所有裝置或用戶端更新根憑證授權機構憑證，以維持連線能力並取得最新的安全最佳實務。

> **Note**
>
> 在裝置程式碼 AWS IoT Core 中連線至 時，請將憑證傳遞至您用來連線的 API。您使用的 API 會因 SDK 而異。如需詳細資訊，請參閱 AWS IoT Core 裝置開發套件。

用戶端身分驗證

AWS IoT 支援三種類型的身分主體進行裝置或用戶端身分驗證：

- X.509 用戶端憑證
- IAM 使用者、群組和角色

- Amazon Cognito 身分

這些身分可用於裝置、行動裝置、網頁或桌面應用程式。使用者輸入 AWS IoT 命令列界面 (CLI) 命令甚至可以使用它們。一般而言， AWS IoT 裝置使用 X.509 憑證，而行動應用程式則使用 Amazon Cognito 身分。Web 和桌面應用程式則使用 IAM 或聯合身分。 AWS CLI 命令會使用 IAM。如需 IAM 身分的詳細資訊，請參閱的身分和存取管理 AWS IoT。

X.509 用戶端憑證

X.509 憑證 AWS IoT 提供驗證用戶端和裝置連線的能力。用戶端憑證必須先向 註冊 AWS IoT ，用戶端才能與 通訊 AWS IoT。您可以在相同 中的多個 AWS 帳戶中註冊用戶端憑證 AWS 區域 ，以便在同一區域中的 AWS 帳戶之間移動裝置。如需詳細資訊，請參閱 在具有多帳戶註冊的多個 AWS 帳戶中使用 X.509 用戶端憑證。

我們建議為每個裝置或用戶端提供唯一的憑證，藉此更精細地管理用戶端，包括憑證撤銷作業。裝置必須支援憑證的輪換和替代，以確保憑證過期時操作順暢。

如需使用 X.509 憑證支援數個裝置的詳細資訊，請參閱 裝置佈建，以檢閱 AWS IoT 支援的不同憑證管理和佈建選項。

AWS IoT 支援這些類型的 X.509 用戶端憑證：

- 產生的 X.509 憑證 AWS IoT
- 由向 註冊的 CA 簽署的 X.509 憑證 AWS IoT。
- 未註冊 AWS IoT且有 CA 簽署的 X.509 憑證。

本節說明如何在 AWS IoT中管理 X.509 憑證。您可以使用 AWS IoT 主控台或 AWS CLI 來執行這些憑證操作：

- 建立 AWS IoT 用戶端憑證
- 建立您自己的用戶端憑證
- 註冊用戶端憑證
- 啟用或停用用戶端憑證
- 撤銷用戶端憑證

如需執行這些操作之 AWS CLI 命令的詳細資訊，請參閱 AWS IoT CLI 參考。

使用 X.509 用戶端憑證

X.509 憑證會驗證用戶端和裝置的連線 AWS IoT。相較於其他識別和身分驗證機制，X.509 憑證具備多種好處。X.509 憑證可讓裝置使用非對稱金鑰。例如，您可以將私密金鑰燒錄到裝置上的安全儲存空間，這樣敏感的密碼編譯資料永遠不會離開裝置。由於私有金鑰絕對不會離開裝置，因此 X.509 憑證提供更勝於其他機制 (例如使用者名稱和密碼或承載符記) 的強大用戶端身分驗證。

AWS IoT 會使用 TLS 通訊協定的用戶端身分驗證模式來驗證用戶端憑證。TLS 支援多種程式設計語言與作業系統，且普遍用於加密資料。在 TLS 用戶端身分驗證中， AWS IoT 會請求 X.509 用戶端憑證，並根據憑證 AWS 帳戶 登錄檔驗證憑證的狀態。然後，它會向用戶端挑戰與憑證中包含的公有金鑰對應的私有金鑰的擁有權證明。 AWS IoT 要求用戶端將伺服器名稱指示 (SNI) 延伸模組傳送至 Transport Layer Security (TLS) 通訊協定。如需設定 SNI 延伸模組的詳細資訊，請參閱 中的傳輸安全性 AWS IoT Core。

為了促進安全且一致的用戶端連線至 AWS IoT 核心，X.509 用戶端憑證必須擁有下列項目：

- 在 AWS IoT Core 中註冊。如需詳細資訊，請參閱註冊用戶端憑證。
- 狀態為 ACTIVE。如需詳細資訊，請參閱啟用或停用用戶端憑證。
- 尚未到達憑證過期日期。

您可以建立使用 Amazon 根 CA 的用戶端憑證，也可以使用由其他憑證授權單位 (CA) 簽署的自有用戶端憑證。如需使用 AWS IoT 主控台建立使用 Amazon 根 CA 的憑證的詳細資訊，請參閱 建立 AWS IoT 用戶端憑證。如需使用您自己的 X.509 憑證的詳細資訊，請參閱 建立您自己的用戶端憑證。

對於有憑證授權機構憑證簽署的憑證，其過期日期與時間會於憑證建立時設定。產生的 X.509 憑證會在 2049 年 12 月 31 日 UTC 午夜 (2049-12-31T23：59：59Z AWS IoT 過期）。

AWS IoT Device Defender 可以對支援常見 IoT 安全最佳實務的 AWS 帳戶 和裝置執行稽核。這包括管理您的 CA 或 Amazon 根 CA 簽署的 X.509 憑證過期日期。如需管理憑證過期日期的詳細資訊，請參閱裝置憑證過期和 CA 憑證過期。

在官方 AWS IoT 部落格上，在如何使用 管理 IoT 裝置憑證輪換中，深入探索裝置憑證輪換和安全最佳實務的管理 AWS IoT。

在具有多帳戶註冊的多個 AWS 帳戶中使用 X.509 用戶端憑證

多帳戶註冊可讓您在相同區域或不同區域的 AWS 帳戶之間移動裝置。您可以在生產前帳戶中註冊、測試和設定裝置，然後在生產帳戶中註冊並使用相同的裝置和裝置憑證。您也可以在裝置上註冊用戶端憑

證，或在沒有註冊 CA 的情況下註冊裝置憑證 AWS IoT。如需詳細資訊，請參閱註冊未註冊 CA 所簽署的用戶端憑證 (CLI)。

> **Note**
>
> iot:Data-ATS、iot:Data (舊式)、iot:Jobs 及 iot:CredentialProvider 端點類型上支援用於多帳戶註冊的憑證。如需 AWS IoT 裝置端點的詳細資訊，請參閱 AWS IoT 裝置資料和服務端點。

使用多帳戶註冊的裝置必須將伺服器名稱指示 (SNI) 擴充功能傳送至 Transport Layer Security (TLS) 通訊協定，並在連線至 時在 host_name 欄位中提供完整的端點地址 AWS IoT。 AWS IoT 會使用 中的端點地址，將連線host_name路由至正確的 AWS IoT 帳戶。未在 host_name 中傳送有效端點位址的現有裝置將繼續運作，但無法使用需要此資訊的功能。如需 SNI 延伸模組的詳細資訊，以及如何識別 host_name 欄位的端點位址，請參閱 中的傳輸安全性 AWS IoT Core。

使用多帳戶註冊

1. 您可以在有 CA 的情況下註冊裝置憑證。您可以在 SNI_ONLY 模式下在多個帳戶中註冊簽署 CA，並使用該 CA 將相同的用戶端憑證註冊到多個帳戶。如需詳細資訊，請參閱在 SNI_ONLY 模式下註冊 CA 憑證 (CLI) - 建議。
2. 您可以在沒有 CA 的情況下註冊裝置憑證。請參閱 註冊未註冊 CA 所簽署的用戶端憑證 (CLI)。註冊 CA 非必須。您不需要註冊簽署裝置憑證的 CA AWS IoT。

支援的憑證簽署演算法 AWS IoT

AWS IoT 支援下列憑證簽署演算法：

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- SHA256WITHRSAANDMGF1 (RSASSA-PSS)
- SHA384WITHRSAANDMGF1 (RSASSA-PSS)
- SHA512WITHRSAANDMGF1 (RSASSA-PSS)
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256

- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

如需憑證身分驗證和安全性的詳細資訊，請參閱裝置憑證金鑰品質。

> **Note**
>
> 憑證簽署請求 (CSR) 必須包含公開金鑰。該金鑰可以是公開金鑰 (長度至少 2,048 位元) 或 ECC 金鑰 (至少 NIST P-256、NIST P-384 或 NIST P-521 曲線)。如需詳細資訊，請參閱《AWS IoT API 參考指南》中的 CreateCertificateFromCsr。

支援的金鑰演算法 AWS IoT

下表顯示如何支援金鑰演算法：

金鑰演算法	憑證簽署演算法	TLS 版本控制	支援？ 是或否
金鑰大小至少為 2048 位元的 RSA	全部	TLS 1.2 TLS 1.3	是
ECC NIST P-256/P-384/ P-521	全部	TLS 1.2 TLS 1.3	是
金鑰大小至少為 2048 位元的 RSA-PSS	全部	TLS 1.2	否
金鑰大小至少為 2048 位元的 RSA-PSS	全部	TLS 1.3	是

若要使用 CreateCertificateFromCSR 建立憑證，您可以使用支援的金鑰演算法來產生 CSR 的公有金鑰。若要使用 RegisterCertificate 或 RegisterCertificateWithoutCA 註冊您自己的憑證，您可以使用支援的金鑰演算法來產生憑證的公有金鑰。

如需詳細資訊，請參閱 安全政策。

建立 AWS IoT 用戶端憑證

AWS IoT 提供由 Amazon 根憑證授權機構 (CA) 簽署的用戶端憑證。

本主題說明如何建立由 Amazon 根憑證授權機構簽署的用戶端憑證，以及憑證檔案的下載方式。建立用戶端憑證檔案之後，您必須將它們安裝在用戶端上。

> **Note**
>
> 提供的每個 X.509 用戶端憑證都會 AWS IoT 保留您在憑證建立時設定的發行者和主體屬性。只在建立憑證之後，憑證屬性才是不可變的。

您可以使用 AWS IoT 主控台或 AWS CLI 來建立由 Amazon 根 AWS IoT 憑證授權單位簽署的憑證。

建立 AWS IoT 憑證 （ 主控台 ）

使用 AWS IoT 主控台建立 AWS IoT 憑證

1. 登入 AWS 管理主控台 並開啟 AWS IoT 主控台。
2. 在導覽窗格中，依序選擇安全、憑證，然後選擇建立。
3. 選擇 One-click certificate creation (一鍵建立憑證) - Create certificate (建立憑證)。
4. 從已建立的憑證頁面上，將該物件的用戶端憑證檔案、公開金鑰和私密金鑰下載到安全的位置。產生的這些憑證只能與 AWS IoT AWS IoT 服務搭配使用。

   如果您還需要 Amazon 根 CA 憑證檔案，此頁面上也有可下載該憑證的頁面連結。
5. 現在已建立用戶端憑證，並已註冊 AWS IoT。您必須先啟動憑證，才能在用戶端中使用憑證。

   若要立即啟用用戶端憑證，請選擇啟用。如果您不想立即啟用憑證，請參閱 啟動用戶端憑證 (主控台) 以了解如何稍後再啟用憑證。
6. 如果您要將政策連接至憑證，請選擇 Attach a policy (連接政策)。

   如果您不想立即連接政策，請選擇 Done (完成) 來完成。您可以稍後附加政策。

完成程序之後，請在用戶端上安裝憑證檔案。

建立 AWS IoT 憑證 (CLI)

AWS CLI 提供 create-keys-and-certificate命令來建立由 Amazon 根憑證授權單位簽署的用戶端憑證。但是，此命令不會下載 Amazon 根憑證授權機構憑證檔案。您可以從 伺服器身分驗證的憑證授權機構憑證 下載 Amazon 根 CA 憑證檔案。

此命令會建立私有金鑰、公有金鑰和 X.509 憑證檔案，並使用 註冊和啟用憑證。 AWS IoT

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

如果您不想在建立並註冊憑證時啟動憑證，此命令會建立私有金鑰、公有金鑰和 X.509 憑證檔案並註冊憑證，但不會啟動憑證。啟動用戶端憑證 (CLI) 描述稍後如何啟用憑證。

```
aws iot create-keys-and-certificate \
    --no-set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

在用戶端上安裝憑證檔案。

建立您自己的用戶端憑證

AWS IoT 支援由任何根憑證或中繼憑證授權機構 (CA) 簽署的用戶端憑證。 AWS IoT 會使用 CA 憑證來驗證憑證的擁有權。若要使用由非 Amazon CA 的 CA 簽署的裝置憑證，CA 的憑證必須向 註冊，AWS IoT 以便我們驗證裝置憑證的擁有權。

AWS IoT 支援多種自帶憑證 (BYOC) 的方式：

- 首先，註冊用於簽署用戶端憑證的 CA，然後註冊個別的用戶端憑證。如果您想要在裝置或用戶端首次連線至其用戶端憑證 AWS IoT （ 也稱為Just-in-Time佈建) 時註冊裝置或用戶端，您必須向 AWS IoT 註冊簽署 CA，並啟用自動註冊。
- 如果您無法註冊簽署 CA，則可以選擇不使用 CA 註冊用戶端憑證。對於不使用 CA 註冊的裝置，在將它們連接到 AWS IoT時需要出示伺服器名稱指示 (SNI)。

> **Note**
>
> 若要使用 CA 註冊用戶端憑證，您必須向 註冊簽署 CA AWS IoT，而不是階層中的任何其他 CAs。

> **Note**
>
> 一個 CA 憑證只能由一個區域中的一個帳戶在 DEFAULT 模式下註冊。一個 CA 憑證可以由一個區域中的多個帳戶在 SNI_ONLY 模式下註冊。

如需有關使用 X.509 憑證支援數個以上裝置的詳細資訊，請參閱 裝置佈建，以檢閱 AWS IoT 支援的不同憑證管理和佈建選項。

主題

- 管理您的憑證授權機構憑證
- 使用您的憑證授權機構憑證建立用戶端憑證

管理您的憑證授權機構憑證

本節說明管理您自己的憑證授權機構 (CA) 憑證的一般工作。

AWS IoT 如果您使用的用戶端憑證是由 AWS IoT 無法辨識的 CA 簽署，則可以向 註冊憑證授權機構 (CA)。

如果您希望用戶端在第一次連線 AWS IoT 時自動向 註冊其用戶端憑證，則必須向簽署用戶端憑證的 CA 註冊 AWS IoT。否則，您不需要註冊已簽署用戶端憑證的憑證授權機構憑證。

> **Note**
>
> 一個 CA 憑證只能由一個區域中的一個帳戶在 DEFAULT 模式下註冊。一個 CA 憑證可以由一個區域中的多個帳戶在 SNI_ONLY 模式下註冊。

主題：

- 建立憑證授權機構憑證
- 註冊您的憑證授權機構憑證
- 停用憑證授權機構憑證

建立憑證授權機構憑證

如果沒有 憑證授權機構憑證，您可以使用 OpenSSL v1.1.1i 工具來建立憑證授權機構憑證。

> **Note**
>
> 您無法在 AWS IoT 主控台中執行此程序。

使用 OpenSSL v1.1.1i 工具來建立憑證授權機構憑證

1. 產生金鑰對。

```
openssl genrsa -out root_CA_key_filename.key 2048
```

2. 透過該金鑰對的私有金鑰來產生憑證授權機構憑證。

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename.key \
    -sha256 -days 1024 \
    -out root_CA_cert_filename.pem
```

註冊您的憑證授權機構憑證

這些程序說明如何從不是 Amazon CA 的憑證授權機構 (CA) 註冊憑證。 AWS IoT Core 會使用 CA 憑證來驗證憑證的擁有權。若要使用由非 Amazon CA 的 CA 簽署的裝置憑證，您必須向 註冊 CA 憑證， AWS IoT Core 以便驗證裝置憑證的擁有權。

註冊憑證授權機構憑證 (主控台)。

> **Note**
>
> 若要在主控台中註冊憑證授權機構憑證，請在主控台中的註冊憑證授權機構憑證開始。您可在多帳戶模式下註冊您的 CA，而無需提供驗證憑證或私有金鑰的存取權。一個 CA 可由相同 AWS 區域中的多個 AWS 帳戶 以多帳戶模式進行註冊。您可透過提供驗證憑證和 CA 私有金鑰的擁有權證明，以單一帳戶模式註冊您的 CA。

註冊憑證授權機構憑證 (CLI)

您可以在 DEFAULT 模式或 SNI_ONLY 模式下註冊 CA 憑證。CA 可以在 DEFAULT 中以 模式註冊 AWS 帳戶 AWS 區域。CA 可以由相同 AWS 帳戶 中的多個 以 SNI_ONLY 模式註冊 AWS 區域。如需 CA 憑證模式的詳細資訊，請參閱 certificateMode。

> **Note**
>
> 我們建議您在 SNI_ONLY 模式下註冊 CA。您不需要提供驗證憑證或存取私有金鑰，而且您可以在相同 AWS 帳戶 中透過多個 註冊 CA AWS 區域。

在 SNI_ONLY 模式下註冊 CA 憑證 (CLI) - 建議

先決條件

繼續操作之前，請確定您的電腦上具有下列資訊：

- 根 CA 憑證檔案 (於下列範例中引用為 *root_CA_cert_filename.pem*)
- OpenSSL v1.1.1i 或更新版本

使用 在 **SNI_ONLY** 模式下註冊 CA 憑證 AWS CLI

1. 向 註冊 CA 憑證 AWS IoT。使用 register-ca-certificate 命令，輸入 CA 憑證檔案名稱。如需詳細資訊，請參閱 AWS CLI 命令參考中的 register-ca-certificate。

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --certificate-mode SNI_ONLY
   ```

   若成功，此命令會傳回 *certificateId*。

2. 此時，CA 憑證已向 註冊， AWS IoT 但處於非作用中狀態。CA 憑證必須處於作用中狀態，您才能註冊已簽署的任何用戶端憑證。

   這個步驟會啟動 CA 憑證。

   若要啟用該 CA 憑證，請按如下所示使用 update-certificate 命令。如需詳細資訊，請參閱 AWS CLI 命令參考中的 update-certificate。

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

若要查看 CA 憑證的狀態，請使用 describe-ca-certificate 命令。如需詳細資訊，請參閱 AWS CLI 命令參考中的 describe-ca-certificate。

在 **DEFAULT** 模式下註冊 CA 憑證 (CLI)

先決條件

繼續操作之前，請確定您的電腦上具有下列資訊：

- 根 CA 憑證檔案 (於下列範例中引用為 *root_CA_cert_filename.pem*)
- 根 CA 憑證私有金鑰檔案 (於下列範例中引用為 *root_CA_key_filename.key*)
- OpenSSL v1.1.1i 或更新版本

使用 在 **DEFAULT** 模式下註冊 CA 憑證 AWS CLI

1. 若要從 取得註冊碼 AWS IoT，請使用 get-registration-code。將傳回的 `registrationCode` 儲存為私有金鑰驗證憑證的 Common Name。如需詳細資訊，請參閱 AWS CLI 命令參考中的 get-registration-code。

   ```
   aws iot get-registration-code
   ```

2. 產生私有金鑰驗證憑證的金鑰對：

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

3. 建立私密金鑰驗證憑證的憑證簽署要求 (CSR)。將憑證的 Common Name 欄位設定為 get-registration-code 傳回的 `registrationCode`。

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   將顯示提示要求您輸入一些此憑證的資訊，包括 Common Name。

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   ```

```
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

4. 使用 CSR 建立私有金鑰驗證憑證：

```
openssl x509 -req \
    -in verification_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out verification_cert_filename.pem \
    -days 500 -sha256
```

5. 向 註冊 CA 憑證 AWS IoT。將 CA 憑證檔案名稱和私有金鑰驗證憑證檔案名稱傳遞給 register-ca-certificate 命令，如下所示。如需詳細資訊，請參閱 AWS CLI 命令參考中的 register-ca-certificate。

```
aws iot register-ca-certificate \
    --ca-certificate file://root_CA_cert_filename.pem \
    --verification-cert file://verification_cert_filename.pem
```

   如果成功，此命令會傳回 *certificateId*。

6. 此時，CA 憑證已向 註冊， AWS IoT 但未處於作用中狀態。CA 憑證必須處於作用中狀態，您才能註冊已簽署的任何用戶端憑證。

   這個步驟會啟動 CA 憑證。

   若要啟用該 CA 憑證，請按如下所示使用 update-certificate 命令。如需詳細資訊，請參閱 AWS CLI 命令參考中的 update-certificate。

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

若要查看 CA 憑證的狀態，請使用 describe-ca-certificate 命令。如需詳細資訊，請參閱 AWS CLI 命令參考中的 describe-ca-certificate。

建立憑證授權機構驗證憑證以在主控台中註冊憑證授權機構憑證

> **Note**
>
> 此程序僅適用於從 AWS IoT 主控台註冊 CA 憑證的情況。
> 如果您沒有從 AWS IoT 主控台前往此程序，請在 主控台的註冊 CA 憑證啟動 CA 憑證註冊程序。

繼續操作之前，請確定您在同一台電腦上具有下列可用項目：

- 根 CA 憑證檔案 (於下列範例中引用為 *root_CA_cert_filename.pem*)
- 根 CA 憑證私有金鑰檔案 (於下列範例中引用為 *root_CA_key_filename.key*)
- OpenSSL v1.1.1i 或更新版本

如要使用命令列介面來建立 CA 驗證憑證，以於主控台中註冊您的 CA 憑證

1. 將 *verification_cert_key_filename.key* 替換為您要建立的驗證憑證金鑰檔案名稱 (例如 **verification_cert.key**)。然後執行此命令以產生私有金鑰驗證憑證的金鑰對：

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

2. 將 *verification_cert_key_filename.key* 替換為您在步驟 1 建立的金鑰檔案名稱。

   verification_cert_csr_filename.csr 替換為您要建立的憑證簽署請求 (CSR) 檔案名稱。例如 **verification_cert.csr**。

   執行此命令以建立 CSR 檔案。

   ```
   openssl req -new \
   ```

```
    -key verification_cert_key_filename.key \
    -out verification_cert_csr_filename.csr
```

命令會提示您輸入其他資訊 (稍後會說明)。

3. 在 AWS IoT 主控台的驗證憑證容器中，複製註冊碼。
4. openssl 命令提示您輸入的資訊如下列範例所示。除了 Common Name 欄位，您可輸入自己的值或將其留空。

   於 Common Name 欄位中，請貼上您在上一個步驟中複製的註冊碼。

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

   完成後，命令將建立 CSR 檔案。

5. *verification_cert_csr_filename.csr* 替換為您在上一個步驟中使用的 *verification_cert_csr_filename.csr*。

   將 *root_CA_cert_filename.pem* 替換為您要註冊的 CA 憑證檔案名稱。

   將 *root_CA_key_filename.key* 替換為 CA 憑證私有金鑰檔案的檔案名稱。

   將 *verification_cert_filename.pem* 替換為您要建立的驗證憑證檔案名稱。例如 **verification_cert.pem**。

```
openssl x509 -req \
    -in verification_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out verification_cert_filename.pem \
    -days 500 -sha256
```

6. OpenSSL 命令完成後，您應準備好這些檔案，以供返回主控台時使用。

   - 您的憑證授權機構憑證檔案 (在上一個命令中使用的 *root_CA_cert_filename.pem*)
   - 您在上一個步驟中建立的驗證憑證 (用於上一個命令中的 *verification_cert_filename.pem*)

停用憑證授權機構憑證

當憑證授權機構 (CA) 憑證啟用自動用戶端憑證註冊時， 會 AWS IoT 檢查 CA 憑證，以確保 CA 為 ACTIVE。如果 CA 憑證為 INACTIVE， AWS IoT 則不允許註冊用戶端憑證。

藉由將 CA 憑證設定為 INACTIVE，您可防止自動註冊 CA 所核發的任何新用戶端憑證。

> **Note**
>
> 由有風險之憑證授權機構憑證簽署的任何已註冊裝置憑證將持續運作，直到您確實逐一撤銷這些憑證為止。

停用憑證授權機構憑證 (主控台)

使用 AWS IoT 主控台停用 CA 憑證

1. 登入 AWS 管理主控台 並開啟 AWS IoT 主控台。
2. 請在左側導覽窗格中，選擇 Secure (安全)、CAs (CA)。
3. 於憑證授權單位清單中，找出您要停用的憑證授權單位，接著選擇省略號圖示開啟選項選單。
4. 在選項選單上，選擇 Deactivate (停用)。

憑證授權機構應在清單中顯示為 Inactive (非作用中)。

> **Note**
>
> AWS IoT 主控台不提供列出您停用之 CA 簽署的憑證的方法。如需列出這些憑證的 AWS CLI 選項，請參閱 停用憑證授權機構憑證 (CLI)。

停用憑證授權機構憑證 (CLI)

AWS CLI 提供停用 CA 憑證的 update-ca-certificate命令。

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

使用 list-certificates-by-ca 命令，取得由指定 CA 簽署的所有已註冊裝置憑證清單。針對由指定憑證授權機構憑證所簽署的每一裝置憑證，請使用 update-certificate 命令來撤銷該裝置憑證，以避免受到使用。

使用 describe-ca-certificate 命令來查看 CA 憑證的狀態。

使用您的憑證授權機構憑證建立用戶端憑證

您可以使用自己的憑證授權機構 (CA) 來建立用戶端憑證。用戶端憑證必須先向 註冊， AWS IoT 才能使用。如需用戶端憑證之註冊選項的詳細資訊，請參閱 註冊用戶端憑證。

建立用戶端憑證 (CLI)

> **Note**
>
> 您無法在 AWS IoT 主控台中執行此程序。

使用 建立用戶端憑證 AWS CLI

1. 產生金鑰對。

   ```
   openssl genrsa -out device_cert_key_filename.key 2048
   ```

2. 建立用戶端憑證的 CSR。

```
openssl req -new \
    -key device_cert_key_filename.key \
    -out device_cert_csr_filename.csr
```

將出現提示，要求您輸入一些資訊，如下所示：

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

3. 從 CSR 建立用戶端憑證。

```
openssl x509 -req \
    -in device_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out device_cert_filename.pem \
    -days 500 -sha256
```

此時，用戶端憑證已建立，但尚未註冊 AWS IoT。如需如何和何時註冊用戶端憑證的詳細資訊，請參閱 註冊用戶端憑證。

註冊用戶端憑證

用戶端憑證必須向 註冊 AWS IoT ，才能啟用用戶端與 之間的通訊 AWS IoT。您可以手動註冊每個用戶端憑證，也可以將用戶端憑證設定為 AWS IoT 在用戶端第一次連線至 時自動註冊。

如果想要您的用戶端和裝置在第一次連線時註冊，您必須使用 註冊您的憑證授權機構憑證，在要使用用戶端憑證的區域中透過 AWS IoT 來簽署該用戶端憑證。Amazon 根 CA 會自動向 註冊 AWS IoT。

用戶端憑證可由 AWS 帳戶 和 區域共用。這些主題中的程序必須在您要使用用戶端憑證的每個帳戶和區域中執行。在一個帳戶或區域中註冊用戶端憑證時，另一個帳戶或區域不會自動加以辨識。

> **Note**
>
> 使用 Transport Layer Security (TLS) 通訊協定來連線 AWS IoT 的用戶端必須支援 TLS 的伺服器名稱指示 (SNI) 延伸。如需詳細資訊，請參閱中的傳輸安全性 AWS IoT Core。

主題

- 手動註冊用戶端憑證
- 在用戶端連線至 AWS IoT just-in-time註冊 (JITR) 時註冊用戶端憑證

手動註冊用戶端憑證

您可以使用 AWS IoT 主控台和 手動註冊用戶端憑證 AWS CLI。

要使用的註冊程序取決於 AWS 帳戶和 區域是否會共用憑證。在一個帳戶或區域中註冊用戶端憑證時，另一個帳戶或區域不會自動加以辨識。

本主題中的程序必須在您要使用用戶端憑證的每個帳戶和區域中執行。用戶端憑證可以由 AWS 帳戶和區域共用。

註冊已註冊 CA 所簽署的用戶端憑證 (主控台)

> **Note**
>
> 執行此程序之前，請確定您擁有用戶端憑證的 .pem 檔案，且用戶端憑證是由您已註冊 AWS IoT的 CA 簽署。

AWS IoT 使用主控台向 註冊現有憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 請在導覽窗格的 Manage (管理) 區段中，選擇 Security (安全性)，然後選擇 Certificates (憑證)。
3. 在 Certificates (憑證) 頁面的 Certificates (憑證) 對話方塊中，選擇 Add certificate (新增憑證)，然後選擇 Register certificates (註冊憑證)。
4. 在 Register certificate (登錄憑證) 頁面的 Certificates to upload (要上傳的憑證) 對話方塊中執行下列操作：

   - 請選擇 CA is registered with AWS IoT (已用註冊 CA)。
   - 從 Choose a CA certificate (選擇 CA 憑證) 中，選取您的 Certification authority (憑證授權機構)。
     - 選擇 Register a new CA (註冊新的憑證) 以註冊未使用 AWS IoT註冊的新 Certification authority (憑證授權機構)。
     - 若 Amazon Root certificate authority (Amazon 根憑證授權機構) 是您的憑證授權機構，Choose a CA certificate (選擇 CA 憑證) 請保留空白。
   - 最多選取要上傳和註冊的 10 個憑證 AWS IoT。
     - 請選擇您在 建立 AWS IoT 用戶端憑證 和 使用您的憑證授權機構憑證建立用戶端憑證 中建立的憑證檔案。
   - 請選擇 Activate (啟動) 或 Deactivate (停用)。若您選擇 Deactive (停用)，啟用或停用用戶端憑證 會說明如何在註冊憑證後啟動憑證。
   - 選擇註冊。

您已註冊的憑證將會出現在 Certificates (憑證) 頁面的 Certificates (憑證) 對話方塊中。

註冊未註冊 CA 所簽署的用戶端憑證 (主控台)

> **Note**
>
> 執行此程序之前，請確定您擁有用戶端憑證的 .pem 檔案。

AWS IoT 使用主控台向 註冊現有憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)、選擇 Certificates (憑證)，然後選擇 Create (建立)。

3. 在 Create a certificate (建立憑證) 上，找出 Use my certificate (使用我的憑證) 項目，然後選擇 Get started (開始使用)。
4. 在 Select a CA (選取 CA) 上，選擇 Next (下一步)。
5. 在 Register existing device certificates (註冊現有裝置憑證) 上，選擇 Select certificates (選取憑證)，然後選取最多 10 個要註冊的憑證檔案。
6. 關閉檔案對話方塊後，請選取您要在註冊用戶端憑證時啟用，還是撤銷用戶端憑證。

   如果您未在註冊憑證時啟用憑證，啟動用戶端憑證 (主控台) 描述稍後如何啟用憑證。

   如果憑證在註冊時遭到撤銷，則無法稍後啟用。

   選擇要註冊的憑證檔案，並選取註冊後要採取的動作之後，請選取 Register certificates (註冊憑證)。

已成功註冊的用戶端憑證會出現在憑證清單中。

註冊已註冊 CA 所簽署的用戶端憑證 (CLI)

> **Note**
>
> 執行此程序之前，請確定您擁有憑證授權機構 (CA) .pem 和用戶端憑證的 .pem 檔案。用戶端憑證必須由您已註冊 AWS IoT的憑證授權機構 (CA) 簽署。

使用 register-certificate 命令來註冊 (但不是啟動) 用戶端憑證。

```
aws iot register-certificate \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

用戶端憑證已向 註冊 AWS IoT，但尚未啟用。如需稍後如何啟用用戶端憑證的詳細資訊，請參閱 啟動用戶端憑證 (CLI)。

當您使用這個命令註冊用戶端憑證時，也可以啟動用戶端憑證。

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_filename.pem \
```

```
    --ca-certificate-pem file://ca_cert_filename.pem
```

如需啟用憑證使其可用於連線的詳細資訊 AWS IoT，請參閱 啟用或停用用戶端憑證

註冊未註冊 CA 所簽署的用戶端憑證 (CLI)

> **Note**
>
> 執行此程序之前，請確定您擁有憑證的 .pem 檔案。

使用 register-certificate-without-ca 命令來註冊 (但不是啟動) 用戶端憑證。

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_filename.pem
```

用戶端憑證已向 註冊 AWS IoT，但尚未啟用。如需稍後如何啟用用戶端憑證的詳細資訊，請參閱 啟動用戶端憑證 (CLI)。

當您使用這個命令註冊用戶端憑證時，也可以啟動用戶端憑證。

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_filename.pem
```

如需啟用憑證使其可用於連線的詳細資訊 AWS IoT，請參閱 啟用或停用用戶端憑證。

在用戶端連線至 AWS IoT just-in-time註冊 (JITR) 時註冊用戶端憑證

您可以設定 CA 憑證，以啟用其簽署的用戶端憑證，以便在用戶端第一次連線時 AWS IoT 自動向 註冊 AWS IoT。

若要在用戶端 AWS IoT 第一次連線至 時註冊用戶端憑證，您必須啟用 CA 憑證以進行自動註冊，並設定用戶端提供所需憑證的第一個連線。

設定憑證授權機構憑證以支援自動註冊 (主控台)

使用 AWS IoT 主控台設定 CA 憑證以支援自動用戶端憑證註冊

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 請在左側導覽窗格中，選擇 Secure (安全)、CAs (CA)。

3. 在憑證授權單位清單中，尋找您要啟用自動註冊的憑證授權單位，然後使用省略符號圖示開啟選項選單。
4. 在選項選單上，選擇 Enable auto-registration (啟用自動註冊)。

> **Note**
>
> 自動註冊狀態不會顯示在憑證授權單位清單中。若要查看憑證授權機構的自動註冊狀態，您必須開啟憑證授權機構的 Details (詳細資料) 頁面。

設定憑證授權機構憑證以支援自動註冊 (CLI)

如果您已向 註冊 CA 憑證 AWS IoT，請使用 update-ca-certificate命令將 CA 憑證`autoRegistrationStatus`設定為 `ENABLE`。

```
aws iot update-ca-certificate \
--certificate-id caCertificateId \
--new-auto-registration-status ENABLE
```

如果您想要在註冊憑證授權機構憑證時啟用 `autoRegistrationStatus`，請使用 register-ca-certificate 命令。

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_cert_filename.pem \
--verification-cert file://verification_cert_filename.pem
```

使用 describe-ca-certificate 命令來查看 CA 憑證的狀態。

設定用戶端的首次連線進行自動註冊

當用戶端 AWS IoT 第一次嘗試連線到 時，在 Transport Layer Security (TLS) 交握期間，用戶端上必須存在由 CA 憑證簽署的用戶端憑證。

當用戶端連線到 時 AWS IoT，請使用您在建立用戶端憑證或建立您自己的用戶端憑證中建立的 AWS IoT用戶端憑證。 會將 CA 憑證 AWS IoT 辨識為已註冊的 CA 憑證、註冊用戶端憑證，並將其狀態設定為 `PENDING_ACTIVATION`。 https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html這代表系統會自動註冊該用戶端憑證，並準備啟用。用戶端憑證的狀態必須先是 `ACTIVE` 才能用來連線 AWS IoT。請參閱 啟用或停用用戶端憑證 以了解啟用用戶端憑證的相關資訊。

> **Note**
>
> 您可以使用 AWS IoT Core just-in-time(JITR) 功能佈建裝置，而不必在裝置的第一個連線上傳送整個信任鏈 AWS IoT Core。雖不要求顯示憑證授權機構憑證，但需要裝置在連線時傳送伺服器名稱指示 (SNI) 延伸。

當 AWS IoT 自動註冊憑證或用戶端呈現 PENDING_ACTIVATION 狀態的憑證時， 會將訊息 AWS IoT 發佈至下列 MQTT 主題：

`$aws/events/certificates/registered/`*`caCertificateId`*

`caCertificateId` 為發行該用戶端憑證的憑證授權機構憑證 ID。

發佈至此主題的訊息具有以下結構：

```
{
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

您可以建立規則以監聽此主題，並執行部分動作。我們建議您建立 Lambda 規則，此規則可驗證用戶端憑證未列於憑證撤銷清單 (CRL) 中，同時啟用憑證，並建立和連接政策至憑證。政策會決定用戶端能夠存取哪些資源。如果您要建立的政策需要連線裝置的用戶端 ID，您可以使用規則的 clientid() 函數來擷取用戶端 ID。範例規則定義可能如下所示：

```
SELECT *,
   clientid() as clientid
from $aws/events/certificates/registered/caCertificateId
```

在此範例中，規則會訂閱 JITR 主題`$aws/events/certificates/registered/`*`caCertificateID`*，並使用 clientid() 函數來擷取用戶端 ID。然後，規則會將用戶端 ID 附加至 JITR 承載。如需規則 clientid() 函數的詳細資訊，請參閱 clientid()。

如需如何建立會接聽 `$aws/events/certificates/registered/`*`caCertificateID`* 主題並執行這些動作之 Lambda 規則的詳細資訊，請參閱 AWS IoT上用戶端憑證的即時註冊。

如果在自動註冊用戶端憑證期間發生任何錯誤或例外狀況， 會將事件或訊息 AWS IoT 傳送至 CloudWatch Logs 中的日誌。如需為您帳戶設定記錄的詳細資訊，請參閱 Amazon CloudWatch 文件。

管理用戶端憑證

AWS IoT 為您提供管理用戶端憑證的功能。

在本主題中：

- 啟用或停用用戶端憑證
- 將物件或政策連接至用戶端憑證
- 撤銷用戶端憑證
- 將憑證傳輸至另一個帳戶

啟用或停用用戶端憑證

AWS IoT 驗證用戶端憑證在驗證連線時是否處於作用中狀態。

您可以建立和註冊用戶端憑證而不需啟用它們，這樣在您想要使用前，它們就不會被使用。您也可以停用作用中的用戶端憑證，以暫時停用這些憑證。最後，您可以撤銷用戶端憑證，以防止未來使用這些憑證。

啟動用戶端憑證 (主控台)

使用 AWS IoT 主控台啟用用戶端憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。
3. 在憑證清單中，找出您要啟用的憑證，然後使用省略符號圖示開啟選項選單。
4. 在選項選單中，選擇 Activate (啟動)。

憑證在憑證清單應顯示為 Active (作用中)。

停用用戶端憑證 (主控台)

使用 AWS IoT 主控台停用用戶端憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。

3. 在憑證清單中，找出您要停用的憑證，然後使用省略符號圖示開啟選項選單。
4. 在選項選單中，選擇 Deactivate (停用)。

憑證在憑證清單中應顯示為 Inactive (非作用中)。

啟動用戶端憑證 (CLI)

AWS CLI 提供 update-certificate命令來啟用憑證。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

如果命令成功，則憑證的狀態將會是 ACTIVE。執行 describe-certificate 以查看憑證的狀態。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

停用用戶端憑證 (CLI)

AWS CLI 提供停用憑證的 update-certificate命令。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

如果命令成功，則憑證的狀態將會是 INACTIVE。執行 describe-certificate 以查看憑證的狀態。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

將物件或政策連接至用戶端憑證

當您建立和註冊與 AWS IoT 物件分開的憑證時，它不會有任何可授權任何 AWS IoT 操作的政策，也不會與任何 AWS IoT 物件相關聯。本節說明如何在已註冊憑證上新增這些關聯。

> **Important**
>
> 要完成這些程序，您必須先建立要連接至憑證的物件或政策。

憑證會使用 驗證裝置， AWS IoT 使其可以連線。將憑證連接至物件資源會在裝置 (透過憑證) 與物件資源之間建立關係。若要授權裝置執行 AWS IoT 動作，例如允許裝置連接和發佈訊息，必須將適當的政策連接到裝置的憑證。

將物件連接至用戶端憑證 (主控台)

您需要物件名稱才能完成此程序。

將物件連接至已註冊憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。
3. 在憑證清單中，找出要連接政策的憑證、選擇省略符號圖示來開啟憑證的選項選單，然後選擇 Attach thing (連接物件)。
4. 在彈出式視窗中，找出要連接至憑證的物件名稱、選擇其核取方塊，然後選擇 Attach (連接)。

物件現應出現在憑證詳細資訊頁面上的物件清單中。

將政策連接至用戶端憑證 (主控台)

您需要政策物件名稱才能完成此程序。

將政策物件連接至已註冊憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。
3. 在憑證清單中，找出要連接政策的憑證、選擇省略符號圖示來開啟憑證的選項選單，然後選擇 Attach policy (連接政策)。
4. 在彈出式視窗中，找出要連接至憑證的政策名稱、選擇其核取方塊，然後選擇 Attach (連接)。

政策物件現應出現在憑證詳細資訊頁面上的政策清單中。

將物件連接至用戶端憑證 (CLI)

AWS CLI 提供將物件連接到憑證的attach-thing-principal命令。

```
aws iot attach-thing-principal \
    --principal certificateArn \
```

```
    --thing-name thingName
```

將政策連接至用戶端憑證 (CLI)

AWS CLI 提供 attach-policy命令，以將政策物件連接至憑證。

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

撤銷用戶端憑證

如果您在已註冊的用戶端憑證上偵測到可疑活動，您可以撤銷該憑證，使其無法再次使用。

> **Note**
>
> 憑證一旦撤銷，就無法變更其狀態。也就是說，憑證狀態無法變更為 `Active` 或任何其他狀態。

撤銷用戶端憑證 (主控台)

使用 AWS IoT 主控台撤銷用戶端憑證

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。
3. 在憑證清單中，找出您要撤銷的憑證，然後使用省略符號圖示開啟選項選單。
4. 在選項選單中，選擇 Revoke (撤銷)。

如果憑證已成功撤銷，它會在憑證清單中顯示為 Revoked (已撤銷)。

撤銷用戶端憑證 (CLI)

AWS CLI 提供撤銷憑證的 update-certificate命令。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

如果命令成功，則憑證的狀態將會是 `REVOKED`。執行 describe-certificate 以查看憑證的狀態。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

將憑證傳輸至另一個帳戶

屬於某個憑證的 X.509 憑證 AWS 帳戶 可以轉移到另一個憑證 AWS 帳戶。

將 X.509 憑證從一個憑證轉移到 AWS 帳戶 另一個憑證

1. the section called “開始傳輸憑證”

   在起始傳輸之前，必須先停用憑證，並將其與所有政策和物件分開。
2. the section called “接受或拒絕憑證傳輸”

   接收帳戶必須明確地接受或拒絕傳輸的憑證。在接收帳戶接受憑證之後，必須先啟用憑證才能使用。
3. the section called “取消憑證傳輸”

   如果憑證未被接受，原始帳戶可以取消傳輸。

開始傳輸憑證

您可以使用 AWS IoT 主控台或 ，開始 AWS 帳戶 將憑證轉移到另一個憑證 AWS CLI。

開始傳輸憑證 (主控台)

若要完成此程序，將需要您想要傳輸之憑證的 ID。

從具有要傳輸之憑證的帳戶執行此程序。

開始將憑證轉移到另一個憑證 AWS 帳戶

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。

   選擇您想要傳輸且狀態為 Active (作用中) 或 Inactive (非作用中) 的憑證，並開啟其詳細資訊頁面。
3. 在憑證 Details (詳細資訊) 頁面的 Actions (動作) 選單中，如果 Deactivate (停用) 選項可用，請選擇 Deactivate (停用) 選項來停用憑證。
4. 在憑證的 Details (詳細資訊) 頁面的左側選單中，選擇 Policies (政策)。

5. 在憑證的 Policies (政策) 頁面上，如果有任何政策連接至憑證，請開啟政策的選項選單並選擇 Detach (分開)，來分開每個政策。

   在您繼續之前，憑證不得有任何連接的政策。
6. 在憑證的 Policies (政策) 頁面的左側選單中，選擇 Things (物件)。
7. 在憑證的 Things (物件) 頁面上，如果有任何物件連接至憑證，請開啟物件的選項選單並選擇 Detach (分開)，來分開每個物件。

   在您繼續之前，憑證不得有任何連接的物件。
8. 在憑證的 Things (物件) 頁面的左側選單中，選擇 Details (詳細資訊)。
9. 在憑證的 Details (詳細資訊) 頁面的 Actions (動作) 選單中，選擇 Start transfer (開始傳輸) 以開啟 Start transfer (開始傳輸) 對話方塊。
10. 在開始轉移對話方塊中，輸入要接收憑證的帳戶 AWS 帳戶 號碼，以及選用的簡短訊息。
11. 選擇 Start transfer (開始傳輸) 以傳輸憑證。

主控台應該會顯示訊息，指出傳輸成功或失敗。如果傳輸已開始，則憑證的狀態會更新為 Transferred (已傳輸)。

開始傳輸憑證 (CLI)

若要完成此程序，將需要您想要傳輸之憑證的 *certificateId* 和 *certificateArn*。

從具有要傳輸之憑證的帳戶執行此程序。

開始將憑證轉移到另一個 AWS 帳戶

1. 使用 update-certificate 命令來停用憑證

   ```
   aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
   ```
2. 分開所有策略。

   1. 使用 list-attached-policies 命令來列出已連接至憑證的政策。

      ```
      aws iot list-attached-policies --target certificateArn
      ```
   2. 對於每個連接的政策，使用 detach-policy 命令來分開政策。

      ```
      aws iot detach-policy --target certificateArn --policy-name policy-name
      ```

3. 分開所有物件。

   1. 使用 list-principal-things 命令來列出已連接至憑證的物件。

      ```
      aws iot list-principal-things --principal certificateArn
      ```

   2. 對於每個連接的物件，使用 detach-thing-principal 命令來分開物件。

      ```
      aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
      ```

4. 使用 transfer-certificate 命令來啟動憑證傳輸。

   ```
   aws iot transfer-certificate --certificate-id certificateId --target-aws-
   account account-id
   ```

接受或拒絕憑證傳輸

您可以使用 AWS IoT 主控台或 AWS 帳戶 ，接受或拒絕從另一個 傳輸給您 AWS 帳戶 的憑證 AWS CLI。

接受或拒絕憑證傳輸 (主控台)

若要完成此程序，將需要已傳輸至您帳戶之憑證的 ID。

從接收所傳輸憑證的帳戶執行此程序。

接受或拒絕已傳輸至 的憑證 AWS 帳戶

1. 登入 AWS 管理主控台並開啟AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。

   選擇您想要接受或拒絕且狀態為 Pending transfer (待定傳輸) 的憑證，並開啟其詳細資訊頁面。
3. 在憑證的 Details (詳細資訊) 頁面的 Actions (動作) 選單中，

   - 若要接受憑證，請選擇 Accept transfer (接受傳輸)。
   - 若不要接受憑證，請選擇 Reject transfer (拒絕傳輸)。

接受或拒絕憑證傳輸 (CLI)

若要完成此程序，將需要您想要接受或拒絕之憑證傳輸的 *certificateId*。

從接收所傳輸憑證的帳戶執行此程序。

接受或拒絕已傳輸至 的憑證 AWS 帳戶

1. 使用 accept-certificate-transfer 命令來接受憑證。

```
aws iot accept-certificate-transfer --certificate-id certificateId
```

2. 使用 reject-certificate-transfer 命令來拒絕憑證。

```
aws iot reject-certificate-transfer --certificate-id certificateId
```

取消憑證傳輸

您可以在接受憑證傳輸之前，使用 AWS IoT 主控台或 AWS CLI來取消憑證傳輸。

取消憑證傳輸 (主控台)

若要完成此程序，將需要您想要取消之憑證傳輸的 ID。

從已起始憑證傳輸的帳戶執行此程序。

取消憑證傳輸

1. 登入 AWS 管理主控台並開啟 AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Secure (安全)，然後選擇 Certificates (憑證)。

   選擇您想要取消其傳輸且狀態為 Transferred (已傳輸) 的憑證，並開啟其選項選單。
3. 在憑證的選項選單上，選擇 Revoke transfer (撤銷傳輸) 選項來取消憑證傳輸。

> **Important**
>
> 請小心不要弄錯 Revoke transfer (撤銷傳輸) 選項與 Revoke (撤銷) 選項。
> Revoke transfer (撤銷傳輸) 選項會取消憑證傳輸，而 Revoke (撤銷) 選項會使 AWS IoT無法復原且無法使用憑證。

取消憑證傳輸 (CLI)

若要完成此程序，將需要您想要取消之憑證傳輸的 *certificateId*。

從已起始憑證傳輸的帳戶執行此程序。

使用 cancel-certificate-transfer 命令來取消憑證傳輸。

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

自訂用戶端憑證驗證

AWS IoT Core 支援 X.509 用戶端憑證的自訂用戶端憑證驗證，可增強用戶端身分驗證管理。此憑證驗證方法也稱為驗證前憑證檢查，您會根據自己的條件（在 Lambda 函數中定義）評估用戶端憑證，並撤銷用戶端憑證或憑證的簽署憑證授權單位 (CA) 憑證，以防止用戶端連線至 AWS IoT Core。例如，您可以建立自己的憑證撤銷檢查，以針對支援線上憑證狀態通訊協定 (OCSP) 或憑證撤銷清單 (CRL) 端點的驗證授權單位驗證憑證的狀態，並防止具有撤銷憑證的用戶端連線。 https://en.wikipedia.org/wiki/Certificate_revocation_list用於評估用戶端憑證的條件是在 Lambda 函數（也稱為預先驗證 Lambda) 中定義。您必須使用網域組態中設定的端點，且身分驗證類型必須是 X.509 憑證。此外，用戶端在連線至 時必須提供伺服器名稱指示 (SNI) 延伸 AWS IoT Core。

> **Note**
>
> AWS GovCloud (US) 區域不支援此功能。

執行自訂用戶端憑證驗證的程序包含下列步驟。

- 步驟 1：向 註冊您的 X.509 用戶端憑證 AWS IoT Core
- 步驟 2：建立 Lambda 函數
- 步驟 3：授權 AWS IoT 叫用您的 Lambda 函數
- 步驟 4：設定網域的身分驗證組態

步驟 1：向 註冊您的 X.509 用戶端憑證 AWS IoT Core

如果您尚未這麼做，請註冊並啟用 X.509 用戶端憑證。 AWS IoT Core否則，跳至下一步。

若要使用 註冊和啟用您的用戶端憑證 AWS IoT Core，請遵循下列步驟：

1. 如果您直接使用 建立用戶端憑證 AWS IoT。這些用戶端憑證會自動向 註冊 AWS IoT Core。
2. 如果您建立自己的用戶端憑證，請依照這些指示向 註冊 AWS IoT Core憑證。
3. 若要啟用您的用戶端憑證，請遵循下列指示。

步驟 2：建立 Lambda 函數

您需要建立 Lambda 函數，以執行憑證驗證，並針對所設定端點的每次用戶端連線嘗試呼叫 。建立此 Lambda 函數時，請遵循建立第一個 Lambda 函數的一般指引。此外，請確保 Lambda 函數遵循預期的請求和回應格式，如下所示：

Lambda 函數事件範例

```
{
 "connectionMetadata": {
  "id": "string"
 },
 "principalId": "string",
 "serverName": "string",
 "clientCertificateChain": [
  "string",
  "string"
 ]
}
```

`connectionMetadata`

中繼資料或與用戶端連線相關的其他資訊 AWS IoT Core。

`principalId`

與 TLS 連線中的用戶端相關聯的主體識別符。

`serverName`

伺服器名稱指示 (SNI) 主機名稱字串。 AWS IoT Core 需要裝置將 SNI 延伸項目傳送至 Transport Layer Security (TLS) 通訊協定，並在 `host_name`欄位中提供完整的端點地址。

`clientCertificateChain`

代表用戶端 X.509 憑證鏈的字串陣列。

Lambda 函數回應範例

```
{
 "isAuthenticated": "boolean"
}
```

`isAuthenticated`

布林值，指出是否驗證請求。

> **Note**
>
> 在 Lambda 回應中， `isAuthenticated` 必須`true`繼續進行進一步的身分驗證和授權。否則，可以停用 IoT 用戶端憑證，也可以封鎖使用 X.509 用戶端憑證的自訂身分驗證，以進一步進行身分驗證和授權。

步驟 3：授權 AWS IoT 叫用您的 Lambda 函數

建立 Lambda 函數之後，您必須使用 add-permission CLI 命令，授予 AWS IoT 叫用它的許可。請注意，每次嘗試連線到您設定的端點時，都會叫用此 Lambda 函數。如需詳細資訊，請參閱授權 AWS IoT 叫用 Lambda 函數。

步驟 4：設定網域的身分驗證組態

下一節說明如何使用 設定自訂網域的身分驗證組態 AWS CLI。

設定網域的用戶端憑證組態 (CLI)

如果您沒有網域組態，請使用 create-domain-configuration CLI 命令來建立組態。如果您已有網域組態，請使用 update-domain-configuration CLI 命令來更新網域的用戶端憑證組態。您必須新增您在上一個步驟中建立的 Lambda 函數 ARN。

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config 'clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config '{"clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

`domain-configuration-name`

網域組態的名稱。

`authentication-type`

網域組態的身分驗證類型。如需詳細資訊，請參閱選擇身分驗證類型。

`application-protocol`

裝置用來與之通訊的應用程式通訊協定 AWS IoT Core。如需詳細資訊，請參閱選擇應用程式通訊協定。

`client-certificate-config`

指定網域用戶端身分驗證組態的物件。

`clientCertificateCallbackArn`

建立新連線時，在 TLS layer 中 AWS IoT 叫用之 Lambda 函數的 Amazon Resource Name (ARN)。若要自訂用戶端身分驗證以執行自訂用戶端憑證驗證，您必須新增您在上一個步驟中建立之 Lambda 函數的 ARN。

如需詳細資訊，請參閱 AWS IoT API 參考中的 CreateDomainConfiguration 和 UpdateDomainConfiguration。如需網域組態的詳細資訊，請參閱網域組態。

IAM 使用者、群組和角色

IAM 使用者、群組和角色為管理 AWS中身分及身分驗證的標準機制，您可以使用它們來使用 AWS SDK 和 連線到 AWS IoT HTTP 介面 AWS CLI。

IAM 角色也允許 代表您 AWS IoT 存取您帳戶中的其他 AWS 資源。例如，如果您想要讓裝置將其狀態發佈至 DynamoDB 資料表，IAM 角色 AWS IoT 允許 與 Amazon DynamoDB 互動。如需詳細資訊，請參閱 IAM 角色。

對於透過 HTTP 的訊息中介裝置連線， 會使用 Signature 第 4 版簽署程序來 AWS IoT 驗證使用者、群組和角色。如需詳細資訊，請參閱簽署 AWS API 請求。

使用查詢參數搭配 AWS Security Token Service (AWS STS) 提供的臨時安全登入資料來驗證請求時，計算簽章時請勿在正式查詢字串`X-Amz-Security-Token`中包含 。相反地，請在計算簽章後附加 `X-Amz-Security-Token` 做為查詢參數。這與要求安全字符成為正式請求一部分的一些 AWS 其他服務不同。如需詳細資訊，請參閱使用臨時安全登入資料簽署請求。

> **Note**
>
> AWS IoT 裝置 SDKs 會自動處理此簽署行為。如果您要實作自訂簽署程式碼，請參閱 SDK 來源以取得參考：
>
> - AWS 適用於 Python v2 的 IoT 裝置 SDK — 請參閱 `websockets_with_default_aws_signing()`，其中設定 `omit_session_token=True`
> - AWS 適用於 Java 的 IoT 裝置 SDK v2 — 請參閱 `setOmitSessionToken(true)`

搭配 AWS Signature 第 4 版使用 時 AWS IoT，用戶端必須在其 TLS 實作中支援下列項目：

- TLS 1.2
- SHA-256 RSA 憑證簽章驗證
- TLS 密碼套件支援部分的一種密碼套件

如需相關資訊，請參閱 的身分和存取管理 AWS IoT。

Amazon Cognito 身分

Amazon Cognito Identity 可讓您建立暫時性的有限權限 AWS 登入資料，以用於行動和 Web 應用程式。當您使用 Amazon Cognito Identity 時，請建立身分集區，為您的使用者建立唯一身分，並使用 Login with Amazon、Facebook 和 Google 等身分提供者進行身分驗證。您也可以將 Amazon Cognito 身分識別與自己的開發人員已驗證身分搭配使用。如需詳細資訊，請參閱 Amazon Cognito 身分。

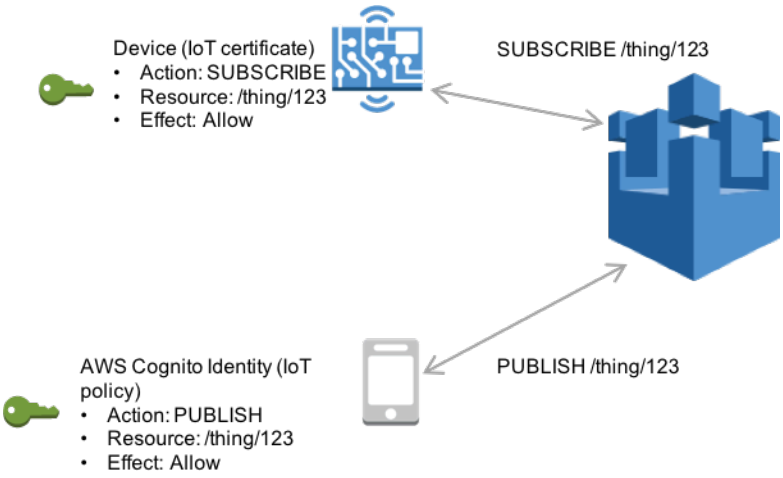
若要使用 Amazon Cognito Identity，請定義與 IAM 角色相關聯的 Amazon Cognito 身分集區。IAM 角色與 IAM 政策相關聯，該政策會授予身分集區中的身分存取 AWS 資源的許可，例如呼叫 AWS 服務。

Amazon Cognito 身分會建立未驗證和已驗證的身分。未驗證的身分適用於行動裝置或 Web 應用程式中想使用應用程式但不想登入的訪客使用者。未驗證的使用者只會獲得與身分集區相關聯之 IAM 政策中所指定的許可。

當您使用已驗證的身分時，除了連接到身分集區的 IAM 政策之外，您還必須將 AWS IoT 政策連接到 Amazon Cognito 身分。若要連接政策 AWS IoT ，請使用 AttachPolicy API，並將許可授予 AWS IoT 應用程式的個別使用者。您可以使用 AWS IoT 政策為特定客戶及其裝置指派精細的許可。

已驗證和未驗證的使用者是不同的身分類型。如果您未將 AWS IoT 政策連接至 Amazon Cognito 身分，則已驗證的使用者在 中的授權會失敗 AWS IoT ，而且無法存取 AWS IoT 資源和動作。如需為

Amazon Cognito 身分建立政策的詳細資訊，請參閱 發佈/訂閱政策範例 和 使用 Amazon Cognito 身分授權。



自訂身分驗證和授權

AWS IoT Core 可讓您定義自訂授權方，以便管理自己的用戶端身分驗證和授權。當您需要使用 AWS IoT Core 非原生支援的身分驗證機制時，這很有用。(如需原生支援機制的詳細資訊，請參閱 the section called “用戶端身分驗證”)。

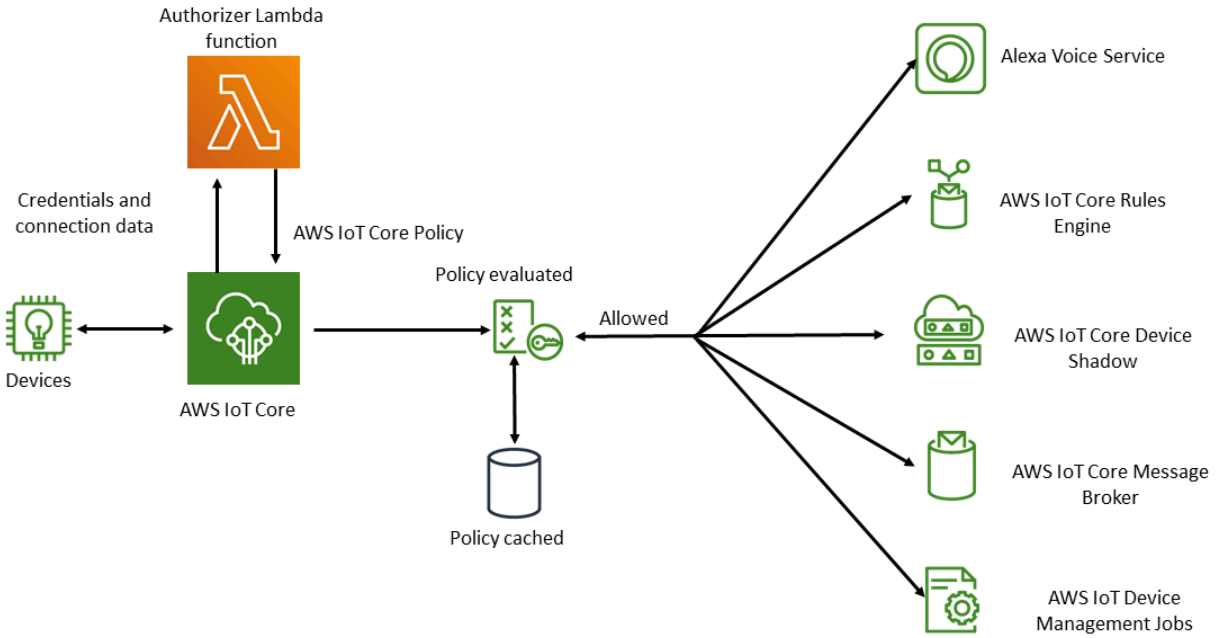
例如，如果您要將 欄位中的現有裝置遷移至 ， AWS IoT Core 而這些裝置使用自訂承載字符或 MQTT 使用者名稱和密碼進行身分驗證， 則可以將其遷移至 ， AWS IoT Core 而不必為其佈建新身分。您可以搭配 AWS IoT Core 支援的任何通訊協定使用自訂身分驗證。如需 AWS IoT Core 支援的通訊協定的詳細資訊，請參閱 the section called “裝置通訊協定”。

主題

- 了解自訂身分驗證工作流程
- 建立和管理自訂授權方 (CLI)
- 使用 X.509 用戶端憑證進行自訂身分驗證
- AWS IoT Core 使用自訂身分驗證連線至
- 疑難排解您的授權方

了解自訂身分驗證工作流程

自訂身分驗證可讓您定義如何使用授權方資源來驗證和授權用戶端。 每個授權方都包含客戶受管 Lambda 函數的參考、用於驗證裝置憑證的選用公有金鑰，以及其他組態資訊。 下圖說明 中自訂身分驗證的授權工作流程 AWS IoT Core。



AWS IoT Core 自訂身分驗證和授權工作流程

下方清單說明自訂身分驗證和授權工作流程中的每個步驟。

1. 裝置會使用其中一個支援的 來連線至客戶 AWS IoT Core 的資料端點the section called “裝置通訊協定”。裝置會在請求的標頭欄位或查詢參數（適用於 HTTP Publish 或 MQTT over WebSockets 通訊協定）中，或在 MQTT CONNECT 訊息的使用者名稱和密碼欄位（適用於 MQTT 和 MQTT over WebSockets 通訊協定）中傳遞登入資料。
2. AWS IoT Core 會檢查兩個條件之一：
   - 傳入的請求指定授權方。
   - 接收請求 AWS IoT Core 的資料端點已為其設定預設授權方。

   如果 以下列其中一種方式 AWS IoT Core 尋找授權方， AWS IoT Core 會觸發與授權方相關聯的 Lambda 函數。
3. （選用）如果您已啟用權杖簽署，請在觸發 Lambda 函數之前，使用存放在授權方中的 公有金鑰 AWS IoT Core 驗證請求簽章。如果驗證失敗， AWS IoT Core 會停止請求，而不會叫用 Lambda 函數。
4. Lambda 函數會接收請求中的憑證和連線中繼資料，並做出身分驗證決策。

5. Lambda 函數會傳回身分驗證決策的結果，以及指定連線中允許哪些動作 AWS IoT Core 的政策文件。Lambda 函數也會傳回資訊，指定透過叫用 Lambda 函數 AWS IoT Core 重新驗證請求中登入資料的頻率。
6. AWS IoT Core 根據從 Lambda 函數收到的政策，評估連線上的活動。
7. 建立連線且最初調用您的自訂授權方 Lambda 後，閒置連線的下一個調用最多可延遲 5 分鐘，而不需要任何 MQTT 操作。之後，後續調用將遵循自訂授權方 Lambda 中的重新整理間隔。這種方法可以防止超出 Lambda 並行限制的過度調用 AWS 帳戶。

擴展考量

因為 Lambda 函數會為您的授權方處理身分驗證和授權，所以此函數會受制於 Lambda 定價和服務配額，例如並行執行速率。 如需 Lambda 定價的詳細資訊，請參閱 Lambda 定價。您可以調整 Lambda 函數回應中的 `refreshAfterInSeconds` 和 `disconnectAfterInSeconds` 參數，管理 Lambda 函數上的負載。如需 Lambda 函數回應內容的詳細資訊，請參閱 the section called “定義您的 Lambda 函式”。

> **Note**
>
> 如果將簽署保持啟用狀態，您可以防止無法辨識的用戶端過度觸發 Lambda。在您的授權方中停用簽署之前，請考慮這一點。

> **Note**
>
> 自訂授權方的 Lambda 函數逾時限制為 5 秒。

建立和管理自訂授權方 (CLI)

AWS IoT Core 使用自訂授權方實作自訂身分驗證和授權機制。自訂授權方是一種 AWS IoT Core 資源，可讓您根據特定需求靈活定義和實作規則和政策。若要使用step-by-step指示建立自訂授權方，請參閱教學課程：為 建立自訂授權方 AWS IoT Core。

每個授權方都包含以下元件：

- 名稱：識別授權方的唯一使用者定義字串。
- Lambda 函數 ARN：Lambda 函數的 Amazon 資源名稱 (ARN)，用於實作授權和身分驗證邏輯。

- 字符金鑰名稱：用來從 HTTP 標頭、查詢參數或 MQTT CONNECT 使用者名稱擷取字符以執行簽章驗證的金鑰名稱。如果在您的授權方中啟用簽署，則此值為必要的。
- 已停用簽署旗標 (選用)：布林值，用來指定是否要停用憑證上的簽署需求。 這對於簽署憑證沒有意義的案例很有用，例如使用 MQTT 使用者名稱和密碼的身分驗證結構述。預設值為 `false`，因此預設會啟用簽署。
- 字符簽署公有金鑰： AWS IoT Core 用來驗證字符簽章的公有金鑰。其長度下限為 2,048 位元。如果在您的授權方中啟用簽署，則此值為必要的。

Lambda 會依據 Lambda 函數執行的次數，以及在您函數中執行程式碼所需的時間，向您收費。如需 Lambda 定價的詳細資訊，請參閱 Lambda 定價。如需建立 Lambda 函數的詳細資訊，請參閱《Lambda 開發人員指南》。

> **Note**
>
> 如果將簽署保持啟用狀態，您可以防止無法辨識的用戶端過度觸發 Lambda。在您的授權方中停用簽署之前，請考慮這一點。

> **Note**
>
> 自訂授權方的 Lambda 函數逾時限制為 5 秒。

在本章中：

- 定義您的 Lambda 函式
- 建立授權方
- 授權 AWS IoT 叫用您的 Lambda 函數
- 測試您的授權方
- 管理自訂授權方

定義您的 Lambda 函式

當 AWS IoT Core 叫用您的授權方時，它會觸發與授權方相關聯的 Lambda，其中包含下列 JSON 物件的事件。範例 JSON 物件包含所有可能的欄位。不包含與連線請求無關的任何欄位。

```
{
```

```
    "token" :"aToken",
    "signatureVerified": Boolean, // Indicates whether the device gateway has validated
 the signature.
    "protocols": ["tls", "http", "mqtt"], // Indicates which protocols to expect for
 the request.
    "protocolData": {
        "tls" : {
            "serverName": "serverName" // The server name indication (SNI) host_name
 string.
        },
        "http": {
            "headers": {
                "#{name}": "#{value}"
            },
            "queryString": "?#{name}=#{value}"
        },
        "mqtt": {
            "username": "myUserName",
            "password": "myPassword", // A base64-encoded string.
            "clientId": "myClientId" // Included in the event only when the device
 sends the value.
        }
    },
    "connectionMetadata": {
        "id": UUID // The connection ID. You can use this for logging.
    },
}
```

Lambda 函數應該使用此資訊來驗證傳入連線，並決定連線中允許哪些動作。此函數應該傳送包含下列值的回應。

- `isAuthenticated`：指出是否驗證請求的布林值。
- `principalId`：英數字串，作為自訂授權請求所傳送之字符的識別符。該值必須為英數字串 (字元數量不低於 1 個且不超過 128 個)，且符合此規則表達式 (regex) 模式：`([a-zA-Z0-9]){1,128}`。不允許非英數字元的特殊字元與 `principalId`搭配使用 AWS IoT Core。如果允許 使用非英數特殊字元，請參閱 AWS 其他服務的文件`principalId`。
- `policyDocuments`：JSON 格式 AWS IoT Core 政策文件的清單 如需建立 AWS IoT Core 政策的詳細資訊，請參閱the section called “AWS IoT Core 政策”。政策文件的數目上限為 10 份政策文件。每份政策文件最多可包含 2,048 個字元。
- `disconnectAfterInSeconds`：整數，用來指定連接至 AWS IoT Core 閘道的持續時間上限 (以秒為單位)。最小值為 300 秒，最大值為 86,400 秒。預設值為 86，400。

> **Note**
>
> 連線建立時，會設定 的值 `disconnectAfterInSeconds`（由 Lambda 函數傳回）。在後續政策重新整理 Lambda 調用期間，無法修改此值。

- `refreshAfterInSeconds`：整數，用來指定政策重新整理的間隔。一旦超出此間隔， AWS IoT Core 便會叫用 Lambda 函數，以允許政策重新整理。最小值為 300 秒，最大值為 86,400 秒。

下列 JSON 物件包含您的 Lambda 函數可以傳送的回應範例。

```
{ "isAuthenticated":true, //A Boolean that determines whether
client can connect. "principalId": "xxxxxxxx",  //A string that
identifies the connection in logs. "disconnectAfterInSeconds": 86400,
  "refreshAfterInSeconds": 300,   "policyDocuments": [       {
 "Version": "2012-10-17",         "Statement": [           {
    "Action": "iot:Publish",              "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:<your_aws_account_id>:topic/
customauthtesting"            }         ]       }     ] }
```

`policyDocument` 值必須包含有效的 AWS IoT Core 政策文件。如需 AWS IoT Core 政策的詳細資訊，請參閱 the section called “AWS IoT Core 政策”。 在 MQTT over TLS 和 MQTT over WebSockets 連線中， 會在 `refreshAfterInSeconds` 欄位值中指定的間隔 AWS IoT Core 內快取此政策。如果採用 HTTP 連線，除非裝置使用的是 HTTP 持續連線 (也稱為 HTTP 保持連線或 HTTP 連線重複使用)，否則每個授權請求都會呼叫 Lambda 函數。您可以在設定授權方時選擇啟用快取。在此間隔期間， 會針對此快取政策 AWS IoT Core 授權已建立連線中的動作，而不會再次觸發您的 Lambda 函數。如果在自訂身分驗證期間發生失敗， 會 AWS IoT Core 終止連線。如果連線開啟的時間超過 `disconnectAfterInSeconds`參數中指定的值， AWS IoT Core 也會終止連線。

下列 JavaScript 包含範例 Node.js Lambda 函數，其會在 MQTT Connect 訊息中尋找值為 的密碼，`test`並傳回政策，授予許可以 AWS IoT Core 連接至名為 的用戶端，`myClientName`並發佈至包含相同用戶端名稱的主題。如果找不到預期的密碼，它會傳回拒絕這兩個動作的政策。

```
// A simple Lambda function for an authorizer. It demonstrates
// how to parse an MQTT password and generate a response.

exports.handler = function(event, context, callback) {
    var uname = event.protocolData.mqtt.username;
    var pwd = event.protocolData.mqtt.password;
```

```
    var buff = new Buffer(pwd, 'base64');
    var passwd = buff.toString('ascii');
    switch (passwd) {
        case 'test':
            callback(null, generateAuthResponse(passwd, 'Allow'));
            break;
        default:
            callback(null, generateAuthResponse(passwd, 'Deny'));
    }
};

// Helper function to generate the authorization response.
var generateAuthResponse = function(token, effect) {
    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'TEST123';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var publishStatement = {};
    var connectStatement = {};
    connectStatement.Action = ["iot:Connect"];
    connectStatement.Effect = effect;
    connectStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:client/
myClientName"];
    publishStatement.Action = ["iot:Publish"];
    publishStatement.Effect = effect;
    publishStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:topic/telemetry/
myClientName"];
    policyDocument.Statement[0] = connectStatement;
    policyDocument.Statement[1] = publishStatement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 300;

    return authResponse;
}
```

當上述 Lambda 函數在 MQTT Connect 訊息中收到 `test` 的預期密碼時，它會傳回下列 JSON。`password` 和 `principalId` 屬性的值會是 MQTT 連結訊息的值。

```
{
```

```
  "password": "password",
  "isAuthenticated": true,
  "principalId": "principalId",
  "policyDocuments": [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "iot:Connect",
          "Effect": "Allow",
          "Resource": "*"
        },
        {
          "Action": "iot:Publish",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Subscribe",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topicfilter/telemetry/
${iot:ClientId}"
        },
        {
          "Action": "iot:Receive",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        }
      ]
    }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 300
}
```

建立授權方

您可以使用 CreateAuthorizer API 建立授權方。下列範例描述此命令。

```
aws iot create-authorizer
--authorizer-name MyAuthorizer
--authorizer-function-arn arn:aws:lambda:us-
west-2:<account_id>:function:MyAuthorizerFunction  //The ARN of the Lambda function.
[--token-key-name MyAuthorizerToken //The key used to extract the token from headers.
```

```
[--token-signing-public-keys FirstKey=
 "-----BEGIN PUBLIC KEY-----
  [...insert your public key here...]
  -----END PUBLIC KEY-----"
[--status ACTIVE]
[--tags <value>]
[--signing-disabled | --no-signing-disabled]
```

您可以使用 `signing-disabled` 參數，在每次叫用授權方時選擇退出簽章驗證。除非您必須停用簽署，否則強烈建議您不要這樣做。簽章驗證可保護您防範未知裝置過度叫用 Lambda 函數。在建立授權方之後，您無法更新授權方的 `signing-disabled` 狀態。若要變更此行為，您必須使用不同的 `signing-disabled` 參數值建立另一個自訂授權方。

如果您已停用簽署，`tokenKeyName` 和 `tokenSigningPublicKeys` 參數的值是選用值。如果啟用簽署，則它們是必要值。

建立 Lambda 函數和自訂授權方之後，您必須明確授予 AWS IoT Core 服務許可，才能代表您叫用函數。 您可以使用下列命令來執行此操作。

> **Note**
>
> 預設 IoT 端點可能不支援搭配 Lambda 函數使用自訂授權方。反之，您可以使用網域組態來定義新的端點，然後為自訂授權方指定該端點。

```
aws lambda add-permission --function-name <lambda_function_name>
--principal iot.amazonaws.com --source-arn <authorizer_arn>
--statement-id Id-123 --action "lambda:InvokeFunction"
```

授權 AWS IoT 叫用您的 Lambda 函數

在本節中，您將授予您剛建立的自訂授權方資源的許可，以執行 Lambda 函數。若要授與許可，您可以使用 add-permission CLI 命令。

使用 將許可授予 Lambda 函數 AWS CLI

1. 在插入您的值之後，輸入以下命令。請注意，`statement-id` 值必須是唯一的。*`Id-1234`* 將 取代為您擁有的確切值，否則您可能會發生錯誤`ResourceConflictException`。

   ```
   aws lambda add-permission  \
   ```

```
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. 如果命令成功，它會傳回許可陳述式，例如此範例。您可以繼續下一節來測試自訂授權方。

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
}
```

如果命令未成功，它會傳回錯誤，例如此範例。您必須先檢閱並更正錯誤，然後才能繼續進行。

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

測試您的授權方

您可以使用 TestInvokeAuthorizer API 來測試授權方的叫用和傳回值。此 API 可讓您指定通訊協定中繼資料，並在您的授權方中測試簽章驗證。

下列索引標籤顯示如何使用 AWS CLI 來測試您的授權方。

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER \
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER ^
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER `
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

`token-signature` 參數的值是簽署的字符。若要了解如何取得此值，請參閱 the section called “簽署字符”。

如果您的授權方取得使用者名稱和密碼，您可以使用 `--mqtt-context` 參數來傳遞此資訊。下列標籤顯示如何使用 `TestInvokeAuthorizer` API，將包含使用者名稱、密碼和用戶端名稱的 JSON 物件傳送給您的自訂授權方。

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  \
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  ^
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  `
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

密碼必須為 base64 編碼。下列範例顯示如何在類似 Unix 環境中編碼密碼。

```
echo -n PASSWORD | base64
```

管理自訂授權方

您可以使用下列 API 來管理授權方。

- ListAuthorizers：顯示您帳戶中的所有授權方。
- DescribeAuthorizer：顯示所指定授權方的屬性。這些值包括建立日期、上次修改日期和其他屬性。
- SetDefaultAuthorizer：指定 AWS IoT Core 資料端點的預設授權方。如果裝置未傳遞 AWS IoT Core 登入資料，且未指定授權方，則 AWS IoT Core 使用此授權方。如需使用 AWS IoT Core 登入資料的詳細資訊，請參閱 the section called “用戶端身分驗證”。
- UpdateAuthorizer：變更所指定授權方的狀態、字符金鑰名稱或公有金鑰。
- DeleteAuthorizer：刪除指定的授權方。

> **Note**
>
> 您無法更新授權方的簽署需求。這表示您無法在需要簽署的現有授權方中停用簽署。您也無法在不需要簽署的現有授權方中要求簽署。

使用 X.509 用戶端憑證進行自訂身分驗證

將裝置連線至 時 AWS IoT Core，您有多種身分驗證類型可用。您可以使用 X.509 用戶端憑證來驗證用戶端和裝置連線，或定義自訂授權方來管理您自己的用戶端身分驗證和授權邏輯。本主題說明如何搭配 X.509 用戶端憑證使用自訂身分驗證。

如果您已經使用 X.509 憑證對裝置進行身分驗證，並想要執行額外的驗證和自訂授權，則使用自訂身分驗證搭配 X.509 憑證可能會有所幫助。例如，如果您在 X.509 用戶端憑證中存放裝置的資料，例如其序號，則在 AWS IoT Core 驗證 X.509 用戶端憑證之後，您可以使用自訂授權方，根據憑證的 CommonName 欄位中存放的資訊來識別特定裝置。將裝置連線至 時，搭配 X.509 憑證使用自訂身分驗證可增強裝置安全管理， AWS IoT Core 並提供管理身分驗證和授權邏輯的更多彈性。 AWS IoT Core 支援搭配 X.509 憑證的自訂身分驗證，其使用 X.509 憑證和自訂授權方身分驗證類型，同時適用於 MQTT 通訊協定和 HTTPS 通訊協定。如需裝置端點支援的身分驗證類型和應用程式通訊協定 AWS IoT Core 的詳細資訊，請參閱裝置通訊協定。

> **Note**
>
> AWS GovCloud (US) 區域不支援使用 X.509 用戶端憑證的自訂身分驗證。

> **Important**
>
> 您必須使用使用網域組態建立的端點。此外，用戶端在連線至 時必須提供伺服器名稱指示 (SNI) 延伸 AWS IoT Core。

使用自訂身分驗證搭配 X.509 用戶端憑證對裝置進行身分驗證的程序包含下列步驟。

- 步驟 1：向 註冊您的 X.509 用戶端憑證 AWS IoT Core
- 步驟 2：建立 Lambda 函數
- 步驟 3：建立自訂授權方
- 步驟 4：在網域組態中設定身分驗證類型和應用程式通訊協定

步驟 1：向 註冊您的 X.509 用戶端憑證 AWS IoT Core

如果您尚未這麼做，請註冊並啟用 X.509 用戶端憑證。 AWS IoT Core否則，跳至下一步。

若要使用 註冊和啟用您的用戶端憑證 AWS IoT Core，請遵循下列步驟：

1. 如果您直接使用 建立用戶端憑證 AWS IoT。這些用戶端憑證會自動向 註冊 AWS IoT Core。
2. 如果您建立自己的用戶端憑證，請依照這些指示向 註冊 AWS IoT Core憑證。
3. 若要啟用您的用戶端憑證，請遵循這些指示。

步驟 2：建立 Lambda 函數

AWS IoT Core 使用自訂授權方來實作自訂身分驗證和授權機制。自訂授權方與 Lambda 函數相關聯，該函數會判斷裝置是否經過身分驗證，以及允許裝置執行哪些操作。當裝置連線到 時 AWS IoT Core， 會 AWS IoT Core 擷取授權方詳細資訊，包括授權方名稱和相關聯的 Lambda 函數，並叫用 Lambda 函數。Lambda 函數會收到事件，其中包含具有裝置 X.509 用戶端憑證資料的 JSON 物件。您的 Lambda 函數使用此事件 JSON 物件來評估身分驗證請求、決定要採取的動作，以及傳送回應。

Lambda 函數事件範例

下列範例 JSON 物件包含可以包含的所有可能欄位。實際 JSON 物件只會包含與特定連線請求相關的欄位。

```
{
 "token": "aToken",
 "signatureVerified": true,
```

```
 "protocols": [
  "tls",
  "mqtt"
 ],
 "protocolData": {
  "tls": {
   "serverName": "serverName",
   "x509CertificatePem": "x509CertificatePem",
   "principalId": "principalId"
  },
  "mqtt": {
   "clientId": "myClientId",
                    "username": "myUserName",
                    "password": "myPassword"
  }
 },
 "connectionMetadata": {
  "id": "UUID"
 }
}
```

`signatureVerified`

布林值，指出在叫用授權方的 Lambda 函數之前，是否驗證授權方中設定的字符簽章。如果授權方設定為停用權杖簽署，則此欄位將為 false。

`protocols`

包含請求預期通訊協定的陣列。

`protocolData`

包含連線中所用通訊協定資訊的物件。它提供通訊協定特定的詳細資訊，可用於身分驗證、授權等。

`tls` - 此物件會保留與 TLS (Transport Layer Security) 通訊協定相關的資訊。

- `serverName` - 伺服器名稱指示 (SNI) 主機名稱字串。 AWS IoT Core 要求裝置將 SNI 延伸模組傳送至 Transport Layer Security (TLS) 通訊協定，並在 `host_name`欄位中提供完整的端點地址。
- `x509CertificatePem` - PEM 格式的 X.509 憑證，用於 TLS 連線中的用戶端身分驗證。
- `principalId` - TLS 連線中與用戶端相關聯的主體識別符。

`mqtt` - 此物件會保留 MQTT 通訊協定的相關資訊。

- `clientId` - 字串只需要包含在裝置傳送此值的事件中。
- `username` - MQTT Connect 封包中提供的使用者名稱。
- `password` - MQTT Connect 封包中提供的密碼。

`connectionMetadata`

連線的中繼資料。

`id` - 用於記錄和故障診斷的連線 ID。

> **Note**
>
> 在此事件中，JSON 物件 `x509CertificatePem`和 `principalId`是請求中的兩個新欄位。的值`principalId`與 的值相同`certificateId`。如需詳細資訊，請參閱憑證。

Lambda 函數回應範例

Lambda 函數應使用來自事件 JSON 物件的資訊來驗證傳入連線，並決定連線中允許哪些動作。

下列 JSON 物件包含 Lambda 函數可以傳送的範例回應。

```
{
 "isAuthenticated": true,
 "principalId": "xxxxxxxx",
 "disconnectAfterInSeconds": 86400,
 "refreshAfterInSeconds": 300,
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Effect": "Allow",
     "Action": "iot:Publish",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/customauthtesting"
    }
   ]
  }
 ]
}
```

在此範例中，此函數應該傳送包含下列值的回應。

`isAuthenticated`

布林值，指出是否驗證請求。

`principalId`

英數字串，可做為自訂授權請求所傳送字符的識別符。值必須是至少包含一個英數字串，且不超過 128 個字元。它會識別日誌中的連線。的值`principalId`必須與事件 JSON 物件`principalId`中的 值相同 （即 X.509 憑證的 certificateId)。

`policyDocuments`

JSON 格式 AWS IoT Core 政策文件的清單。此值為選用，並支援物件政策變數和憑證政策變數。政策文件的數量上限為 10。每份政策文件最多可包含 2,048 個字元。如果您將多個政策連接到用戶端憑證和 Lambda 函數，則許可是所有政策的集合。如需建立 AWS IoT Core 政策的詳細資訊，請參閱 政策。

`disconnectAfterInSeconds`

整數，指定與 AWS IoT Core 閘道連線的最長持續時間 （以秒為單位）。最小值為 300 秒，最大值為 86，400 秒。 `disconnectAfterInSeconds`是連線的生命週期，不會在連續政策重新整理時重新整理。

`refreshAfterInSeconds`

整數，指定政策重新整理之間的間隔。當此間隔通過時， 會 AWS IoT Core 叫用 Lambda 函數以允許政策重新整理。最小值為 300 秒，最大值為 86,400 秒。

Lambda 函數範例

以下是 Node.js Lambda 函數的範例。函數會檢查用戶端的 X.509 憑證，並擷取相關資訊，例如序號、指紋和主旨名稱。如果擷取的資訊符合預期值，則會授予用戶端連線的存取權。此機制可確保只有具有有效憑證的授權用戶端才能建立連線。

```
const crypto = require('crypto');

exports.handler = async (event) => {

    // Extract the certificate PEM from the event
    const certPem = event.protocolData.tls.x509CertificatePem;

    // Parse the certificate using Node's crypto module
    const cert = new crypto.X509Certificate(certPem);
```

```
    var effect = "Deny";
    // Allow permissions only for a particular certificate serial, fingerprint, and
 subject
    if (cert.serialNumber === "7F8D2E4B9C1A5036DE8F7C4B2A91E5D80463BC9A1257" // This is
 a random serial
        && cert.fingerprint ===
 "F2:9A:C4:1D:B5:E7:08:3F:6B:D0:4E:92:A7:C1:5B:8D:16:0F:E3:7A" // This is a random
 fingerprint
        && cert.subject === "allow.example.com") {
      effect = "Allow";
    }

    return generateAuthResponse(event.protocolData.tls.principalId, effect);
};


// Helper function to generate the authorization response.
function generateAuthResponse(principalId, effect) {
    const authResponse = {
        isAuthenticated: true,
        principalId,
        disconnectAfterInSeconds: 3600,
        refreshAfterInSeconds: 300,
        policyDocuments: [
          {
            Version: "2012-10-17",
            Statement: [
              {
                Action: ["iot:Connect"],
                Effect: effect,
                Resource: [
                  "arn:aws:iot:us-east-1:123456789012:client/myClientName"
                ]
              },
              {
                Action: ["iot:Publish"],
                Effect: effect,
                Resource: [
                  "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
                ]
              },
              {
                Action: ["iot:Subscribe"],
                Effect: effect,
```

```
                Resource: [
                   "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
                ]
              },
              {
                Action: ["iot:Receive"],
                Effect: effect,
                Resource: [
                   "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
                ]
              }
            ]
          }
        ]
      };

  return authResponse;
}
```

上述 Lambda 函數在收到具有預期序列、指紋和主體的憑證時，會傳回下列 JSON。的值`x509CertificatePem`將是 TLS 交握中提供的用戶端憑證。如需詳細資訊，請參閱[定義 Lambda 函數](#)。

```
{
 "isAuthenticated": true,
 "principalId": "principalId in the event JSON object",
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Action": "iot:Connect",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:client/myClientName"
    },
    {
     "Action": "iot:Publish",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
    },
    {
     "Action": "iot:Subscribe",
```

```
      "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
    },
    {
      "Action": "iot:Receive",
      "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
    }
   ]
  }
 ],
 "disconnectAfterInSeconds": 3600,
 "refreshAfterInSeconds": 300
}
```

步驟 3：建立自訂授權方

定義 Lambda 函數之後，請建立自訂授權方來管理您自己的用戶端身分驗證和授權邏輯。您可以遵循步驟 3：建立客戶授權方資源及其授權中的詳細說明。如需詳細資訊，請參閱建立 授權方。

在建立自訂授權方的過程中，您必須授予 AWS IoT 許可，以在建立 Lambda 函數之後叫用該函數。如需詳細說明，請參閱授權 AWS IoT 叫用您的 Lambda 函數。

步驟 4：在網域組態中設定身分驗證類型和應用程式通訊協定

若要使用自訂身分驗證搭配 X.509 用戶端憑證來驗證裝置，您必須在網域組態中設定身分驗證類型和應用程式通訊協定，而且必須傳送 SNI 延伸。的值`authenticationType`必須是 `CUSTOM_AUTH_X509`，而 的值`applicationProtocol`可以是 `SECURE_MQTT`或 `HTTPS`。

在網域組態 (CLI) 中設定身分驗證類型和應用程式通訊協定

如果您沒有網域組態，請使用 create-domain-configuration命令來建立組態。的值`authenticationType`必須是 `CUSTOM_AUTH_X509`，而 的值`applicationProtocol`可以是 `SECURE_MQTT`或 `HTTPS`。

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
```

```
    }'
```

如果您已有網域組態，applicationProtocol請視需要使用 update-domain-configuration 命令更新authenticationType和 。請注意，您無法變更預設端點 () 上的身分驗證類型或通訊協定iot:Data-ATS。

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

domain-configuration-name

網域組態的名稱。

authentication-type

網域組態的身分驗證類型。如需詳細資訊，請參閱選擇身分驗證類型。

application-protocol

裝置用來與之通訊的應用程式通訊協定 AWS IoT Core。如需詳細資訊，請參閱選擇應用程式通訊協定。

--authorizer-config

在網域組態中指定授權方組態的物件。

defaultAuthorizerName

網域組態的授權方名稱。

如需詳細資訊，請參閱 AWS IoT API 參考中的 CreateDomainConfiguration 和 UpdateDomainConfiguration。如需網域組態的詳細資訊，請參閱網域組態。

AWS IoT Core 使用自訂身分驗證連線至

裝置可以使用 AWS IoT Core 自訂身分驗證搭配 AWS IoT Core 支援裝置傳訊的任何通訊協定來連線至 。如需受支援通訊協定的詳細資訊，請參閱 the section called “裝置通訊協定”。 傳遞至授權方 Lambda 函數的連線資料取決於您使用的通訊協定。如需建立授權方 Lambda 函數的詳細資訊，請參

閱 the section called “定義您的 Lambda 函式”。以下各節說明如何使用每個支援的通訊協定來連線，以便進行身分驗證。

HTTPS

AWS IoT Core 使用 HTTP Publish API 將資料傳送到 的裝置，可以透過請求標頭或查詢其 HTTP POST 請求中的參數來傳遞憑證。裝置可以指定要使用 `x-amz-customauthorizer-name` 標頭或查詢參數叫用的授權方。如果您已在授權方中啟用字符簽署，則必須在請求標頭或查詢參數中傳遞 *`token-key-name`* 和 `x-amz-customauthorizer-signature`。請注意，在瀏覽器中使用 JavaScript 時，*`token-signature`* 的值必須是 URL 編碼格式。

> **Note**
>
> HTTPS 通訊協定的客戶授權方僅支援發佈操作。如需有關 HTTP 通訊協定的詳細資訊，請參閱 the section called “裝置通訊協定”。

下列範例請求會顯示如何在請求標頭和查詢參數中傳遞這些參數。

```
//Passing credentials via headers
POST /topics/topic?qos=qos HTTP/1.1
Host: your-endpoint
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
x-amz-customauthorizer-name: authorizer-name

//Passing credentials via query parameters
POST /topics/topic?qos=qos&x-amz-customauthorizer-signature=token-signature&token-key-name=token-value HTTP/1.1
```

MQTT

AWS IoT Core 使用 MQTT 連線連線至 的裝置可以透過 MQTT 訊息的 `username`和 `password` 欄位傳遞登入資料。此 `username` 值也可以選擇包含一個查詢字串，將其他值 (包括字符、簽章和授權方名稱) 傳遞給您的授權方。如果您想要使用字符型身分驗證結構描述，而不是 `username` 和 `password` 值，則可使用此查詢字串。

> **Note**
>
> 密碼欄位中的資料由 base64 編碼 AWS IoT Core。您的 Lambda 函數必須將其解碼。

以下範例包含一個 username 字串，其中包含指定字符和簽章的額外參數。

```
username?x-amz-customauthorizer-name=authorizer-name&x-amz-customauthorizer-
signature=token-signature&token-key-name=token-value
```

若要叫用 授權方， AWS IoT Core 使用 MQTT 和自訂身分驗證連線至 的裝置必須在連接埠 443 上連線。他們還必須傳遞值為 的 Application Layer Protocol Negotiation (ALPN) TLS 延伸mqtt，以及其 AWS IoT Core 資料端點主機名稱的 Server Name Indication (SNI) 延伸。為了避免潛在的錯誤，x-amz-customauthorizer-signature 的值必須進行 URL 編碼。我們也強烈建議您對 x-amz-customauthorizer-name 和 token-key-name 值進行 URL 編碼。如需這些值的詳細資訊，請參閱 the section called “裝置通訊協定”。V2 AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端 可以設定這些延伸模組。

MQTT over WebSockets

AWS IoT Core 透過 WebSockets 使用 MQTT 連線至 的裝置可以透過下列兩種方式之一傳遞登入資料。

- 透過 HTTP UPGRADE 請求中的請求標頭或查詢參數來建立 WebSockets 連線。
- 透過 MQTT CONNECT 訊息中的 username 和 password 欄位。

如果透過 MQTT CONNECT 訊息傳遞憑證，則需要 ALPN 和 SNI TLS 延伸。如需這些延伸的詳細資訊，請參閱 the section called “MQTT”。下列範例顯示如何透過 HTTP Upgrade 請求傳遞憑證。

```
GET /mqtt HTTP/1.1
Host: your-endpoint
Upgrade: WebSocket
Connection: Upgrade
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
sec-WebSocket-Key: any random base64 value
sec-websocket-protocol: mqtt
sec-WebSocket-Version: websocket version
```

簽署字符

您必須利用您在 create-authorizer 呼叫中所使用之公有/私有金鑰對中的私有金鑰簽署字符。下列範例顯示如何使用類似 UNIX 的命令和 JavaScript 來建立字符簽章。它們會使用 SHA-256 雜湊演算法來編碼簽章。

Command line

```
echo -n TOKEN_VALUE | openssl dgst -sha256 -sign PEM encoded RSA private key |
 openssl base64
```

JavaScript

```
const crypto = require('crypto')

const key = "PEM encoded RSA private key"

const k = crypto.createPrivateKey(key)
let sign = crypto.createSign('SHA256')
sign.write(t)
sign.end()
const s = sign.sign(k, 'base64')
```

疑難排解您的授權方

本主題逐步解說可能會在自訂身分驗證工作流程中產生問題的常見問題，以及解決這些問題的步驟。若要最有效地疑難排解問題，請啟用 的 CloudWatch 日誌， AWS IoT Core 並將日誌層級設定為 DEBUG。您可以在 AWS IoT Core 主控台中啟用 CloudWatch 日誌 (https://console.aws.amazon.com/iot/：//)。如需針對 AWS IoT Core啟用和設定記錄的詳細資訊，請參閱 the section called “設定 AWS IoT 記錄”。

> **Note**
>
> 如果您長期將日誌層級保留在 DEBUG，CloudWatch 可能會存放大量的記錄資料。這可能會增加您的 CloudWatch 費用。請考慮使用資源型記錄，以增加特定物件群組中僅限裝置的詳細資訊。如需資源型記錄的詳細資訊，請參閱 the section called “設定 AWS IoT 記錄”。此外，當您完成疑難排解時，請將日誌層級降低到較不詳細的層級。

開始疑難排解之前，請檢閱 the section called “了解自訂身分驗證工作流程”，以取得自訂身分驗證程序的高階檢視。這協助您了解可在哪裡尋找問題的來源。

本主題討論下列兩個可供您調查的領域。

- 與您授權方 Lambda 函數相關的問題。
- 與您裝置相關的問題。

檢查您的授權方 Lambda 函數中的問題

執行下列步驟，以確定您裝置的連線嘗試正在叫用 Lambda 函數。

1. 確認哪個 Lambda 函數與您的授權方相關聯。

   若要執行此確認，您可以呼叫 DescribeAuthorizer API，或在 AWS IoT Core 主控台的 Secure (安全) 區段中按一下所需的授權方。

2. 檢查 Lambda 函數的叫用指標。執行下列步驟來執行此動作。

   a. 開啟 AWS Lambda 主控台 (https://console.aws.amazon.com/lambda/：//)，然後選取與您的授權方相關聯的 函數。

   b. 選擇 Monitor (監控) 標籤，並檢視與您問題相關之時間範圍的指標。

3. 如果您未看到叫用，請確認 AWS IoT Core 具有叫用 Lambda 函數的許可。如果您看到叫用，請跳到下一個步驟。執行下列步驟來驗證您的 Lambda 函數是否具有必要的許可。

   a. 在 AWS Lambda 主控台中為您的函數選擇許可索引標籤。

   b. 在頁面底部尋找 Resource-based Policy (資源型政策) 區段。如果您的 Lambda 函數具有必要的許可，政策看起來像下列範例。

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Id123",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:111111111111:function:FunctionName",
      "Condition": {
        "ArnLike": {
```

```
          "AWS:SourceArn": "arn:aws:iot:us-east-1:111111111111:authorizer/
AuthorizerName"
        },
        "StringEquals": {
          "AWS:SourceAccount": "111111111111"
        }
      }
    }
  ]
}
```

c. 此政策會將函數的InvokeFunction許可授予 AWS IoT Core 委託人。如果您沒有看到它，您必須使用 AddPermission API 來新增它。下列範例顯示如何使用 AWS CLI來做到這一點。

```
aws lambda add-permission --function-name FunctionName --principal
 iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
 "lambda:InvokeFunction"
```

4. 如果您看到叫用，請確認沒有錯誤。錯誤可能表示 Lambda 函數未正確處理 AWS IoT Core 傳送給它的連線事件。

   如需在 Lambda 函數中處理事件的相關資訊，請參閱 the section called “定義您的 Lambda 函式”。您可以使用 AWS Lambda 主控台中的測試功能 (https://console.aws.amazon.com/lambda/：//) 在函數中硬式編碼測試值，以確保函數正確處理事件。

5. 如果您看到叫用沒有錯誤，但您的裝置無法連接 (或發佈、訂閱和接收訊息)，問題可能是您 Lambda 函數傳回的政策未對您裝置正在嘗試採取的動作提供許可。執行下列步驟，以判斷函數傳回的政策是否發生任何錯誤。

   a. 使用 Amazon CloudWatch Logs Insights 查詢，來掃描短時間內的記錄以檢查是否發生失敗。下列範例查詢會依時間戳記排序事件並尋找失敗。

   ```
   display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
    status = "Failure"
   ```

   b. 更新您的 Lambda 函數以記錄其傳回的資料， AWS IoT Core 以及觸發函數的事件。您可以使用這些記錄來檢查函數建立的政策。

6. 如果您看到叫用沒有錯誤，但您的裝置無法連接 (或發佈、訂閱和接收訊息)，另一個原因可能是您 Lambda 函數超過逾時限制。自訂授權方的 Lambda 函數逾時限制為 5 秒。您可以在 CloudWatch 日誌或指標中查看函數持續時間。

調查裝置問題

如果您發現叫用 Lambda 函數沒有問題，函數傳回的政策也沒有問題，請尋找裝置嘗試連線方面的問題。格式不正確的連線請求可能會導致 AWS IoT Core 不觸發您的授權方。在 TLS 和應用程式層可能會同時發生連線問題。

可能的 TLS 層問題：

- 客戶必須在所有自訂身分驗證請求中傳遞主機名稱標頭 (HTTP、MQTT over WebSockets) 或伺服器名稱指示 TLS 延伸 (HTTP、MQTT over WebSockets、MQTT)。在這兩種情況下，傳遞的值必須符合您帳戶的其中一個 AWS IoT Core 資料端點。這些是當您執行下列 CLI 命令時傳回的端點。
  - `aws iot describe-endpoint --endpoint-type iot:Data-ATS`
  - `aws iot describe-endpoint --endpoint-type iot:Data` (適用於舊式 VeriSign 端點)
- 使用自訂身分驗證進行 MQTT 連線的裝置也須傳遞應用程式層通訊協定交涉 (ALPN) TLS 延伸 (其值為 `mqtt`)。
- 自訂身分驗證目前僅適用於連接埠 443。

可能的應用程式層問題：

- 如果已啟用簽署 (在您的授權方中，`signingDisabled` 欄位是 false)，請尋找下列簽章問題。
  - 確定您是在 `x-amz-customauthorizer-signature` 標頭或在查詢字串參數中傳遞字符簽章。
  - 確定服務不是簽署字符以外的值。
  - 確定您在標頭或查詢參數中傳遞字符，而此標頭或查詢參數是您在授權方的 `token-key-name` 欄位中所指定的。
- 確定您在 `x-amz-customauthorizer-name` 標頭或查詢字串參數中傳遞的授權方名稱是有效的，或者您已為您的帳戶定義預設授權方。

Authorization

授權是授與許可給已驗證身分的程序。您可以在 AWS IoT Core 中使用 AWS IoT Core 和 IAM 政策授予許可。本主題涵蓋 AWS IoT Core 政策。如需建立 IAM 政策的詳細資訊，請參閱 的身分和存取管理 AWS IoT 和 AWS IoT 如何使用 IAM。

AWS IoT Core 政策決定已驗證身分可以執行的操作。裝置、行動應用程式、Web 應用程式和桌面應用程式，都會使用未驗證的身分，驗證身分甚至可以是輸入 CLI AWS IoT Core 命令的使用者。只有當身分具有授予這些 AWS IoT Core 操作許可的政策時，才能執行操作。

AWS IoT Core 政策和 IAM 政策都與 搭配使用， AWS IoT Core 以控制身分 （ 也稱為委託人) 可執行的操作。您使用的政策類型取決於您用來進行身分驗證的身分類型 AWS IoT Core。

AWS IoT Core 操作分為兩個群組：

- 控制平面 API 可讓您執行管理任務，例如建立或更新憑證、物件、規則等。
- 資料平面 API 可讓您將資料傳送至 並從中接收資料 AWS IoT Core。

您使用的政策類型，取決於您正使用控制平面或資料平面 API。

下表說明身分類型、其使用的通訊協定以及可用於授權的政策類型。

AWS IoT Core 資料平面 API 和政策類型

通訊協定和身分驗證機制	SDK	身分類型	Policy type (政策類型)		
MQTT over TLS/TCP、TLS 相互授權 (連接埠 8883 或 443)[‡])	AWS IoT 裝置 SDK	X.509 憑證	AWS IoT Core 政策		
透過 HTTPS/WebSocket 的 MQTT，AWS SigV4 身分驗證 （ 連接埠 443)	AWS 行動 SDK	已驗證的 Amazon Cognito 身分	IAM 和 AWS IoT Core 政策		
		未驗證的 Amazon Cognito 身分	IAM 政策		
		IAM 或聯合身分	IAM 政策		
HTTPS、AWS 簽章第 4 版身分驗證 （ 連接埠 443)	AWS CLI	Amazon Cognito、IAM 或聯合身分	IAM 政策		

通訊協定和身分驗證機制	SDK	身分類型	Policy type (政策類型)		
HTTPS、TLS 相互授權 (通訊埠 8443)	無 SDK 支援項目	X.509 憑證	AWS IoT Core 政策		
透過自訂身分驗證的 HTTPS(連接埠 443)	AWS IoT 裝置 SDK	自訂授權方	自訂授權方政策		

AWS IoT Core 控制平面 API 和政策類型

通訊協定和身分驗證機制	SDK	身分類型	Policy type (政策類型)		
HTTPS AWS Signature 第 4 版身分驗證 （連接埠 443)	AWS CLI	Amazon Cognito 身分	IAM 政策		
		IAM 或聯合身分	IAM 政策		

AWS IoT Core 政策會連接到 X.509 憑證、Amazon Cognito 身分或物件群組。IAM 政策會連接至 IAM 使用者、群組或角色。如果您使用 AWS IoT 主控台或 AWS IoT Core CLI 連接政策 （連接至憑證、Amazon Cognito Identity 或物件群組），您可以使用 AWS IoT Core 政策。否則，您使用連接到物件群組的 IAM policy. AWS IoT Core policies 適用於該物件群組中的任何物件。若要讓 AWS IoT Core 政策生效， `clientId`和 物件名稱必須相符。

政策型授權是一項強大工具。能夠讓您完全控制裝置、使用者或應用程式可在 AWS IoT Core執行的動作。例如，請考慮 AWS IoT Core 使用憑證連線至 的裝置。您可以允許該裝置存取所有 MQTT 主題，或者限制其存取單一主題。以另一個範例而言，假設一名使用者於命令列輸入 CLI 命令。透過使用政策，您可以允許或拒絕使用者存取任何命令或 AWS IoT Core 資源。您也可以控制應用程式存取 AWS IoT Core 資源。

由於 AWS IoT 快取政策文件的方式，對政策所做的變更可能需要幾分鐘的時間才能生效。亦即，存取最近已授與存取權的資源可能需要幾分鐘的時間，而且在撤銷資源的存取權之後，可能仍有數分鐘的時間可存取該資源。

AWS 訓練和認證

如需有關 中授權的資訊 AWS IoT Core，請前往 AWS Training and Certification 網站上的 Deep Dive to AWS IoT Core Authentication and Authorization 課程。

AWS IoT Core 政策

AWS IoT Core 政策是 JSON 文件。它們遵循與 IAM 政策相同的慣例。 AWS IoT Core 支援具名政策，因此許多身分可以參考相同的政策文件。具名政策會經過版本控制，因此可輕鬆還原。

AWS IoT Core 政策可讓您控制對 AWS IoT Core 資料平面的存取。 AWS IoT Core 資料平面包含您可進行的操作，像是連線至 AWS IoT Core 訊息代理程式、傳送及接收 MQTT 訊息，以及取得或更新物件的 Device Shadow。

AWS IoT Core 政策是包含一或多個政策陳述式的 JSON 文件。每個陳述式都包含：

- `Effect`，指定是否允許或拒絕該動作。
- `Action`，指定政策允許或拒絕的動作。
- `Resource`，指定資源或動作可用或不可用的資源。

由於 AWS IoT 快取政策文件的方式，對政策所做的變更可能需要 6 到 8 分鐘才會生效。亦即，存取最近已授與存取權的資源可能需要幾分鐘的時間，而且在撤銷資源的存取權之後，可能仍有數分鐘的時間可存取該資源。

AWS IoT Core 政策可以連接到 X.509 憑證、Amazon Cognito 身分和物件群組。附加至物件群組的政策會套用至該群組內的任何物件。若要使政策生效，`clientId` 與物件名稱必須相符。 AWS IoT Core 政策遵循與 IAM 政策相同的政策規避邏輯。根據預設，所有的政策都會以隱含方式拒絕。任何身分型或資源型政策中的明確允許會覆寫預設行為。任何政策中的明確拒絕會覆寫任何允許。如需詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的政策評估邏輯。

主題

- AWS IoT Core 政策動作
- AWS IoT Core 動作資源
- AWS IoT Core 政策變數

- 預防跨服務混淆代理人
- AWS IoT Core 政策範例
- 使用 Amazon Cognito 身分授權

AWS IoT Core 政策動作

以下為 AWS IoT Core定義的政策動作：

MQTT 政策動作

`iot:Connect`

代表連線至 AWS IoT Core 訊息中介裝置的許可。代理程式每次接收 `iot:Connect` 要求，就會檢查 `CONNECT` 許可。訊息代理程式不允許兩個相同用戶端 ID 的用戶端同時保持連線。第二個用戶端連線之後，代理程式會關閉現有的連線。使用 `iot:Connect` 許可，以確保僅使用特定用戶端 ID 的已授權用戶端可連線。

`iot:DeleteConnection`

代表中斷連線 MQTT 用戶端與 的連線許可 AWS IoT Core。每次發出強制中斷連線用戶端的請求時，都會檢查 `iot:DeleteConnection`許可。當您中斷連接用戶端時， 會 AWS IoT Core 關閉用戶端的網路連線，並選擇性地清除工作階段狀態。

`iot:GetRetainedMessage`

代表單一保留訊息內容的取得許可。保留訊息是已設定 RETAIN 旗標並存放的訊息 AWS IoT Core。如需取得所有帳戶保留訊息清單的許可，請參閱 `iot:ListRetainedMessages`。

`iot:ListRetainedMessages`

代表與帳戶保留訊息相關之摘要資訊的擷取許可，但不包括訊息內容。保留訊息是已設定 RETAIN 旗標並存放的訊息 AWS IoT Core。針對此動作指定的資源 ARN 必須是 *。如需取得單一保留訊息內容的許可，請參閱 `iot:GetRetainedMessage`。

`iot:Publish`

代表發佈 MQTT 主題的許可。代理程式每次接收 PUBLISH 請求，就會檢查此許可。您可以使用此許可來允許用戶端發佈至特定主題模式。

> **Note**
>
> 您必須也授予 `iot:Connect` 的許可，方能授予 `iot:Publish` 的許可。

`iot:Receive`

代表從 接收訊息的許可 AWS IoT Core。每次在將訊息傳送至用戶端時，系統就會確認 `iot:Receive` 許可。由於每次交付都會檢查此許可，因此您可以用其來撤銷目前訂閱某主題的用戶端許可。

`iot:RetainPublish`

代表發佈已設定 RETAIN 旗標之 MQTT 訊息的許可。

> **Note**
>
> 您必須也授予 `iot:Publish` 的許可，方能授予 `iot:RetainPublish` 的許可。

`iot:Subscribe`

代表主題篩選條件的訂閱許可。代理程式每次接收 SUBSCRIBE 請求，就會檢查此許可。可以使用此許可來允許用戶端訂閱與特定主題模式相符的主題。

> **Note**
>
> 您必須也授予 `iot:Connect` 的許可，方能授予 `iot:Subscribe` 的許可。

Device Shadow 政策動作

`iot:DeleteThingShadow`

代表可刪除物件 Device Shadow 的許可。每次發出刪除物件 Device Shadow 內容的請求時，就會檢查一次 `iot:DeleteThingShadow` 許可。

`iot:GetThingShadow`

代表可擷取物件 Device Shadow 的許可。每次發出擷取物件 Device Shadow 內容的請求時，就會檢查一次 `iot:GetThingShadow` 許可。

`iot:ListNamedShadowsForThing`

代表可列出物件具名影子的許可。每次發出列出物件具名影子的請求時，就會檢查一次 `iot:ListNamedShadowsForThing` 許可。

`iot:UpdateThingShadow`

代表可更新裝置影子的許可。每次發出更新物件 Device Shadow 內容的請求時，就會檢查一次 `iot:UpdateThingShadow` 許可。

> **Note**
>
> 任務執行政策動作僅適用於 HTTP TLS 端點。如果您使用 MQTT 端點，必須使用此主題上述所定義的 MQTT 政策動作。
> 如需示範此情況的任務執行政策範例，請參閱可與 MQTT 通訊協定搭配使用的 the section called “基本任務政策範例”。

任務執行 AWS IoT Core 政策動作

`iotjobsdata:DescribeJobExecution`

代表可擷取指定物件之任務執行的許可。每次發出取得任務執行的要求時，就會檢查一次 `iotjobsdata:DescribeJobExecution` 許可。

`iotjobsdata:GetPendingJobExecutions`

代表可擷取任務 (對物件來說並非結束狀態) 之清單的許可。每次發出擷取清單的要求時，就會檢查一次 `iotjobsdata:GetPendingJobExecutions` 許可。

`iotjobsdata:UpdateJobExecution`

代表可更新任務執行的許可。每次發出更新任務執行狀態的要求時，就會檢查一次 `iotjobsdata:UpdateJobExecution` 許可。

`iotjobsdata:StartNextPendingJobExecution`

代表可取得並啟動物件之下一個待定任務執行的許可。(也就是將狀態為 QUEUED 的任務執行，更新為 IN_PROGRESS。) 每次發出啟動下一個待定任務執行的要求時，就會檢查一次 `iotjobsdata:StartNextPendingJobExecution` 許可。

AWS IoT Core 登入資料提供者政策動作

`iot:AssumeRoleWithCertificate`

代表呼叫 AWS IoT Core 憑證提供者以使用憑證型身分驗證擔任 IAM 角色的許可。每次向 AWS IoT Core 登入資料提供者提出擔任角色的請求時，都會檢查 `iot:AssumeRoleWithCertificate`許可。

AWS IoT Core 動作資源

若要指定 AWS IoT Core 政策動作的資源，請使用資源的 Amazon Resource Name (ARN)。所有資源 ARNs都遵循下列格式：

```
arn:partition:iot:region:AWS-account-ID:Resource-type/Resource-name
```

下表顯示要為每個動作類型指定的資源。ARN 範例適用於分區 123456789012中的帳戶 ID aws，以及區域 的特定 us-east-1。如需 ARNs，請參閱 AWS Identity and Access Management 《使用者指南》中的 Amazon Resource Name (ARNs)。

Action	Resour Type (資源類型)	資源名稱	ARN 範例
iot:Connect	client	用戶端的用戶端 ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteConnection	client	用戶端的用戶端 ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteThingShadow	thing	物件的名稱和影子的名稱，如果適用的話	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne

Action	Resour Type (資源類型)	資源名稱	ARN 範例
iotjobsdata:DescribeJobExecution	thing	物件的名稱	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iotjobsdata:GetPendingJobExecutions	thing	物件的名稱	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:GetRetainedMessage	topic	保留訊息主題	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:GetThingShadow	thing	物件的名稱和影子的名稱，如果適用的話	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:ListNamedShadowsForThing	全部	全部	*
iot:ListRetainedMessages	全部	全部	*
iot:Publish	topic	主題字串	arn:aws:iot:us-east-1:123456789012:topic/myTopicName

Action	Resour Type (資源類型)	資源名稱	ARN 範例
iot:Receive	topic	主題字串	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:RetainPublish	topic	發佈已設定 RETAIN 旗標的主題	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iotjobsdata:StartNextPendingJobExecution	thing	物件的名稱	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:Subscribe	topicfilter	主題篩選條件字串	arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter
iotjobsdata:UpdateJobExecution	thing	物件的名稱	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:UpdateThingShadow	thing	物件的名稱和影子的名稱，如果適用的話	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne

Action	Resour Type (資源類型)	資源名稱	ARN 範例
iot:AssumeRoleWithCertificate	rolea]	指向角色 ARN 的角色別名	arn:aws:iot:us-east-1:123456789012:rolealias/CredentialProviderRole_alias

AWS IoT Core 政策變數

AWS IoT Core 定義可在 Resource或 Condition區塊中的政策中使用的 AWS IoT Core 政策變數。當政策受到評估時，實際值就會替代政策變數。例如，如果裝置連接到用戶端 ID 為 100-234-3456 AWS IoT Core 的訊息中介裝置，iot:ClientId政策變數會以 100-234-3456 取代政策文件中的政策變數。

AWS IoT Core 政策可以使用萬用字元，並遵循與 IAM 政策類似的慣例。在字串中插入 * (星號) 可以視為萬用字元，比對任何字元。例如，您可以使用 * 在政策的 Resource 屬性描述多個 MQTT 主題名稱。字元 + 和 # 在政策中視為文字字串。如需示範如何使用萬用字元的政策範例，請參閱 在 MQTT 和 AWS IoT Core 政策中使用萬用字元。

您也可以使用具有固定值的預先定義政策變數，表示具有特殊含意的字元。這些特殊字元包括 ${*}、${?} 和 ${$}。如需更多關於政策變數與特殊字元的資訊，請參閱 IAM 政策元素：變數與標籤和建立具有多個索引鍵或值的條件。

主題

- 基本 AWS IoT Core 政策變數
- 物件政策變數
- X.509 憑證 AWS IoT Core 政策變數

基本 AWS IoT Core 政策變數

AWS IoT Core 定義下列基本政策變數：

- aws:SourceIp：連接至 AWS IoT Core 訊息中介裝置的用戶端 IP 地址。
- iot:ClientId：用於連接至 AWS IoT Core 訊息代理程式的用戶端 ID。

- `iot:DomainName`：用戶端連線的網域名稱 AWS IoT Core。

範例

- ClientId 和 SourceIp政策變數的範例
- iot:DomainName 政策變數的範例

`ClientId` 和 `SourceIp`政策變數的範例

下列 AWS IoT Core 政策顯示使用政策變數的政策。 `aws:SourceIp`可用於政策的條件元素，以允許委託人僅在特定地址範圍內提出 API 請求。如需範例，請參閱 授權使用者和雲端服務使用 AWS IoT 任務。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
   ],
   "Condition": {
    "IpAddress": {
     "aws:SourceIp": "123.45.167.89"
    }
   }
  }
 ]
```

```
}
```

在這些範例中，${iot:ClientId} 會在評估政策時，以連線至 AWS IoT Core 訊息中介裝置的用戶端 ID 取代。在使用諸如 ${iot:ClientId} 的政策變數時，您可能會意外開放部分主題的存取權限。例如，若您的政策使用 ${iot:ClientId} 來指定主題篩選條件：

```
{
 "Effect": "Allow",
 "Action": [
  "iot:Subscribe"
 ],
 "Resource": [
  "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
 ]
}
```

用戶端可使用 + 作為用戶端 ID 來連線，使用者即可訂閱符合主題篩選條件 my/+/topic 的任何主題。若要避免這類安全漏洞，請使用 iot:Connect 政策動作來控制可連接的用戶端 ID。例如，此政策僅允許那些用戶端 ID 為 clientid1 的用戶端連接：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid"
   ]
  }
 ]
}
```

> **Note**
>
> 不建議搭配 Connect 使用政策變數 ${iot:ClientId}。沒有對 ClientId 的值進行檢查，因此具有不同用戶端 ID 的附加工具可以通過驗證，但會導致連線中斷。因為允許任何 ClientId，所以設定隨機用戶端 ID 可以略過物件群組政策。

iot:DomainName 政策變數的範例

您可以新增iot:DomainName政策變數，以限制允許使用哪些網域。新增 iot:DomainName政策變數可讓裝置僅連線到特定設定的端點。

下列政策允許裝置連線到指定的網域。

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "AllowConnectionsToSpecifiedDomain",
  "Effect": "Allow",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
  "Condition": {
   "StringEquals": {
    "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
   }
  }
 }
}
```

下列政策拒絕裝置連線到指定的網域。

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "DenyConnectionsToSpecifiedDomain",
  "Effect": "Deny",
  "Action": [
```

```
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
   "Condition": {
    "StringEquals": {
     "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
    }
   }
  }
}
```

如需政策條件式運算子的詳細資訊，請參閱 IAM JSON 政策元素：條件式運算子。如需網域組態的詳細資訊，請參閱什麼是網域組態？。

物件政策變數

物件政策變數可讓您撰寫 AWS IoT Core 政策，根據物件名稱、物件類型和物件屬性值等物件屬性來授予或拒絕許可。您可以使用物件政策變數來套用相同的政策來控制許多 AWS IoT Core 裝置。如需裝置佈建的詳細資訊，請參閱裝置佈建。

如果您使用非專屬物件關聯，則可以將相同的憑證連接到多個物件。若要維持明確的關聯並避免潛在的衝突，您必須將用戶端 ID 與物件名稱相符。在此情況下，您會在物件連線時傳送的 MQTT `Connect` 訊息中，從用戶端 ID 取得物件名稱 AWS IoT Core。

使用 AWS IoT Core 政策中的物件政策變數時，請注意下列事項。

- 使用 AttachThingPrincipal API，將憑證或委託人 (已驗證的 Amazon Cognito 身分) 連接至物件。
- 如果具有非專屬物件關聯，當您將物件名稱取代為物件政策變數時，MQTT 連線訊息或 TLS 連線`clientId`中的 值必須與物件名稱完全相符。

現已提供下列物件政策變數：

- `iot:Connection.Thing.ThingName`

  這會解析為正在評估政策之 AWS IoT Core 登錄檔中的物件名稱。 AWS IoT Core 會使用裝置在驗證時提供的憑證來判斷要使用哪些物件來驗證連線。本政策變數僅在裝置透過 MQTT 或 MQTT over WebSocket 通訊協定進行連線時可用。
- `iot:Connection.Thing.ThingTypeName`

當政策受到評估時，這可以解析為所屬物件相關的物件類型。MQTT/WebSocket 連線的用戶端 ID 必須與物件名稱相同。本政策變數僅在透過 MQTT 或是經 WebSocket 通訊協定的 MQTT 進行連線時可用。

- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  當政策受到評估時，這可以解析為所屬物件相關的特定屬性值。單一物件至多可具備 50 個屬性。每一屬性均可作為政策變數：`iot:Connection.Thing.Attributes[`*`attributeName`*`]`，其中 *`attributeName`* 為屬性名稱。MQTT/WebSocket 連線的用戶端 ID 必須與物件名稱相同。本政策變數僅在透過 MQTT 或是經 WebSocket 通訊協定的 MQTT 進行連線時可用。

- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]` 強制執行只有在 中註冊 AWS IoT 並連接到主體的裝置才能存取政策內的許可。如果裝置提供的憑證未連接到登錄檔中的 AWS IoT Core IoT 物件，您可以使用此變數來防止 AWS IoT Core 裝置連接到 。此變數具有 值，`true`或`false`指出連線物件使用 AttachThingPrincipal API 連接到登錄檔中的憑證或 Amazon Cognito 身分。物件名稱作為用戶端 ID。

如果您的用戶端 ID 與您的物件名稱相符，或者您只將憑證連接到物件，則在政策定義中使用政策變數可以簡化政策管理。您可以使用物件政策變數來定義單一政策，而不是為每個 IoT 物件建立個別政策。此政策可以動態套用至所有裝置。以下是顯示其運作方式的範例政策。如需詳細資訊，請參閱???。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Condition": {
    "StringLike": {
     "iot:ClientId": "*${iot:Connection.Thing.Attributes[envType]}"
    }
   },
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/*"
  }
 ]
}
```

AWS IoT Core 如果物件的用戶端 ID 以envType屬性的值結尾，則此政策範例允許物件連線到 。只有具有相符用戶端 ID 模式的物件才能連線。

X.509 憑證 AWS IoT Core 政策變數

X.509 憑證政策變數可協助撰寫 AWS IoT Core 政策。這些政策會根據 X.509 憑證屬性授予許可。下列各節說明如何使用這些憑證政策變數。

> **Important**
>
> 如果您的 X.509 憑證不包含特定的憑證屬性，但您的政策文件中使用了對應的憑證政策變數，則政策評估可能會導致非預期的行為。

CertificateId

在 RegisterCertificate API 中，`certificateId` 會顯示在回應內文中。若要取得憑證的相關資訊，請使用 DescribeCertificate `certificateId`中的 。

發行者屬性

下列 AWS IoT Core 政策變數支援根據憑證發行者設定的憑證屬性，允許或拒絕許可。

- `iot:Certificate.Issuer.DistinguishedNameQualifier`
- `iot:Certificate.Issuer.Country`
- `iot:Certificate.Issuer.Organization`
- `iot:Certificate.Issuer.OrganizationalUnit`
- `iot:Certificate.Issuer.State`
- `iot:Certificate.Issuer.CommonName`
- `iot:Certificate.Issuer.SerialNumber`
- `iot:Certificate.Issuer.Title`
- `iot:Certificate.Issuer.Surname`
- `iot:Certificate.Issuer.GivenName`
- `iot:Certificate.Issuer.Initials`
- `iot:Certificate.Issuer.Pseudonym`

- iot:Certificate.Issuer.GenerationQualifier

主體屬性

下列 AWS IoT Core 政策變數支援根據憑證發行者設定的憑證主體屬性授予或拒絕許可。

- iot:Certificate.Subject.DistinguishedNameQualifier
- iot:Certificate.Subject.Country
- iot:Certificate.Subject.Organization
- iot:Certificate.Subject.OrganizationalUnit
- iot:Certificate.Subject.State
- iot:Certificate.Subject.CommonName
- iot:Certificate.Subject.SerialNumber
- iot:Certificate.Subject.Title
- iot:Certificate.Subject.Surname
- iot:Certificate.Subject.GivenName
- iot:Certificate.Subject.Initials
- iot:Certificate.Subject.Pseudonym
- iot:Certificate.Subject.GenerationQualifier

X.509 憑證為這些屬性提供包含一或多個值的選項。根據預設，每個多值屬性的政策變數會回傳第一個值。例如，Certificate.Subject.Country 屬性可能包含國家/地區名稱的清單，但在政策中評估時，會將 iot:Certificate.Subject.Country 取代為第一個國家/地區的名稱。

您可以使用開頭為一的索引，要求第一個值以外的特定屬性值。例如，在 iot:Certificate.Subject.Country.1 屬性中，第二個國家/地區名稱將取代 Certificate.Subject.Country。若您指定的索引值不存在 (例如，僅對該屬性指派兩個值但您要求第三個值)，將不會進行替換，而授權會失敗。您可以在政策變數名稱添加 .List 尾碼，指定屬性全部的值。

發行者別名屬性

下列 AWS IoT Core 政策變數支援根據憑證發行者設定的發行者替代名稱屬性授予或拒絕許可。

- iot:Certificate.Issuer.AlternativeName.RFC822Name

- iot:Certificate.Issuer.AlternativeName.DNSName
- iot:Certificate.Issuer.AlternativeName.DirectoryName
- iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier
- iot:Certificate.Issuer.AlternativeName.IPAddress

主體別名屬性

下列 AWS IoT Core 政策變數支援根據憑證發行者設定的主題替代名稱屬性授予或拒絕許可。

- iot:Certificate.Subject.AlternativeName.RFC822Name
- iot:Certificate.Subject.AlternativeName.DNSName
- iot:Certificate.Subject.AlternativeName.DirectoryName
- iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier
- iot:Certificate.Subject.AlternativeName.IPAddress

其他屬性

您可以使用 根據憑證的序號，iot:Certificate.SerialNumber允許或拒絕對 AWS IoT Core 資源的存取。iot:Certificate.AvailableKeys 政策變數包含的所有憑證政策變數名稱都具備值。

使用 X.509 憑證政策變數

本主題提供如何使用憑證政策變數的詳細資訊。當您建立 AWS IoT Core 根據 X.509 憑證屬性授予許可的政策時，X.509 憑證政策變數至關重要。如果您的 X.509 憑證不包含特定的憑證屬性，但您的政策文件中使用了對應的憑證政策變數，則政策評估可能會導致非預期的行為。這是因為缺少的政策變數不會在政策陳述式中進行評估。

在本主題中：

- X.509 憑證範例
- 使用憑證發行者屬性做為憑證政策變數
- 使用憑證主體屬性做為憑證政策變數
- 使用憑證發行者替代名稱屬性做為憑證政策變數
- 使用憑證主體替代名稱屬性做為憑證政策變數
- 使用其他憑證屬性做為憑證政策變數
- X.509 憑證政策變數限制

- 使用憑證政策變數的範例政策

X.509 憑證範例

典型的 X.509 憑證可能會出現如下。此範例憑證包含憑證屬性。在評估 AWS IoT Core 政策期間，下列憑證屬性會填入為憑證政策變數：Serial Number、Issuer、X509v3 Issuer Alternative Name、 Subject和 X509v3 Subject Alternative Name。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            92:12:85:cb:b7:a5:e0:86
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=IoT Devices, OU=SmartHome, ST=WA, CN=IoT Devices Primary CA,
    GN=Primary CA1/initials=XY/dnQualifier=Example corp,
    SN=SmartHome/ title=CA1/pseudonym=Primary_CA/generationQualifier=2/serialNumber=987

        Validity
            Not Before: Mar 26 03:25:40 2024 GMT
            Not After : Apr 28 03:25:40 2025 GMT
        Subject: C=US, O=IoT Devices, OU=LightBulb, ST=NY, CN=LightBulb Device Cert,
    GN=Bulb/initials=ZZ/dnQualifier=Bulb001,
    SN=Multi Color/title=RGB/pseudonym=RGB Device/generationQualifier=4/
serialNumber=123
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    << REDACTED >>
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, IP Address:1.2.3.4, URI:ResourceIdentifier001,
 email:device1@example.com, DirName:/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert
            X509v3 Issuer Alternative Name:
                DNS:issuer.com, IP Address:5.6.7.8, URI:PrimarySignerCA,
 email:primary@issuer.com, DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA
    Signature Algorithm: sha256WithRSAEncryption
```

```
        << REDACTED >>
```

使用憑證發行者屬性做為憑證政策變數

下表提供憑證發行者屬性如何填入 AWS IoT Core 政策的詳細資訊。

要填入政策中的發行者屬性

憑證發行者屬性	憑證政策變數
• C=US • O=IoT 裝置 • OU=SmartHome • ST=WA • CN=IoT 裝置主要 CA • GN=主要 CA1 • initials=XY • dnQualifier=範例 corp • SN=SmartHome • title=CA1 • pseudonym=Primary_CA • generationQualifier=2 • serialNumber=987	• `iot:Certificate.Issuer.Country = US` • `iot:Certificate.Issuer.Organization = IoT Devices` • `iot:Certificate.Issuer.OrganizationalUnit = SmartHome` • `iot:Certificate.Issuer.State = WA` • `iot:Certificate.Issuer.CommonName = IoT Devices Primary CA` • `iot:Certificate.Issuer.GivenName = Primary CA1` • `iot:Certificate.Issuer.initials = XY` • `iot:Certificate.Issuer.Distinguished NameQualifier = Example corp` • `iot:Certificate.Issuer.Surname = SmartHome` • `iot:Certificate.Issuer.Title = CA1` • `iot:Certificate.Issuer.Pseudonym = Primary_CA` • `iot:Certificate.Issuer.GenerationQualifier = 2` • `iot:Certificate.Issuer.SerialNumber = 987`

使用憑證主體屬性做為憑證政策變數

下表提供如何在 AWS IoT Core 政策中填入憑證主體屬性的詳細資訊。

要在政策中填入的主題屬性

憑證主體屬性	憑證政策變數
• C=US • O=IoT 裝置 • ST=NY • CN=LightBulb 裝置憑證 • GN=Bulb • initials=ZZ • dnQualifier=Bulb001 • SN=Multi Color • title=RGB • pseudonym=RGB 裝置 • generationQualifier=4 • serialNumber=123	• `iot:Certificate.Subject.Country = US` • `iot:Certificate.Subject.Organization = IoT Devices` • `iot:Certificate.Subject.State = NY` • `iot:Certificate.Subject.CommonName = LightBulb Device Cert` • `iot:Certificate.Subject.GivenName = Bulb` • `iot:Certificate.Subject.initials = ZZ` • `iot:Certificate.Subject.DistinguishedNameQualifier = Bulb001` • `iot:Certificate.Subject.Surname = Multi Color` • `iot:Certificate.Subject.Title = RGB` • `iot:Certificate.Subject.Pseudonym = RGB Device` • `iot:Certificate.Subject.GenerationQualifier = 4` • `iot:Certificate.Subject.SerialNumber = 123`

使用憑證發行者替代名稱屬性做為憑證政策變數

下表提供憑證發行者替代名稱屬性如何填入 AWS IoT Core 政策的詳細資訊。

要填入政策中的發行者替代名稱屬性

X509v3 發行者替代名稱	政策中的屬性
• DNS：issuer.com • IP 地址：5.6.7.8 • URI：PrimarySignerCA	• `iot:Certificate.Issuer.AlternativeName.DNSName = issuer.com` • `iot:Certificate.Issuer.AlternativeName.IPAddress = 5.6.7.8`

X509v3 發行者替代名稱	政策中的屬性
• 電子郵件：primary@issuer.com • DirName：/C=US/O=發行者/OU=IoT 裝置/CN=主要發行者 CA	• iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier = PrimarySignerCA • iot:Certificate.Issuer.AlternativeName.RFC822Name = primary@issuer.com • iot:Certificate.Issuer.AlternativeName.DirectoryName = cn=Primary Issuer CA,ou=IoT Devices,o=Issuer,c=US

使用憑證主體替代名稱屬性做為憑證政策變數

下表提供如何在政策中 AWS IoT Core 填入憑證主體替代名稱屬性的詳細資訊。

要在政策中填入的主題替代名稱屬性

X509v3 主體替代名稱	政策中的屬性
• DNS：example.com • IP 地址：1.2.3.4 • URI：ResourceIdentifier001 • 電子郵件：device1@example.com • DirName：/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert	• iot:Certificate.Subject.AlternativeName.DNSName = example.com • iot:Certificate.Subject.AlternativeName.IPAddress = 1.2.3.4 • iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier = ResourceIdentifier001 • iot:Certificate.Subject.AlternativeName.RFC822Name = device1@example.com • iot:Certificate.Subject.AlternativeName.DirectoryName = cn=LightBulbCert,ou=SmartHome,o=IoT,c=US

使用其他憑證屬性做為憑證政策變數

下表提供如何在 AWS IoT Core 政策中填入其他憑證屬性的詳細資訊。

要在政策中填入的其他屬性

其他憑證屬性	憑證政策變數
`Serial Number: 92:12:85:cb:b7:a5:e0:86`	`iot:Certificate.SerialNumber = 105256223 89124227206`

X.509 憑證政策變數限制

以下限制適用於 X.509 憑證政策變數：

缺少政策變數

如果您的 X.509 憑證不包含特定的憑證屬性，但您的政策文件中使用了對應的憑證政策變數，則政策評估可能會導致非預期的行為。這是因為缺少的政策變數不會在政策陳述式中進行評估。

Certificate SerialNumber 格式

AWS IoT Core 會將憑證序號視為十進位整數的字串表示法。例如，如果政策只允許用戶端 ID 與憑證序號相符的連線，則用戶端 ID 必須是十進位格式的序號。

萬用字元

如果憑證屬性中存在萬用字元，則不會以憑證屬性值取代政策變數。這會將`${policy-variable}`文字保留在政策文件中。如此可能導致授權失敗。可使用下列萬用字元：*、$、+、？及 #。

陣列欄位

具備陣列的憑證屬性僅限五個項目，其他項目會遭到忽略。

字串長度

所有字串值上限為 1024 個字元。如果憑證屬性包含超過 1024 個字元的字串，則不會以憑證屬性值取代政策變數。這會將 保留`${policy-variable}`在政策文件中。如此可能導致授權失敗。

特殊字元

在政策變數中使用時，`,`、`"`、`\`、+、=、<、> 及 ; 等任何特殊字元的字首必須加上反斜線 (\)。例如，`Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US` 會變成 `Amazon Web Service O\=Amazon.com Inc. L\=Seattle ST\=Washington C \=US`。

使用憑證政策變數的範例政策

下列政策文件允許用戶端 ID 符合憑證序號的連線，並發佈至符合模式的主題：`${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*`。

> **Important**
>
> 如果您的 X.509 憑證不包含特定的憑證屬性，但您的政策文件中使用了對應的憑證政策變數，則政策評估可能會導致非預期的行為。這是因為缺少的政策變數不會在政策陳述式中進行評估。例如，如果您將下列政策文件連接到不包含 `iot:Certificate.Subject.Organization` 屬性的憑證，則不會在政策評估期間填入`iot:Certificate.Subject.Organization`憑證政策變數。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Certificate.SerialNumber}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*"
   ]
  }
 ]
}
```

您也可以使用 Null 條件運算子，以確保政策中使用的憑證政策變數會在政策評估期間填入。下列政策文件只有在憑證序號和憑證主體通用名稱屬性存在時，才允許 iot:Connect 搭配憑證使用 。

所有憑證政策變數都有字串值，因此支援所有字串條件運算子。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/*"
   ],
   "Condition": {
    "Null": {
     "iot:Certificate.SerialNumber": "false",
     "iot:Certificate.Subject.CommonName": "false"
    }
   }
  }
 ]
}
```

預防跨服務混淆代理人

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多權限的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了預防這種情況， AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

若要限制將另一個服務 AWS IoT 提供給資源的許可，我們建議在資源政策中使用 aws:SourceArn和 aws:SourceAccount全域條件內容金鑰。如果同時使用全域條件內容索引鍵，則在相同政策陳述式中使用aws:SourceAccount 值和 aws:SourceArn 值中的帳戶時，必須使用相同的帳戶 ID。

防範混淆代理人問題的最有效方法是使用 aws:SourceArn 全域條件內容索引鍵，其中包含資源的完整 Amazon Resource Name (ARN)。對於 AWS IoT，您的 aws:SourceArn 必須符合格

式：arn:*aws*:iot:*region*:*account-id*:*resource-type/resource-id*適用於資源特定許可或 arn:*aws*:iot:*region*:*account-id*:*。resource-id 可以是允許資源的名稱或 ID，也可以是允許資源 IDs的萬用字元陳述式。請確定該*##符合您的 AWS IoT 區域，且*## *ID* 符合您的客戶帳戶 ID。

下列範例示範如何使用 AWS IoT 角色信任政策中的 aws:SourceArn和 aws:SourceAccount全域條件內容索引鍵，來防止混淆代理人問題。如需更多範例，請參閱預防混淆代理人的詳細範例。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-east-1:123456789012:*"
        }
         }
      }
   ]
}
```

> **Note**
>
> 如果您收到存取拒絕錯誤，可能是因為與 AWS Security Token Service (STS) 的服務整合不支援 aws:SourceArn和 aws:SourceAccount 內容金鑰。

預防混淆代理人的詳細範例

本節提供詳細範例，說明如何使用 AWS IoT 角色信任政策中的 aws:SourceArn和 aws:SourceAccount 全域條件內容索引鍵來防止混淆代理人問題。

- 機群佈建

- JITP
- 登入資料提供者

機群佈建

您可以使用佈建範本資源來設定機群佈建。當佈建範本參考佈建角色時，該角色的信任政策可以包含 aws:SourceArn和 aws:SourceAccount條件金鑰。這些金鑰會限制組態可以調用sts:AssumeRole請求的資源。

具有下列信任政策的角色只能由 IoT 主體 (iot.amazonaws.com) 擔任 中指定的佈建範本SourceArn。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:provisioningtemplate/example_template"
         }
          }
      }
   ]
}
```

JITP

在just-in-time佈建 (JITP) 中，您可以使用佈建範本做為與 CA 分開的資源，或定義範本內文和角色做為 CA 憑證組態的一部分。 AWS IoT 角色信任政策aws:SourceArn中的 值取決於您如何定義佈建範本。

將佈建範本定義為個別資源

如果您將佈建範本定義為單獨的資源，則 的值aws:SourceArn可以是 "arn:aws:iot:*region*:*account-id*:provisioningtemplate/*example_template*"。您可以在 中使用相同的政策範例機群佈建。

定義 CA 憑證中的佈建範本

如果您在 CA 憑證資源中定義佈建範本，則 的值aws:SourceArn可以是 "arn:aws:iot:*region*:*account-id*:cacert/*cert_id*"或 "arn:aws:iot:*region*:*account-id*:cacert/*"。您可以在建立時未知資源識別符時使用萬用字元，例如 CA 憑證的 ID。

具有下列信任政策的角色只能由 IoT 主體 (iot.amazonaws.com) 擔任 中指定的 CA 憑證SourceArn。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:cacert/
8ecde6884f3d87b1125ba31ac3fcb13d7016de7f57cc904fe1cb97c6ae98196e"
        }
         }
      }
   ]
}
```

建立 CA 憑證時，您可以在註冊組態中參考佈建角色。佈建角色的信任政策可以使用 `aws:SourceArn`來限制角色可以擔任哪些資源。不過，在註冊 CA 憑證的初始 RegisterCACertificate 呼叫期間，您沒有條件中要指定的 CA 憑證 ARN`aws:SourceArn`。

若要解決此問題，例如，若要將佈建角色信任政策指定給向 註冊的特定 CA 憑證 AWS IoT Core，您可以執行下列動作：

- 首先，呼叫 RegisterCACertificate 而不提供 `RegistrationConfig` 參數。
- 向 CA 憑證註冊後 AWS IoT Core，請呼叫 UpdateCACertificate。

  在 UpdateCACertificate 呼叫中，提供 `RegistrationConfig`，其中包含佈建角色信任政策，並將 `aws:SourceArn`設定為新註冊 CA 憑證的 ARN。

登入資料提供者

對於AWS IoT Core 登入資料提供者，請使用 AWS 帳戶 您在 中建立角色別名的相同 `aws:SourceAccount`，並指定符合 中角色別名資源類型之資源 ARN 的陳述式`aws:SourceArn`。建立 IAM 角色以搭配 AWS IoT Core 登入資料提供者使用時，您必須在`aws:SourceArn`條件中包含可能需要擔任該角色之任何角色別名的 ARNs，藉此授權跨服務`sts:AssumeRole`請求。

具有下列信任政策的角色只能由 中指定之 roleAlias 的 AWS IoT Core 登入資料提供者 (`credentials.iot.amazonaws.com`) 主體擔任`SourceArn`。如果委託人嘗試擷取`aws:SourceArn`條件中指定以外之角色別名的登入資料，則請求將被拒絕，即使該其他角色別名參考相同的 IAM 角色。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
```

```
          "aws:SourceArn": "arn:aws:iot:us-
east-1:123456789012:rolealias/example_rolealias"
        }
      }
    }
  ]
}
```

AWS IoT Core 政策範例

本節中的範例政策說明在 AWS IoT Core中完成一般任務所用的政策文件。在為解決方案建立政策時，您可以將其作為範例來開始。

本節中的範例使用這些政策元素：

- the section called “AWS IoT Core 政策動作”
- the section called “AWS IoT Core 動作資源”
- the section called “身分型政策範例”
- the section called “基本 AWS IoT Core 政策變數”
- the section called “X.509 憑證 AWS IoT Core 政策變數”

本節中的政策範例：

- 連接政策範例
- 發佈/訂閱政策範例
- 連線和發佈政策範例
- 保留訊息政策範例
- 憑證政策範例
- 物件政策範例
- 基本任務政策範例
- 直接傳訊政策範例

連接政策範例

下列政策拒絕用戶端 IDs`client1`和 連線`client2`的許可 AWS IoT Core，同時允許裝置使用用戶端 ID 連線。用戶端 ID 符合在 AWS IoT Core 登錄檔中註冊並連接到用於連線之主體的物件名稱：

> **Note**
>
> 對於已註冊的裝置，建議您使用 Connect 動作的物件政策變數，並將物件附加至用於連線的主體。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1",
    "arn:aws:iot:us-east-1:123456789012:client/client2"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  }
 ]
}
```

下列政策會授予使用 AWS IoT Core 用戶端 ID 連線至 的許可client1。此政策範例適用於未註冊的裝置。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1"
   ]
  }
 ]
}
```

MQTT 持續工作階段政策範例

connectAttributes 可讓您在 IAM 政策中指定要在連線訊息中使用的屬性，例如 PersistentConnect 和 LastWill。如需詳細資訊，請參閱使用 connectAttributes。

下列政策允許與 PersistentConnect 功能連接：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
```

```
}
```

下列政策不允許 PersistentConnect，允許其他功能：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringNotEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

上述政策也可以使用 StringEquals 表達，並允許任何其他功能，包括新功能：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1"
        },
        {
            "Effect": "Deny",
            "Action": [
```

```
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

下列政策允許透過 PersistentConnect 和 LastWill 連接，但不允許任何其他新功能：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  }
 ]
}
```

下列政策允許透過具有或沒有 LastWill 的用戶端進行全新連接，但不允許任何其他功能：

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iot:Connect"
        ],
        "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
        "Condition": {
            "StringEquals": {
                "iot:ConnectAttributes": "LastWill"
        }
        }
    }]
}
```

下列政策只允許使用預設功能進行連接：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

下列政策僅允許與 PersistentConnect 連接，而且只要連線使用 PersistentConnect，就允許任何新功能：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

下列政策指出連接必須同時使用 PersistentConnect 和 LastWill，而且不允許任何新功能：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  },
  {
```

```
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

下列政策不得具有 PersistentConnect，但可以具有 LastWill，而且不允許任何其他新功能：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
            }
        }
        }
    ]
}
```

下列政策只允許透過具有 LastWill 與主題 "my/lastwill/topicName" 的用戶端連接，而且只要其使用 LastWill 主題，就允許任何功能：

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        }
    ]
}
```

下列政策僅允許使用特定的 LastWillTopic 進行全新連接，而且只要其使用 LastWillTopic，就允許任何功能：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

發佈/訂閱政策範例

您使用的政策取決於您的連線方式 AWS IoT Core。您可以使用 MQTT AWS IoT Core 用戶端、HTTP 或 WebSocket 連線至 。當您使用 MQTT 用戶端連接，必須使用 X.509 憑證進行驗證。當您透過 HTTP 或 WebSocket 通訊協定連接時，必須使用 Signature 第 4 版和 Amazon Cognito 進行驗證。

> **Note**
>
> 對於已註冊的裝置，建議您使用 `Connect` 動作的物件政策變數，並將物件附加至用於連線的主體。

在本節中：

- 在 MQTT 和 AWS IoT Core 政策中使用萬用字元
- 向/從特定主題發佈、訂閱及接收訊息的政策
- 向/從具有特定前綴主題發佈、訂閱及接收訊息的政策
- 向/從各項裝置特定主題發佈、訂閱及接收訊息的政策
- 向/從主題名稱中具有物件屬性的主題發佈、訂閱和接收訊息的政策
- 拒絕向特定主題名稱的子主題進行發佈的政策
- 拒絕從特定主題名稱的子主題進行接收的政策
- 從使用 MQTT 萬用字元的主題進行訂閱的政策
- 適用 HTTP 和 WebSocket 用戶端的政策

在 MQTT 和 AWS IoT Core 政策中使用萬用字元

MQTT 和 AWS IoT Core 政策具有不同的萬用字元，您應該在仔細考慮之後選擇它們。在 MQTT 中，萬用字元 +和 # 用於 MQTT 主題篩選條件，以訂閱多個主題 name. AWS IoT Core policies 使用 * 和 ?做為萬用字元，並遵循 IAM 政策的慣例。在政策文件中，* 代表任意字元組合，? 則代表任何單一字元。在政策文件中，MQTT 萬用字元 + 和 # 視為沒有特殊含義的字元。若要在政策中的 `resource` 屬性描述多個主題名稱和主題篩選條件，請使用 * 和 ? 萬用字元代替 MQTT 萬用字元。

當您選擇要在政策文件中使用的萬用字元時，請考慮該*字元不限於單一主題層級。+ 字元僅限於 MQTT 主題篩選條件中的單一主題層級。要幫助將萬用字元規範限制為單一 MQTT 主題篩選條件層級，請考慮使用多個 ? 字元。更多有關在政策資源中使用萬用字元的資訊以及它們比對的更多範例，請參閱在資源 ARN 中使用萬用字元。

下表顯示 MQTT 用戶端的 AWS IoT Core 政策中使用的不同萬用字元。

萬用字元	是 MQTT 萬用字元	MQTT 中的範例	是 AWS IoT Core 政策萬用字元	MQTT 用戶端 AWS IoT Core 政策中的範例
#	是	`some/#`	否	N/A
+	是	`some/+/topic`	否	不適用
*	否	N/A	是	`topicfilter/some/*/topic` `topicfilter/some/sensor*/topic`
?	否	N/A	是	`topic/some/?????/topic` `topicfilter/some/sensor???/topic`

向/從特定主題發佈、訂閱及接收訊息的政策

以下範例顯示已註冊和未註冊裝置向/從名為 "some_specific_topic" 的主題發佈、訂閱和接收訊息的情形。這些範例也突顯 `Publish` 和 `Receive` 使用 "topic" 作為資源，以及 `Subscribe` 使用 "topicfilter" 作為資源。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。針對名為 "some_specific_topic" 的主題，它也提供 `Publish`、`Subscribe` 和 `Receive` 的許可。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/some_specific_topic"
   ]
  },
  {
   "Effect": "Allow",
```

```
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
   ]
  }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。針對名為 "some_specific_topic" 的主題，它也提供 `Publish`、`Subscribe` 和 `Receive` 的許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/
some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        }
    ]
}
```

向/從具有特定前綴主題發佈、訂閱及接收訊息的政策

以下範例顯示已註冊和未註冊裝置向/從具有 "topic_prefix" 前綴的主題發佈、訂閱和接收訊息的情形。

> **Note**
>
> 請注意*，在此範例中使用萬用字元。雖然在單一陳述式中提供多個主題名稱的許可*非常有用，但透過為裝置提供比所需更多的權限，可能會導致意外的後果。因此，我們建議您在仔細考慮*之後，才使用萬用字元。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。針對以 "topic_prefix" 為前綴的主題，它也提供 `Publish`、`Subscribe` 和 `Receive` 的許可。

```
{
 "Version":"2012-10-17",
 "Statement": [
```

```
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
   ]
  }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。它也為以 "topic_prefix" 為前綴的主題提供 `Publish`、`Subscribe` 和 `Receive` 的許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
            ]
        }
    ]
}
```

向/從各項裝置特定主題發佈、訂閱及接收訊息的政策

以下範例顯示已註冊和未註冊裝置向/從特定裝置專屬的主題發佈、訂閱和接收訊息的情形。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。它提供向物件特定主題 (sensor/device/${iot:Connection.Thing.ThingName}) 進行發佈的許可，以及從物件特定主題 (command/device/${iot:Connection.Thing.ThingName}) 進行訂閱和接收的許可。如果登錄檔中的物件名稱是「thing1」，裝置將能夠發佈到主題「sensor/device/thing1」。裝置也可以訂閱主題「command/device/thing1」並從中接收。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
```

```
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。它提供向用戶端特定主題 (sensor/device/${iot:ClientId}) 進行發佈的許可，以及從用戶端特定主題 (command/device/${iot:ClientId}) 進行訂閱和接收的許可。如果裝置以 clientId1 連接 clientId，它將能夠發佈到主題 "sensor/device/clientId1"。裝置也可以從主題 訂閱和接收 device/clientId1/command。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

向/從主題名稱中具有物件屬性的主題發佈、訂閱和接收訊息的政策

以下範例顯示已註冊裝置向/從名稱含有物件屬性的主題發佈、訂閱和接收訊息的情形。

> **Note**
>
> 物件屬性僅存在於登錄檔中 AWS IoT Core 註冊的裝置。未註冊的裝置沒有對應的範例。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。它提供向主題 (sensor/${iot:Connection.Thing.Attributes[version]}) 進行發佈的許可，以及從主題名稱含有物件屬性的主題 (command/${iot:Connection.Thing.Attributes[location]}) 進行訂閱和接收的許可。如果登錄檔中的物件名稱具有 version=v1和 location=Seattle，裝置將能夠發佈至主題 "sensor/v1"，並從主題 "mand/Seattle" 訂閱和接收。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/
${iot:Connection.Thing.Attributes[version]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
```

```
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  }
 ]
}
```

Unregistered devices

由於物件屬性僅存在於 AWS IoT Core 登錄檔中註冊的裝置，因此沒有未登錄物件的對應範例。

拒絕向特定主題名稱的子主題進行發佈的政策

以下範例顯示已註冊和未註冊裝置向特定子主題以外的所有主題發佈訊息的情形。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。它提供向所有以 "department/" 為前綴的主題進行發佈的許可，但未及於 "department/admins" 子主題。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
```

```
    ],
    "Condition": {
     "Bool": {
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/department/*"
    ]
   },
   {
    "Effect": "Deny",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
    ]
   }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。它提供向所有以 "department/" 為前綴的主題進行發佈的許可，但未及於 "department/admins" 子主題。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
            ]
        }
    ]
}
```

拒絕從特定主題名稱的子主題進行接收的政策

以下範例顯示已註冊和未註冊裝置從特定子主題以外所有具特定前綴主題訂閱和接收訊息的情形。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。此政策允許裝置訂閱任何前綴為 "topic_prefix" 的主題。藉由在 `iot:Receive` 的陳述式中使用，`NotResource`我們允許裝置接收所有來自其已訂閱主題的訊息，但前綴為 "topic_prefix/restricted" 的主題除外。例如在此政策中，裝置可以訂閱 "topic_prefix/topic1" 甚至 "topic_prefix/restricted"，但是它們只會接收來自 "opic_prefix/topic1" 主題的訊息，且不會接收來自 "topic_prefix/restricted" 主題的訊息。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix/*"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix/
restricted/*"
  }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。此政策允許裝置訂閱任何前綴為 "topic_prefix" 的主題。藉由在 `iot:Receive` 的陳述式中使用 `NotResource`，我們允許裝置接收所有來自其已訂閱主題的訊息，但前綴為 "topic_prefix/restricted" 的主題除外。例如，使用此政策，裝置可以訂閱 "topic_prefix/topic1" 甚至 "topic_prefix/restricted"。不過，他們只會收到主題 "topic_prefix/topic1" 的訊息，而不會收到主題 "topic_prefix/restricted" 的訊息。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
topic_prefix/*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/
topic_prefix/restricted/*"
        }
    ]
}
```

從使用 MQTT 萬用字元的主題進行訂閱的政策

MQTT 萬用字元 + 和 # 視為文字字串，但在 AWS IoT Core 政策中使用時不會視為萬用字元。在 MQTT 中，僅在訂閱主題篩選條件時才會將 + 和 # 視為萬用字元，但在所有其他情況下視為常值字串。我們建議您在仔細考慮之後，才使用這些 MQTT 萬用字元做為 AWS IoT Core 政策的一部分。

以下顯示使用 MQTT 萬用字元在 AWS IoT Core 政策中註冊和未註冊物件的範例。這些萬用字元會視為常值字串。

Registered devices

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用與登錄檔中物件名稱相符的 clientId 進行連線。此政策允許裝置訂閱 "department/+/employees" 和 "location/#" 主題。由於 + 和 # 在 AWS IoT Core 政策中被視為文字字串，裝置可以訂閱主題「department/+/employees」，但不能訂閱主題「department/engineering/employees」。同樣，裝置可以訂閱 "location/#" 主題，但不能訂閱 "location/Seattle" 主題。但是，一旦裝置訂閱了 "department/+/employees" 主題，此政策將允許其接收來自 "department/engineering/employees" 主題的訊息。同樣，一旦裝置訂閱了 "location/#" 主題，其也將接收來自 "location/Seattle" 主題的訊息。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/department/+/
employees"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/#"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
```

```
  }
 ]
}
```

Unregistered devices

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策允許裝置使用 clientId1、 clientId2 或 clientId3 進行連線。此政策允許裝置訂閱 "department/+/employees" 和 "location/#" 主題。由於 + 和 # 在 AWS IoT Core 政策中被視為文字字串，裝置可以訂閱主題「department/+/employees」，但不能訂閱主題「department/engineering/employees」。同樣，裝置可以訂閱 "location/#" 主題，但不能訂閱 "location/Seattle"。但是，一旦裝置訂閱了 "department/+/employees" 主題，此政策將允許其接收來自 "department/engineering/employees" 主題的訊息。同樣，一旦裝置訂閱了 "location/#" 主題，其也將接收來自 "location/Seattle" 主題的訊息。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
department/+/employees"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/
#"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        }
    ]
}
```

適用 HTTP 和 WebSocket 用戶端的政策

當您透過 HTTP 或 WebSocket 通訊協定連接時，必須使用 Signature 第 4 版和 Amazon Cognito 進行驗證。Amazon Cognito 身分可以是已驗證或未驗證的身分。已驗證的身分屬於已由任何支援的身分提供者驗證的使用者。未驗證的身分通常屬於未向身分提供者驗證身分的訪客使用者。Amazon Cognito 提供唯一識別符和 AWS 登入資料，以支援未經驗證的身分。如需詳細資訊，請參閱the section called “使用 Amazon Cognito 身分授權”。

對於下列操作， AWS IoT Core 會使用透過 `AttachPolicy` API 連接到 Amazon Cognito 身分 AWS IoT Core 的政策。這會縮小連接到具有已驗證身分之 Amazon Cognito 身分集區的許可範圍。

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

這表示 Amazon Cognito Identity 需要 IAM 角色政策和 AWS IoT Core 政策的許可。您可以透過 `AttachPolicy` API 將 IAM 角色政策連接至集區，並將 AWS IoT Core 政策連接至 Amazon Cognito Identity AWS IoT Core 。

已驗證和未驗證的使用者是不同的身分類型。如果您未將 AWS IoT 政策連接至 Amazon Cognito 身分，已驗證的使用者會失敗授權， AWS IoT 而且無法存取 AWS IoT 資源和動作。

> **Note**
>
> 對於其他 AWS IoT Core 操作或未經驗證的身分， AWS IoT Core 不會縮小連接到 Amazon Cognito 身分集區角色的許可範圍。對於已驗證和未驗證兩者的身分而言，這是我們針對連接至 Amazon Cognito 集區角色的政策，所提供的最寬容建議。

HTTP

若要允許未驗證 Amazon Cognito 身分透過 HTTP 對 Amazon Cognito 身分特定的主題發佈訊息，請將下列 IAM 政策連接至 Amazon Cognito 身分集區角色：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"
            ]
        }
    ]
}
```

若要允許已驗證的使用者，請使用 AWS IoT Core AttachPolicy API 將上述政策連接至 Amazon Cognito 身分集區角色和 Amazon Cognito 身分。

> **Note**
>
> 授權 Amazon Cognito 身分時， AWS IoT Core 請考慮政策並授予指定的最低權限。只有在兩個政策都允許要求的動作時，才會允許執行動作。如果兩個政策都禁止動作，則該動作為未授權。

MQTT

若要允許未驗證 Amazon Cognito 身分透過 WebSocket 對您帳戶中 Amazon Cognito 身分特定的主題發佈 MQTT 訊息，請將下列 IAM 政策連接至 Amazon Cognito 身分集區角色：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

若要允許已驗證的使用者，請使用 AWS IoT Core AttachPolicy API 將上述政策連接至 Amazon Cognito 身分集區角色和 Amazon Cognito 身分。

> **Note**
>
> 授權 Amazon Cognito 身分時， AWS IoT Core 同時考慮 和 會授予指定的最低權限。只有在兩個政策都允許要求的動作時，才會允許執行動作。如果兩個政策都禁止動作，則該動作為未授權。

連線和發佈政策範例

對於在 AWS IoT Core 登錄檔中註冊為實物的裝置，下列政策會授予許可，以 AWS IoT Core 使用與實物名稱相符的用戶端 ID 連線至 ，並限制裝置在用戶端 ID 或實物名稱特定的 MQTT 主題上發佈。若要

讓連線成功，物件名稱必須在 AWS IoT Core 登錄檔中註冊，並使用連接到物件的身分或主體進行身分驗證：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:Connection.Thing.ThingName}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

對於未在 AWS IoT Core 登錄檔中註冊為實物的裝置，下列政策會授予使用 AWS IoT Core 用戶端 ID 連線至 的許可，`client1`並限制裝置在 clientID 特定的 MQTT 主題上發佈：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
```

```
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/client1"
      ]
    }
  ]
}
```

保留訊息政策範例

使用保留訊息需要特定的政策。保留的訊息是使用 RETAIN 旗標設定並存放的 MQTT 訊息 AWS IoT Core。本節提供允許保留訊息常見用途的政策範例。

在本節中：

- 連線和發佈保留訊息的政策
- 連線和發佈保留 Will 訊息的政策
- 列出和取得保留訊息的政策

連線和發佈保留訊息的政策

對於發佈保留訊息的裝置，裝置必須能夠連線和發佈任何 MQTT 訊息，也能發佈 MQTT 保留訊息。下列政策會授予以下主題的許可：`device/sample/configuration` 至用戶端 **`device1`**。如需授予連線許可的其他範例，請參閱 the section called “連線和發佈政策範例”。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
```

```
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/device1"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish",
     "iot:RetainPublish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/device/sample/configuration"
    ]
   }
  ]
}
```

連線和發佈保留 Will 訊息的政策

用戶端可以設定訊息，AWS IoT Core 在用戶端意外中斷連線時發佈。MQTT 會將這類訊息稱為 Will 訊息。用戶端必須將擁有的其他條件新增至其連線許可，才能包含這些條件。

下列政策文件會授予所有用戶端連線和發佈 Will 訊息的許可；此類訊息由其主題 (`will`) 識別，AWS IoT Core 也將保留。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ],
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
```

```
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/will"
   ]
  }
 ]
}
```

列出和取得保留訊息的政策

服務和應用程式可以透過呼叫 ListRetainedMessages 和 GetRetainedMessage 來存取保留訊息，不需支援 MQTT 用戶端。必須透過使用如下範例的政策來對呼叫這些動作的服務和應用程式進行授權。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:ListRetainedMessages"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/device1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetRetainedMessage"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo"
            ]
```

```
        }
    ]
}
```

憑證政策範例

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以 AWS IoT Core 使用與物件名稱相符的用戶端 ID 連線至 ，並發佈至名稱等於裝置用來驗證自己的憑證`certificateId`的 主題：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以 AWS IoT Core 使用用戶端 IDs、`client2`、 `client1client3`和 連線至 ，以發佈至其名稱等於用於驗證裝置本身之憑證`certificateId`的 主題：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以 AWS IoT Core 使用符合物件名稱的用戶端 ID 連線至 ，並發佈至其名稱等於裝置用來驗證其身分之憑證主體CommonName欄位的主題：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
```

```
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

> **Note**
>
> 在這個範例中，憑證的主體通用名稱欄位會用作為主題識別符，並假設主體通用名稱對每個登錄憑證是唯一的。如果憑證在多個裝置間共用，所有共用此憑證之裝置的主體通用名稱都是相同的，因此允許從多個裝置對相同主題的發佈權限 (不建議)。

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以 AWS IoT Core 使用用戶端 IDs、`client2`、 `client1client3`和 連線至 ，並發佈至其名稱等於裝置用來驗證其身分之憑證主體`CommonName`欄位的主題：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
```

```
}
```

> **Note**
>
> 在這個範例中，憑證的主體通用名稱欄位會用作為主題識別符，並假設主體通用名稱對每個登錄憑證是唯一的。如果憑證在多個裝置間共用，所有共用此憑證之裝置的主體通用名稱都是相同的，因此允許從多個裝置對相同主題的發佈權限 (不建議)。

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以 AWS IoT Core 使用與物件名稱相符的用戶端 ID 連線至 ，並在用來驗證裝置的憑證將 `Subject.CommonName.2` 欄位設定為 `admin/`時，發佈至其名稱字首為 的主題`Administrator`：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
            }
        }
        }
    ]
}
```

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，以使用 AWS IoT Core 用戶端 IDs client1、 和 連線至 client2，client3並在用於驗證裝置的憑證將其Subject.CommonName.2欄位設定為 admin/時，發佈至其名稱字首為 的主題Administrator：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
                }
        }
        }
    ]
}
```

對於在 AWS IoT Core 登錄檔中註冊的裝置，以下政策允許裝置使用其實物名稱發佈到特定主題，該主題包含 admin/，當用於驗證裝置的憑證將其任何一個Subject.CommonName欄位設定為ThingName時，後面接著 Administrator：

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

對於未在 AWS IoT Core 登錄檔中註冊的裝置，當用於驗證裝置的憑證將其任何`Subject.CommonName`欄位設定為 `admin`時`client1`，下列政策會授予許可，以 AWS IoT Core 使用用戶端 IDs `client2`、 `client3`和 連線至 ，並發佈至 主題`Administrator`：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

物件政策範例

如果用於向 驗證的憑證 AWS IoT Core 連接到正在評估政策的物件，則下列政策允許裝置連線：

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
            }
        }
        }
    ]
}
```

下列政策在憑證連接至具有特定物件類型的物件，且該物件的 `attributeName` 屬性具有 `attributeValue` 值時，允許裝置發佈。如需物件政策變數的詳細資訊，請參閱物件政策變數。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/device/stats",
      "Condition": {
        "StringEquals": {
          "iot:Connection.Thing.Attributes[attributeName]": "attributeValue",
          "iot:Connection.Thing.ThingTypeName": "Thing_Type_Name"
        },
        "Bool": {
          "iot:Connection.Thing.IsAttached": "true"
        }
      }
    }
  ]
}
```

下列政策允許裝置發佈至以物件屬性開頭的主題。如果裝置憑證與物件沒有關聯，則無法解析此變數，並會導致存取遭拒錯誤。如需物件政策變數的詳細資訊，請參閱物件政策變數。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.Attributes[attributeName]}/*"
    }
  ]
}
```

基本任務政策範例

此範例展示任務目標所需的政策陳述式，該目標為單一裝置，可接收任務請求並與 AWS IoT溝通任務執行狀態。

將 *us-west-2#57EXAMPLE833* 取代為您的 AWS 區域、冒號字元 (:) 和 12 位數字 AWS 帳戶 ，然後將 *uniqueThingName* 取代為代表裝置所在的物件資源名稱 AWS IoT。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/jobExecution/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/events/
jobExecution/*",
```

```
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

直接傳訊政策範例

使用直接傳訊需要特定政策。直接傳訊使用 SendDirectMessage HTTP API，將訊息從寄件者傳送到由 MQTT 用戶端 ID 識別的單一接收者。本節提供允許直接傳訊常用的政策範例。

在本節中：

- 在特定主題上傳送直接訊息給特定用戶端的政策
- 接收直接訊息的政策
- 將直接訊息傳送至特定主題上任何用戶端的政策
- 將直接訊息傳送至任何主題上任何用戶端的政策

在特定主題上傳送直接訊息給特定用戶端的政策

若要讓寄件者傳送直接訊息，寄件者必須具有目標用戶端 ID 做為資源的iot:SendDirectMessage許可。iot:Topic 條件索引鍵 （選用） 會限制寄件者可以傳送訊息的主題。

- 對於SigV4-authenticated後端伺服器，請將此新增至 IAM 政策。
- 對於 X.509 驗證的 IoT 裝置，將此新增至 AWS IoT Core 政策。
- 對於自訂授權方驗證的用戶端，Lambda 函數必須傳回使用 iot:Topic條件索引鍵在目標用戶端資源iot:SendDirectMessage上授予的政策文件

下列政策允許用戶端將主題 commands/reboot和 myDevice的直接訊息device1傳送給用戶端commands/update。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/myDevice",
            "Condition": {
                "StringEquals": {
                    "iot:Topic": ["commands/reboot", "commands/update"]
                }
            }
        }
    ]
}
```

接收直接訊息的政策

接收者的政策必須授予 iot:Receive 主題。接收者不需要iot:Subscribe許可 - 無需主題訂閱即可 AWS IoT Core 傳遞直接訊息。接收者可以使用 X.509 用戶端憑證 (AWS IoT Core 政策） 或 SigV4 (IAM 政策） 進行身分驗證。在這兩種情況下，接收主題都需要 iot:Receive許可。

下列政策允許接收者用戶端myDevice接收有關主題 commands/reboot和 的直接訊息commands/update。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/reboot"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/update"
        }
    ]
}
```

下列政策使用萬用字元，允許接收者接收字commands/首下任何主題的直接訊息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/*"
        }
    ]
}
```

> **Note**
>
> 接收者必須先建立 的 AWS IoT Core MQTT 連線，才能接收直接訊息。離線裝置的直接訊息不會排入佇列。

將直接訊息傳送至特定主題上任何用戶端的政策

下列政策允許寄件者將直接訊息傳送至任何用戶端，但僅限於符合commands/*字首的主題。這對於需要連接任何裝置的機群管理服務非常有用，但僅適用於特定的命令主題。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/*",
            "Condition": {
                "StringLike": {
                    "iot:Topic": "commands/*"
                }
            }
        }
    ]
}
```

> **Note**
>
> `iot:Topic` 條件索引鍵支援與 `StringLike`條件運算子進行萬用字元比對。

將直接訊息傳送至任何主題上任何用戶端的政策

下列政策允許寄件者在任何主題上將直接訊息傳送至任何用戶端。這適用於後端服務需要不受限制存取的管理或機群管理使用案例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/*",
            "Condition": {
                "StringLike": {
                    "iot:Topic": "*"
                }
            }
        }
    ]
}
```

使用 Amazon Cognito 身分授權

有兩種類型的 Amazon Cognito 身分：未驗證和已驗證。如果您的應用程式支援未驗證 Amazon Cognito 身分，則不會執行身分驗證，因此您不知道使用者是誰。

Unauthenticated Identities：(未驗證的身分) 對於未驗證的 Amazon Cognito 身分，您可以將 IAM 角色連接至未驗證的身分集區來授予許可。建議只授予您想要提供給未知使用者使用之資源的存取權。

> **Important**
>
> 對於未驗證的 Amazon Cognito 使用者連線到 AWS IoT Core，我們建議您授予 IAM 政策中非常有限資源的存取權。

Authenticated Identities (已驗證的身分)：對於已驗證的 Amazon Cognito 身分，您需要在兩個位置指定許可：

- 將 IAM 政策連接到已驗證的 Amazon Cognito 身分集區，
- 將 AWS IoT Core 政策連接至 Amazon Cognito 身分 （已驗證的使用者）。

未驗證和驗證連線至 的 Amazon Cognito 使用者的政策範例 AWS IoT Core

下列範例顯示在 IAM 政策與 Amazon Cognito 身分 IoT 政策中的許可。已驗證身分的使用者希望發佈到裝置特定主題 (例如 device/DEVICE_ID/status)。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/Client_ID"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/device/Device_ID/status"
            ]
        }
    ]
}
```

下列範例顯示在 Amazon Cognito 未經驗證角色 IAM 政策中的許可。未驗證身分的使用者希望發佈到不需驗證身分的非裝置特定主題。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/non_device_specific_topic"
            ]
        }
    ]
}
```

GitHub 範例

GitHub 上的下列範例 Web 應用程式，展示如何將已驗證使用者的政策附件合併到使用者註冊和身分驗證程序中。

- MQTT 使用 AWS Amplify 和 發佈/訂閱 React Web 應用程式 適用於 JavaScript 的 AWS IoT Device SDK
- MQTT 使用 AWS Amplify、 適用於 JavaScript 的 AWS IoT Device SDK和 Lambda 函數發佈/訂閱 React Web 應用程式

Amplify 是一組工具和服務，可協助您建置與 AWS 服務整合的 Web 和行動應用程式。如需有關 Amplify 的詳細資訊，請參閱 Amplify Framework 說明文件。

這兩個範例都會執行以下步驟。

1. 當使用者註冊帳戶時，應用程式會建立 Amazon Cognito 使用者集區和身分。
2. 當使用者進行身分驗證時，應用程式會建立政策並將其連接至身分。這會為使用者提供發佈和訂閱許可。
3. 使用者可以使用應用程式來發佈和訂閱 MQTT 主題。

第一個範例在身分驗證操作中直接使用 `AttachPolicy` API 操作。下面的範例示範如何在使用 Amplify 和 適用於 JavaScript 的 AWS IoT Device SDK的 React Web 應用程式中實作此 API 呼叫。

```
function attachPolicy(id, policyName) {
    var Iot = new AWS.Iot({region: AWSConfiguration.region, apiVersion:
 AWSConfiguration.apiVersion, endpoint: AWSConfiguration.endpoint});
    var params = {policyName: policyName, target: id};

    console.log("Attach IoT Policy: " + policyName + " with cognito identity id: " +
 id);
    Iot.attachPolicy(params, function(err, data) {
         if (err) {
               if (err.code !== 'ResourceAlreadyExistsException') {
                  console.log(err);
               }
          }
         else  {
            console.log("Successfully attached policy with the identity", data);
         }
```

```
    });
}
```

此程式碼會出現在 AuthDisplay.js 檔案中。

第二個範例會在 Lambda 函數中實作 `AttachPolicy` API 操作。下列範例展示 Lambda 如何使用此 API 呼叫。

```
iot.attachPolicy(params, function(err, data) {
    if (err) {
          if (err.code !== 'ResourceAlreadyExistsException') {
             console.log(err);
             res.json({error: err, url: req.url, body: req.body});
          }
     }
    else  {
       console.log(data);
       res.json({success: 'Create and attach policy call succeed!', url: req.url,
 body: req.body});
    }
 });
```

此程式碼出現在 app.js 檔案的 `iot.GetPolicy` 函數中。

> **Note**
>
> 當您使用透過 Amazon Cognito Identity Pools 取得的 AWS 登入資料呼叫函數時，Lambda 函數中的內容物件會包含 的值`context.cognito_identity_id`。如需更多資訊，請參閱下列內容。
>
> - AWS Lambda Node.js 中的內容物件
> - AWS Lambda Python 中的內容物件
> - AWS Lambda Ruby 中的內容物件
> - AWS Lambda Java 中的內容物件
> - AWS Lambda Go 中的內容物件
> - AWS Lambda C# 中的內容物件

- AWS Lambda PowerShell 中的內容物件

使用 AWS IoT Core 登入資料提供者授權直接呼叫 AWS 服務

裝置可以使用 X.509 憑證， AWS IoT Core 使用 TLS 交互身分驗證通訊協定連線至 。 AWS 其他服務不支援以憑證為基礎的身分驗證，但可以使用 AWS Signature 第 4 版格式的 AWS 登入資料來呼叫。Signature 第 4 版演算法通常要求發起人擁有存取金鑰 ID 和私密存取金鑰。 AWS IoT Core 具有登入資料提供者，可讓您使用內建的 X.509 憑證作為唯一裝置身分來驗證 AWS 請求。這樣就不需要在裝置上儲存存取金鑰 ID 與私密存取金鑰。

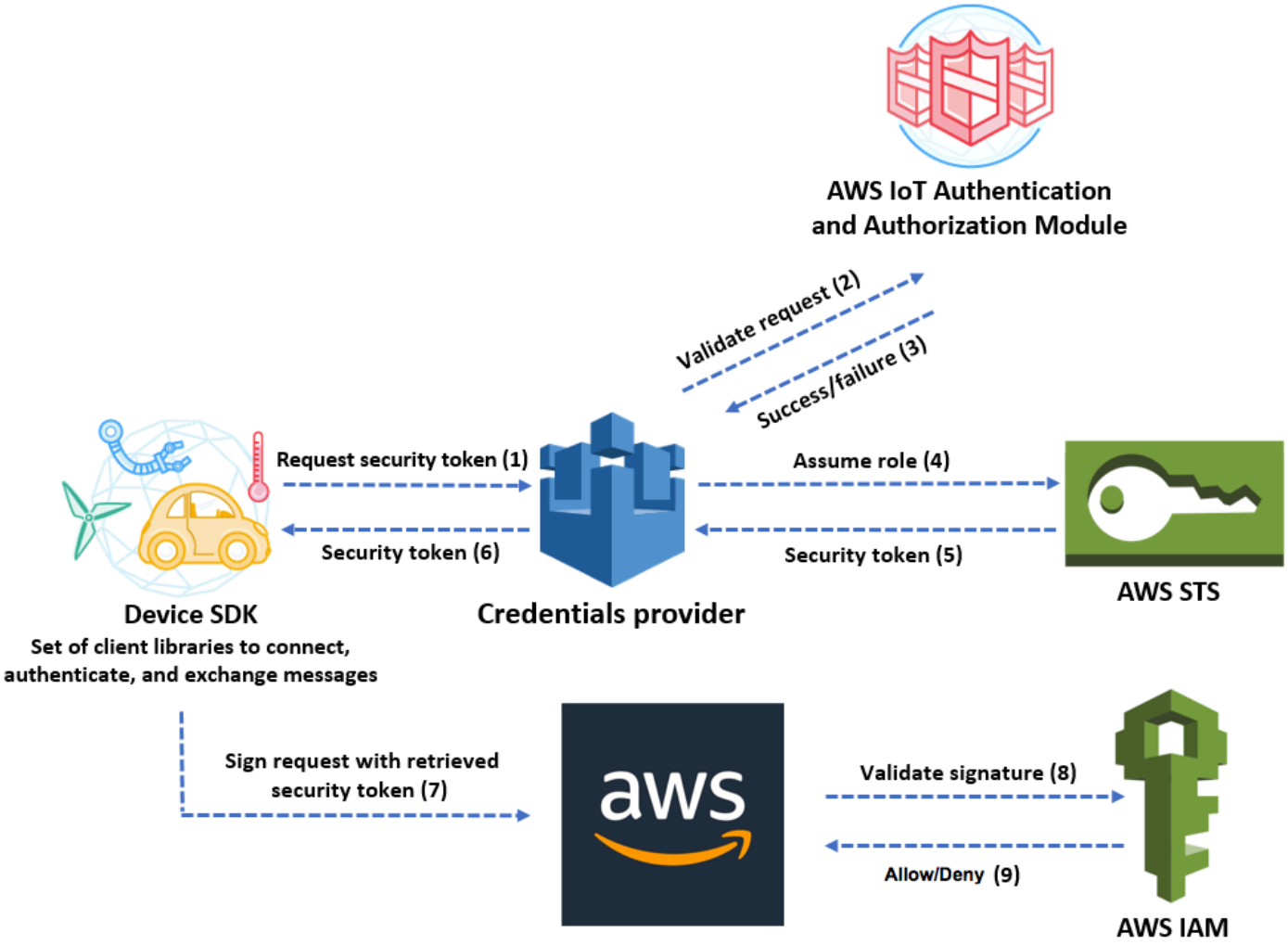
憑證提供者會使用 X.509 憑證來對呼叫者進行身分驗證並發出有限權限的臨時安全性字符。權杖可用來簽署和驗證任何 AWS 請求。這種驗證 AWS 請求的方式需要您建立和設定 AWS Identity and Access Management (IAM) 角色，並將適當的 IAM 政策連接到角色，以便登入資料提供者可以代表您擔任該角色。如需 AWS IoT Core 和 IAM 的詳細資訊，請參閱 的身分和存取管理 AWS IoT。

AWS IoT 需要裝置將伺服器名稱指示 (SNI) 延伸項目傳送至 Transport Layer Security (TLS) 通訊協定，並在 `host_name`欄位中提供完整的端點地址。該 `host_name` 字段必須包含您正在呼叫的端點，並且它必須是：

- 由 `aws iot describe-endpoint --endpoint-type iot:CredentialProvider` 傳回的 `endpointAddress`

沒有正確 `host_name` 值的裝置所嘗試的連線將會失敗。

下圖說明憑證提供者的工作流程。



1. AWS IoT Core 裝置向登入資料提供者提出 HTTPS 請求，以取得安全字符。請求包含用於身分驗證的裝置 X.509 憑證。
2. 登入資料提供者會將請求轉送至 AWS IoT Core 身分驗證和授權模組，以驗證憑證並驗證裝置是否具有請求安全字符的許可。
3. 如果憑證有效且具有請求安全字符的許可，則 AWS IoT Core 身分驗證和授權模組會傳回成功。否則，它會傳送例外到裝置。
4. 在成功驗證憑證後，憑證提供者會叫用 AWS Security Token ServiceAWS STS 來擔任您為該憑證建立的 IAM 角色。
5. AWS STS 會傳回臨時、有限權限的安全字符給登入資料提供者。
6. 憑證提供者傳回安全性字符到裝置。
7. 裝置使用安全字符來簽署 Signature AWS 第 4 版的 AWS 請求。

8. 請求的服務會叫用 IAM 來驗證簽章有效性並根據附加到您為憑證提供者建立的 IAM 角色的存取政策來授權請求。
9. 如果 IAM 成功驗證簽章有效性並授權請求，表示請求成功。否則，IAM 會傳送例外。

下列章節說明如何使用憑證來取得安全性字符。該流程假設您已經註冊裝置且已建立並啟用自己的憑證。

如何使用憑證來取得安全性字符

1. 設定憑證提供者代表您的裝置所擔任的 IAM 角色。將下列信任政策連接至該角色。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"Service": "credentials.iot.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

針對您要呼叫的每個 AWS 服務，將存取政策連接至角色。憑證提供者支援以下政策變數：

- `credentials-iot:ThingName`
- `credentials-iot:ThingTypeName`
- `credentials-iot:AwsCertificateId`

當裝置提供其對 AWS 服務的請求中的物件名稱時，憑證提供者會將 `credentials-iot:ThingName` 與 `credentials-iot:ThingTypeName` 當作內容變數新增到安全性字符。憑證提供者會提供 `credentials-iot:AwsCertificateId` 作為上下文變數，即使裝置在請求中未提供物件名稱。您以 `x-amzn-iot-thingname` HTTP 請求標頭的值來傳遞物件名稱。

這三個變數僅適用於 IAM 政策，不適用於 AWS IoT Core 政策。

2. 請確認執行下一步驟 (建立角色別名) 的使用者擁有傳遞這個新建角色到 AWS IoT Core的權限。下列政策同時將 `iam:GetRole`和 `iam:PassRole`許可提供給 AWS 使用者。`iam:GetRole` 許可讓使用者取得有關您剛建立的角色資訊。`iam:PassRole` 許可允許使用者將角色傳遞給另一個 AWS 服務。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::123456789012:role/your role name"
    }
}
```

3. 建立 AWS IoT Core 角色別名。將直接呼叫 AWS 服務的裝置必須知道連線到 時要使用的角色 ARN AWS IoT Core。硬式編碼的角色 ARN 不是好的解決方案，因為您需要隨角色 ARN 變更而更新裝置。更好的解決方案是使用 `CreateRoleAlias` API 來建立指向該角色 ARN 的角色別名。如果角色 ARN 變更時，您只需更新角色別名。無需在裝置上進行變更。此 API 會使用下列參數：

   `roleAlias`

   必要. 辨識角色別名的任意字串。可作為在角色別名資料模型中的主索引鍵功能。它包含 1-128 字元且必須僅包含英數字元和 =、@ 和 - 符號。可使用大寫和小寫字母字元。角色別名名稱區分大小寫。

   `roleArn`

   必要. 角色別名所指的角色之 ARN。

   `credentialDurationSeconds`

   選用。憑證有效期間 (秒)。最低值為 900 秒 (15 分鐘)。最高值為 43,200 秒 (12 小時)。預設值為 3,600 秒 (1 小時)。

   > **Important**
   >
   > AWS IoT Core 登入資料提供者可以發出生命週期上限為 43，200 秒 (12 小時） 的登入資料。讓憑證有效時間長達 12 小時，有助於藉由快取憑證更長的時間，減少呼叫憑證提供者的次數。

`credentialDurationSeconds` 值必須小於或等於該角色別名參考的 IAM 角色最大工作階段持續時間。如需詳細資訊，請參閱 Identity and Access Management 使用者指南中的 AWS 修改角色最長工作階段持續時間 (AWS API)。

如需此 API 的詳細資訊，請參閱 CreateRoleAlias。

4. 將政策連接至裝置憑證。連接至本裝置憑證的政策，必須授予裝置承擔該角色的許可。您需要授予角色別名 `iot:AssumeRoleWithCertificate` 動作的許可，如下列範例。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/your role
 alias"
        }
    ]
}
```

5. 對憑證提供者發出 HTTPS 請求，以取得安全性字符。請提供下列資訊：

   - 憑證：因為這是一個對 TLS 共同身分驗證的 HTTP 請求，您提出請求時需提供憑證與私有金鑰給用戶端。使用您在註冊憑證時使用的相同憑證和私有金鑰 AWS IoT Core。

     為了確保您的裝置正在與 AWS IoT Core （而不是模擬它的服務）通訊，請參閱伺服器身分驗證，遵循連結下載適當的 CA 憑證，然後將它們複製到您的裝置。
   - RoleAlias：您為憑證提供者所建立的角色別名的名稱。角色別名名稱區分大小寫，且必須符合在其中建立的角色別名 AWS IoT Core。
   - ThingName：您在註冊物件時建立的 AWS IoT Core 物件名稱。這會以 `x-amzn-iot-thingname` HTTP 標頭的值來傳遞。只有在您使用物件屬性做為 AWS IoT Core 或 IAM 政策中的政策變數時，才需要此值。

     > **Note**
     >
     > 您在 `x-amzn-iot-thingname` 中提供的 ThingName 必須符合指派給憑證的 AWS IoT 物件資源。如果不符合，則會傳回 403 錯誤。

在 中執行下列命令 AWS CLI ，以取得 的登入資料提供者端點 AWS 帳戶。如需此 API 行為的詳細資訊，請參閱 DescribeEndpoint。如需啟用 FIPS 的端點，請參閱 AWS IoT Core - 登入資料 提供者端點。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

以下 JSON 物件為 describe-endpoint 命令的範例輸出。它包含用來要求安全字符的 `endpointAddress`。

```
{
    "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
 region.amazonaws.com"
}
```

使用端點對憑證提供者發出 HTTPS 請求，以傳回安全性字符。下列範例命令使用 `curl`，但您可以使用任何 HTTP 用戶端。

> **Note**
>
> roleAlias 名稱區分大小寫，且必須符合在其中建立的角色別名 AWS IoT。

```
curl --cert your certificate --key your private key -H "x-amzn-iot-thingname: your
 thing name" --cacert AmazonRootCA1.pem https://your endpoint /role-aliases/your
 role alias/credentials
```

此命令會傳回一個包含 `accessKeyId`、`secretAccessKey`、`sessionToken` 以及一個過期情況的安全性字符物件。以下 JSON 物件為 `curl` 命令的範例輸出。

```
   {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access
 key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

然後，您可以使用 `accessKeyId`、 `secretAccessKey`和 `sessionToken`值來簽署對 AWS 服務的請求。如需end-to-end示範，請參閱 AWS 安全部落格上的[如何使用 AWS 登入資料提供者部落格文章，消除裝置中的硬式編碼 AWS IoT 登入資料需求](#)。

利用 IAM 跨帳戶存取

AWS IoT Core 可讓您讓委託人發佈或訂閱委託人 AWS 帳戶 未擁有之 中定義的主題。您可以建立 IAM 政策和 IAM 角色，藉此設定跨帳戶存取，並將政策連接至該角色。

首先，如[建立 IAM 政策](#)中所述建立客戶受管的 IAM 政策，如同為 AWS 帳戶中的其他使用者和憑證所做的一樣。

對於在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會授予許可，讓裝置 AWS IoT Core 使用符合裝置物件名稱的用戶端 ID 連線至 ，並發佈至 ，`my/topic/`*`thing-name`* 其中 *`thing-name`* 是裝置的物件名稱：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
```

```
}
```

對於未在 AWS IoT Core 登錄檔中註冊的裝置，下列政策會將許可授予裝置，以使用您帳戶 (123456789012) AWS IoT Core 登錄檔中`client1`註冊的物件名稱來連線至 ， AWS IoT Core 並發佈至名稱字首為 的用戶端 ID 特定主題`my/topic/`：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

接著，依照建立角色以將許可委派給 IAM 使用者中的步驟。輸入您欲共用存取的 AWS 帳戶 之帳戶 ID。最後一個步驟是，將政策連接至您剛建立的角色。若稍後您需要修改授予存取的 AWS 帳戶 ID，您可以使用下列信任政策的格式：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws:iam::567890123456:user/MyUser"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

中的資料保護 AWS IoT Core

將 AWS 共同責任模式應用於 AWS IoT Core中的資料保護。如此模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。 如需歐洲資料保護的詳細資訊，請參閱一般資料保護規則 (GDPR) 中心。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 AWS IoT 或使用主控台、API AWS CLI或其他 AWS 服務 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

如需關於資料保護的詳細資訊，請參閱 AWS 安全部落格上的 AWS 共同責任模型和歐盟《一般資料保護規範》(GDPR) 部落格文章。

AWS IoT 裝置會收集資料、對該資料執行一些操作，然後將該資料傳送至另一個 Web 服務。您可以選擇在裝置上短時間存放一些資料。您有責任對靜態資料提供資料保護。當您的裝置傳送資料至 時 AWS IoT，會透過本節稍後討論的 TLS 連線執行此操作。 AWS IoT 裝置可以將資料傳送至任何 AWS 服務。如需每個服務的資料安全性詳細資訊，請參閱該服務的文件。 AWS IoT 可設定為將日誌寫入 CloudWatch Logs 並記錄 AWS IoT API 呼叫 AWS CloudTrail。如需這些服務資料安全性的詳細資訊，請參閱 Amazon CloudWatch 的身分驗證和存取控制，以及使用 AWS KMS 受管金鑰加密 CloudTrail 日誌檔案。

中的資料加密 AWS IoT

根據預設，傳輸中和靜態的所有 AWS IoT 資料都會加密。傳輸中的資料會使用 TLS 加密，而靜態資料會使用 AWS 擁有的金鑰加密。 AWS IoT 支援來自 AWS Key Management Service AWS KMS keys () 的客戶受管 (KMS 金鑰) AWS KMS。不過，Device Advisor 和 AWS IoT Wireless 僅使用 AWS 擁有的金鑰 來加密客戶資料。

中的傳輸安全性 AWS IoT Core

TLS (Transport Layer Security) 是一種加密通訊協定，專為透過電腦網路進行安全通訊而設計。AWS IoT Core Device Gateway 要求客戶在傳輸期間加密所有通訊，方法是使用 TLS 從裝置連線至 Gateway。TLS 可針對 AWS IoT Core支援的應用程式通訊協定 (MQTT、HTTP 和 WebSocket)，達到其機密性。多種程式設計語言與作業系統提供 TLS 支援。內的資料由特定 AWS 服務 AWS 加密。如需其他服務資料加密的詳細資訊 AWS ，請參閱該服務的安全文件。

目錄

- TLS 通訊協定
- 安全政策
- 中傳輸安全的重要注意事項 AWS IoT Core
- LoRaWAN 無線裝置的傳輸安全性

TLS 通訊協定

AWS IoT Core 支援下列版本的 TLS 通訊協定：

- TLS 1.3
- TLS 1.2

使用 AWS IoT Core，您可以在網域組態中設定 TLS 設定（適用於 TLS 1.2 和 TLS 1.3)。如需詳細資訊，請參閱???。

安全政策

安全政策是 TLS 通訊協定及其密碼的組合，用來決定在用戶端與伺服器之間進行 TLS 交涉期間所支援的通訊協定和密碼。您可根據需要將裝置設定為使用預先定義的安全政策。請注意， AWS IoT Core 不支援自訂安全政策。

連線裝置時，您可以選擇其中一個預先定義的安全政策 AWS IoT Core。中最新預先定義安全政策的名稱 AWS IoT Core 包括根據發行月份和年份的版本資訊。非AWS GovCloud (US) 區域的預設預先定義安全政策為`IoTSecurityPolicy_TLS13_1_2_2022_01` AWS GovCloud (US) 區域`IoTSecurityPolicy_TLS13_1_2_2022_10`和 。若要指定安全政策，您可以使用 AWS IoT 主控台或 AWS CLI。如需詳細資訊，請參閱???。

下表說明 AWS IoT Core 支援的最新預先定義安全政策。已從標頭列的政策名稱移除`IotSecurityPolicy_`，使其相符。

安全政策	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2_2022_01*
TCP 連接埠	443/8443/8883	443/8443/8883	443/8443/8883	443	8443/888	443	8443/888	443/8443/8883
TLS 通訊協定								
TLS 1.2		✓	✓	✓	✓	✓	✓	✓
TLS 1.3	✓	✓						✓
TLS 加密								
TLS_AES_128_GCM_A256	✓	✓						✓
TLS_AES_256_GCM_A384	✓	✓						✓

安全政策	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2_2022_01*
TLS_CHACHA20_POLY1305_SHA256	✓	✓						
ECDHE-RSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA384		✓	✓	✓	✓	✓	✓	✓

安全政策	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
ECDHE-RSA-AES256-SHA		✓	✓	✓	✓	✓	✓	✓
AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	
AES128-SHA256		✓	✓	✓		✓	✓	
AES128-SHA		✓	✓	✓	✓	✓	✓	✓
AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	
AES256-SHA256		✓	✓	✓	✓	✓	✓	
AES256-SHA		✓	✓	✓	✓	✓	✓	✓
DHE-RSA-AES256-SHA						✓	✓	

安全政策	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
ECDHE-ECDSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	✓

安全政策	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2_2022_01*
ECDHE-ECDSA-AES256-SHA384		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA		✓	✓	✓	✓	✓	✓	✓

> **Note**
>
> TLS12_1_0_2016_01 僅適用於下列項目 AWS 區域：ap-east-1、ap-northeast-2、ap-south-1、ap-southeast-2、ca-central-1、cn-north-1、cn-northwest-1、eu-north-1、eu-west-2、eu-west-3、me-south-1、sa-east-1、us-east-2、us-west-1。
> TLS12_1_0_2015_01 僅適用於下列項目 AWS 區域：ap-northeast-1、ap-southeast-1、eu-central-1、eu-west-1、us-east-1、us-west-2。
> TLS13_1_2_2022_01 僅適用於 AWS GovCloud (US) 區域。

中傳輸安全的重要注意事項 AWS IoT Core

對於 AWS IoT Core 使用 MQTT 連線至 的裝置，TLS 會加密裝置與代理程式之間的連線， AWS IoT Core 並使用 TLS 用戶端身分驗證來識別裝置。如需更多資訊，請參閱用戶端身分驗證。對於 AWS IoT Core 使用 HTTP 連線至 的裝置，TLS 會加密裝置與代理程式之間的連線，並將身分驗證委派給 AWS Signature 第 4 版。如需詳細資訊，請參閱《AWS 一般參考》中的使用 Signature 第 4 版簽署請求。

當您將裝置連線到 時 AWS IoT Core，不需要傳送伺服器名稱指示 (SNI) 延伸模組，但強烈建議您這麼做。若要使用多帳戶註冊、自訂網域、VPC 端點和已設定的 TLS 政策等功能，您必須使用 SNI 延伸，

並在 host_name欄位中提供完整的端點地址。該 host_name 欄位必須包含您正在呼叫的端點。該端點必須是下列其中一個：

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS` 返回的 endpointAddress
- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` 返回的 domainName

host_name 值不正確或無效的裝置嘗試的連線將會失敗。 AWS IoT Core 會針對自訂身分驗證的身分驗證類型，將失敗記錄到 CloudWatch。

AWS IoT Core 不支援 SessionTicket TLS 延伸。

LoRaWAN 無線裝置的傳輸安全性

LoRaWAN 裝置遵循 LoRaWAN ™ SECURITY: A White Paper Prepared for the LoRa Alliance™ by Gemalto, Actility, and Semtech 中所述的安全實務。

如需使用 LoRaWAN 裝置傳輸安全性的詳細資訊，請參閱 LoRaWAN 資料和傳輸安全性。

中的資料加密 AWS IoT

資料保護是指在傳輸中（往返 時 AWS IoT Core) 和靜態（存放於裝置或其他 AWS 服務時） 時保護資料。傳送到 的所有資料 AWS IoT Core 都會使用 MQTT、HTTPS 和 WebSocket 通訊協定透過 TLS 連線傳送，因此預設在傳輸時安全。 會從裝置 AWS IoT Core 收集資料，然後將其傳送至其他 AWS 服務以進行進一步處理。如需有關其他 AWS 服務上資料加密的詳細資訊，請參閱該服務的安全性文件。如需詳細資訊，請參閱靜態資料加密。

FreeRTOS 提供 PKCS#11 程式庫，可抽象化金鑰儲存、存取密碼編譯物件和管理工作階段。您有責任使用此程式庫來加密裝置上的靜態資料。如需詳細資訊，請參閱 FreeRTOS 公有金鑰加密標準 (PKCS) #11 程式庫。

中的靜態資料加密 AWS IoT Core

根據預設，所有靜態 AWS IoT Core 資料都會使用 AWS 擁有的金鑰進行加密。 AWS IoT Core 也支援來自 AWS Key Management Service (AWS KMS) 的對稱客戶受管金鑰。使用客戶受管金鑰，您可以建立、擁有和管理 AWS 帳戶中的 AWS KMS 金鑰。 AWS IoT Core 將使用 KMS 金鑰來加密靜態資料。您可以完全控制這些 KMS 金鑰，包括建立和維護其金鑰政策。您也可以為存取 的角色設定 IAM 政策， AWS KMS 以控制這些金鑰的許可。

AWS 擁有的金鑰

AWS 擁有的金鑰是 AWS 服務擁有和管理的 KMS 金鑰集合，可用於多個 AWS 帳戶。 AWS 服務可以使用 AWS 擁有的金鑰來保護您的資料。根據預設， 會使用 AWS 擁有的金鑰 AWS IoT Core 加密靜態資料。這些金鑰由 服務管理。您無法檢視、管理或使用 AWS 擁有的金鑰。不過，您不需要採取任何動作來保護這些金鑰。

如需 AWS 擁有金鑰的詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的 AWS 擁有金鑰。

客戶自管金鑰

客戶受管金鑰是您帳戶中建立、擁有和管理 AWS 的 KMS 金鑰。您可以完全控制這些 AWS KMS 金鑰，包括建立和維護其金鑰政策。您也可以為存取 的角色設定 IAM 政策， AWS KMS 以控制這些金鑰的許可。您可以設定 AWS IoT Core 使用客戶受管 KMS 金鑰來加密您的資料。

如需客戶受管金鑰的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的客戶受管金鑰。

若要選擇加入客戶受管金鑰 AWS IoT Core，請遵循下列步驟：

主題

- 步驟 1：建立客戶受管金鑰
- 步驟 2：建立 IAM 角色以授予使用 KMS 金鑰的 AWS IoT Core 許可
- 步驟 3：選擇加入 中的客戶受管金鑰 AWS IoT Core
- 步驟 4： AWS IoT Core 控制平面操作所需的其他許可
- 步驟 5：管理金鑰
- 步驟 6：監控金鑰運作狀態

步驟 1：建立客戶受管金鑰

您可以使用 AWS KMS 主控台或 CLI AWS KMS 命令來建立對稱客戶受管金鑰。keySpec 必須是 SYMMETRIC_DEFAULT，而 keyUsage必須是 ENCRYPT_DECRYPT。

> **Note**
>
> AWS IoT Core 僅支援具有客戶受管 AWS KMS 金鑰SYMMETRIC_DEFAULT之金鑰規格和ENCRYPT_DECRYPT金鑰用量的金鑰。

以下是建立 KMS 金鑰的範例 AWS CLI 命令，可用於 AWS IoT Core 客戶受管金鑰。

```
aws kms create-key --key-spec SYMMETRIC_DEFAULT --key-usage ENCRYPT_DECRYPT --region
 us-west-2
```

以下是 命令的範例輸出。

```
{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-09-19T11:45:23.982000-07:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
```

如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的建立對稱客戶受管金鑰。

金鑰政策

建立客戶受管金鑰時，您可以指定金鑰政策。金鑰政策會控制客戶受管金鑰的存取權限。每個客戶受管金鑰都必須只有一個金鑰政策，其中包含決定誰可以使用金鑰及其使用方式的陳述式。如需詳細資訊，請參閱《 AWS Key Management Service 開發人員指南》中的金鑰政策。

AWS IoT Core 會在您的帳戶中使用 IAM 角色來存取您的客戶受管金鑰。如果您使用的是自訂金鑰政策，請確定在此金鑰上建立的 IAM 角色具有下列許可：

- `kms:DescribeKey`

- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

步驟 2：建立 IAM 角色以授予使用 KMS 金鑰的 AWS IoT Core 許可

若要 AWS IoT Core 讓 使用您建立的 KMS 金鑰來加密靜態資料，您也需要在帳戶中建立 IAM 角色，其 AWS IoT Core 可擔任 以存取 KMS 金鑰。

角色必須具有下列信任政策，才能允許 AWS IoT Core 擔任該角色。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "iot.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iot:us-west-2:111122223333:*"
            }
        }
    }
}
```

確保連接至 IAM 角色的 IAM 政策具有 KMS 金鑰的下列許可：

- kms:DescribeKey
- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo

- kms:ReEncryptFrom

以下是具有客戶受管金鑰所需許可的 IAM 政策範例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIoTToAccessKMSResource",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:ReEncryptTo",
                "kms:ReEncryptFrom",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": [
                "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            ],
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws-crypto-ec:vendor": "iot.amazonaws.com"
                }
            }
        }
    ]
}
```

如需詳細資訊，請參閱《 AWS Identity and Access Management 使用者指南》中的建立角色以將許可委派給 IAM 使用者。

步驟 3：選擇加入 中的客戶受管金鑰 AWS IoT Core

完成所有先前的步驟後，請執行 update-encryption-configuration CLI 命令，以選擇在 中使用客戶受管金鑰 AWS IoT Core。當您選擇使用客戶受管金鑰時， AWS 帳戶中的所有 AWS IoT Core 資源都會使用指定的 AWS KMS 金鑰加密。

1. 若要選擇使用 AWS IoT Core 中的客戶受管金鑰 AWS CLI，請執行 CLI update-encryption-configuration 命令。

```
aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
 \
--kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
--kms-key-arn "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab" --region us-west-2
```

2. 若要 AWS IoT Core 使用 驗證 中的客戶受管金鑰 AWS CLI，請執行 CLI `describe-encryption-configuration` 命令：

```
aws iot describe-encryption-configuration --region us-west-2
```

如果您已在 中啟用客戶受管金鑰 AWS IoT Core，輸出可能如下所示：

```
{
    "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "configurationDetails": {
        "configurationStatus": "HEALTHY"
    },
    "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
}
```

`lastModifiedDate` 欄位指出上次更新加密組態的日期。

如果您尚未啟用客戶受管金鑰，輸出可能如下所示：

```
{
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
}
```

步驟 4： AWS IoT Core 控制平面操作所需的其他許可

在您選擇加入客戶受管金鑰之後，屬於您 AWS 帳戶的所有 AWS IoT Core 資源都會使用提供的 KMS 金鑰加密。除了 AWS IoT Core 資源上特定操作所需的`kms:Decrypt`許可之外，所有控制平面操作現在還要求發起人擁有 KMS 金鑰的許可。如果發起人沒有`kms:Decrypt`許可，而且他們進行需要加密或解密資料的 API 呼叫 （例如 `GetPolicy`），他們將收到 `UnauthorizedException`。

例如，當您呼叫 時GetPolicy，您需要客戶受管 KMS 金鑰的 iot:GetPolicy和 kms:Decrypt許可，才能成功呼叫 API。

> **Note**
>
> 更新 IAM 使用者或角色以授予加密組態所用金鑰的 AWS KMS 許可時，請確定 KMS 金鑰政策也授予個別 IAM 使用者或角色所需的許可。

AWS KMS 的 許可 **UpdateEncryptionConfiguration**

UpdateEncryptionConfiguration API 呼叫需要 KMS 金鑰的下列 AWS KMS 許可，才能選擇加入客戶受管金鑰或修改金鑰組態：

- kms:DescribeKey
- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

AWS KMS 所有其他控制平面 APIs 許可

啟用客戶受管金鑰時，大多數控制平面 APIs 都需要kms:Decrypt許可。不過，某些 APIs不需要這些額外的許可：

不需要 AWS KMS 許可APIs

List* 和 Delete* APIs 不屬於此儲存貯體。客戶可以隨時叫用任何 List*或 Delete* 控制平面 API，即使發起人沒有kms:Decrypt許可，這些 API 呼叫也會成功。即使您的客戶受管金鑰運作狀態不佳，List*且 API 不會進行任何解密，這些 Delete* API 呼叫仍會成功。 APIs

- List* APIs – 所有列出操作 （ 例如，、ListThingsListPolicies、ListCertificates)
- Delete* APIs – 所有刪除操作 （ 例如，、DeleteThingDeletePolicy、DeleteCertificate)

步驟 5：管理金鑰

AWS IoT Core 會定期檢查客戶受管金鑰組態，以確保加密和解密操作不會受到影響。這些運作狀態檢查每分鐘執行一次 AWS IoT Core，並驗證 存取和使用 AWS KMS 金鑰和相關聯 IAM 角色的能力，以進行加密和解密操作。

HEALTHY (狀況良好)

AWS IoT Core 可以透過指定的 IAM 角色成功存取 AWS KMS 金鑰，並執行加密/解密操作。所有元件都能正常運作。

UNHEALTHY (狀況不良)

AWS IoT Core 無法存取或使用 AWS KMS 金鑰。這可防止新的加密操作，並可能影響服務功能。`errorCode` 欄位指出問題是與金鑰還是 IAM 角色有關。

可能影響金鑰運作狀態的客戶動作

數個客戶動作可能會導致金鑰運作狀態從 變更為 `HEALTHY` `UNHEALTHY`：

金鑰相關動作

- 刪除 AWS KMS 金鑰 – 當您排程刪除金鑰時，金鑰處於 `Pending deletion` 狀態且無法使用
- 停用 AWS KMS 金鑰 – 當您停用 KMS 金鑰時，它無法再用於加密/解密操作
- 排程刪除金鑰 – 刪除完成時，金鑰變得無法使用
- 修改金鑰政策 – 移除 AWS IoT Core 存取所需的許可
- 變更金鑰使用許可 – 限制必要的 AWS KMS 動作

IAM 角色相關動作

- 刪除 IAM 角色 – AWS IoT Core 無法擔任該角色以存取金鑰
- 修改角色許可 – 從角色政策移除必要的 AWS KMS 許可
- 變更信任政策 – 防止 AWS IoT Core 服務擔任該角色
- 新增限制條件 – AWS IoT Core 防止 使用該角色的條件

帳戶層級動作

- 跨帳戶金鑰存取變更 – 修改不同帳戶中金鑰的許可
- 服務控制政策 SCPs) – 限制 AWS KMS 存取的組織層級政策
- 帳戶層級 IAM 政策 – 覆寫金鑰存取或與金鑰存取衝突的政策

> **Important**
>
> 對 使用的 AWS KMS 金鑰、IAM 角色或政策的任何變更都 AWS IoT Core 應先在開發環境中進行測試。在進行任何變更後密切監控金鑰運作狀態，以確保 AWS IoT Core 功能不受影響。

更新加密組態

在 中更新您的加密組態 AWS IoT Core ，以從一個客戶受管金鑰變更為另一個金鑰，或在 AWS 擁有的金鑰和客戶受管金鑰之間進行變更。

若要將組態變更為不同的客戶受管金鑰：

1. 依照中的步驟建立新的客戶受管金鑰the section called “步驟 1：建立客戶受管金鑰”。
2. 更新您的 IAM 角色政策，以在更新期間同時包含舊金鑰和新金鑰的許可。
3. 更新您的加密組態以使用新的金鑰：

```
aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
 \
--kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
--kms-key-arn "arn:aws:kms:us-west-2:111122223333:key/new-key-id"
```

若要將組態從客戶受管金鑰變更回 AWS 擁有的金鑰：

```
aws iot update-encryption-configuration --encryption-type "AWS_OWNED_KMS_KEY"
```

> **Note**
>
> 更新新客戶受管金鑰的加密組態時，請確保舊金鑰和新金鑰都保持可存取，以便操作成功。

常見的失敗案例和影響

下表說明刪除或停用金鑰時的常見失敗案例：

案例	立即影響	長期後果
金鑰已停用	所有新的加密/解密操作都會立即失敗	服務中斷，直到金鑰重新啟用或取代為止
排程刪除的金鑰	金鑰狀態變更為待刪除，且所有加密/解密操作都會失敗	刪除完成時自動服務失敗
金鑰已永久刪除	所有操作的立即和永久故障	永久資料遺失和無法復原加密的資料
金鑰政策修改不正確	AWS IoT Core 失去金鑰的存取許可	服務失敗，直到政策修正為止
已刪除 IAM 角色	AWS IoT Core 無法擔任存取金鑰的角色	完成加密服務失敗
IAM 角色修改不正確	AWS IoT Core 無法擔任角色或使用角色來存取金鑰	修正 IAM 角色之前的服務失敗

預防和最佳實務

若要防止意外刪除或停用金鑰，並將服務失敗的風險降至最低：

實作金鑰生命週期政策

建立明確的金鑰建立、輪換和淘汰程序。記錄哪些金鑰用於 AWS IoT Core 資源和維護作用中金鑰的清查。

使用 IAM 政策來限制金鑰刪除

建立 IAM 政策，防止未經授權的使用者刪除或停用關鍵加密金鑰。使用條件要求金鑰刪除操作的額外核准。

啟用 CloudTrail 記錄

透過 CloudTrail 監控所有 AWS KMS 金鑰操作，以偵測未經授權或意外的金鑰管理活動。設定金鑰刪除、停用或政策變更的提醒。

測試金鑰取代程序

在非生產環境中定期測試您的金鑰取代程序，以確保您可以快速從與金鑰相關的故障中復原。

維護金鑰備份

雖然您無法匯出 AWS KMS 金鑰材料，但請保留金鑰 ARNs、政策和相關 AWS IoT Core 組態的詳細記錄，以便在需要時快速取代金鑰。

監控金鑰運作狀態

持續監控CMK.Health指標，並設定金鑰運作狀態變更的自動提醒。實作自動化回應，以快速解決與金鑰相關的問題。

> **Important**
>
> 在生產環境中實作金鑰更新程序之前，請務必在開發環境中測試這些更新程序。制定有文件記錄的轉返計劃，並確保在緊急情況下可以快速執行金鑰替換程序。

步驟 6：監控金鑰運作狀態

作為定期檢查 AWS IoT Core 執行的一部分，會發出 CloudWatch 指標和日誌，以提供客戶受管金鑰組態運作狀態的可見性

AWS IoT Core 每分鐘至少向 CloudWatch 發出一次CMK.Health指標。此指標提供 用來 AWS IoT Core 加密和解密資料之客戶受管金鑰的運作狀態相關資訊。

CMK.Health 指標可以有下列值：

- 值為 1： AWS IoT Core 能夠成功地使用加密金鑰來加密和解密您的資料。
- 值為 0： AWS IoT Core 無法使用加密金鑰來加密和解密您的資料。

AWS IoT Core 當加密金鑰的運作狀態變更時， 也會發出 AWS IoT V2 日誌。這些日誌提供運作狀態更新的其他詳細資訊。若要檢視這些日誌，您必須啟用 AWS IoT V2 日誌。HEALTHY 日誌會在 INFO 層級發出，而UNHEALTHY日誌會在 ERROR層級發出。如需日誌層級的詳細資訊，請參閱日誌層級。

下列範例是 發出的 CloudWatch 日誌項目 AWS IoT Core ，用於指示客戶受管金鑰的運作狀態更新。

若要有效監控和回應關鍵運作狀態變更：

1. 設定 指標的 CloudWatch 警示： CMK.Health

```
aws cloudwatch put-metric-alarm --region us-west-2 \
  --alarm-name "IoTCore-CMK-Health-Alert" \
```

```
  --alarm-description "Alert when IoT Core CMK health is unhealthy" \
  --metric-name "CMK.Health" \
  --namespace "AWS/IoT" \
  --statistic "Minimum" \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator "LessThanThreshold" \
  --alarm-actions "arn:aws:sns:us-west-2:111122223333:iot-alerts"
```

2. Enable AWS IoT V2 記錄功能可擷取詳細的運作狀態變更事件，其中包含錯誤碼和訊息。
3. 檢查組態狀態以進行故障診斷：

```
aws iot describe-encryption-configuration --region us-west-2
```

4. 檢查 errorCode 欄位以調查 UNHEALTHY 狀態：
   - KMS_KEY_VALIDATION_ERROR – AWS KMS 金鑰的問題（停用、刪除或政策問題）
   - ROLE_VALIDATION_ERROR – IAM 角色的問題（已刪除、政策問題或信任問題）

從不良到健康

當加密金鑰的狀態從 更新為 UNHEALTHY 時HEALTHY， AWS IoT Core 會發出下列格式的 AWS IoT V2 日誌訊息。

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "SUCCESS",
    "cmkStatus": "HEALTHY",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

從 HEALTHY 到 UNHEALTHY

當加密金鑰的狀態從 更新為 HEALTHY 時UNHEALTHY， AWS IoT Core 會發出下列格式的 AWS IoT V2 日誌訊息。

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "FAILURE",
    "cmkStatus": "UNHEALTHY",
    "errorCode": "KMS_KEY_VALIDATION_ERROR / ROLE_VALIDATION_ERROR",
    "errorMessage": "Error message on why there was a failure",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

Warning

當金鑰運作狀態變成 時UNHEALTHY， AWS IoT Core 操作會立即失敗。如果發生這種情況，請檢閱您的金鑰組態、IAM 角色許可和政策。監控 CMK.Health 指標是否有狀態變更。如果在檢閱您的組態後操作仍持續失敗，請聯絡您的客戶經理或AWS 支援中心以取得其他協助。

監控分支金鑰輪換運作狀態

AWS IoT Core 使用階層式金鑰結構來加密您的資料。此階層的頂端是您的 AWS KMS 金鑰 （您設定的客戶受管金鑰）。接下來， AWS IoT Core 會維護分支金鑰 – 從金鑰衍生的內部 AWS KMS 金鑰。AWS IoT Core 然後， 使用這些分支金鑰來產生唯一的包裝金鑰，以執行資料的實際加密和解密。這種分層方法會限制任何單一金鑰的範圍，並遵循 AWS 加密最佳實務。 AWS IoT Core 會管理分支金鑰做為內部實作詳細資訊，與直接 AWS KMS 管理的 AWS KMS 金鑰輪換分開。如需詳細資訊，請參閱輪換作用中的分支金鑰。

AWS IoT Core 每年輪換分支金鑰做為安全最佳實務。輪換會建立新的作用中分支金鑰版本，同時保留先前的版本以解密現有的資料。此輪換需要與客戶受管金鑰組態相關聯的 IAM 角色kms:Encrypt許可。 每分鐘向 CloudWatch AWS IoT Core 發出一次CMK.BranchKeyRotationHealth指標，以指出您的 AWS KMS 存取角色是否具有輪換所需的許可。

CMK.BranchKeyRotationHealth 指標可以有下列值：

- 值為 1： AWS KMS 存取角色具有必要的kms:Encrypt許可，且分支金鑰輪換可以正常進行。

- 值為 `0`： AWS KMS 存取角色缺少`kms:Encrypt`許可，無法執行分支金鑰輪換。您現有的解密操作會繼續正常運作，但在還原許可之前 AWS IoT Core ，將無法輪換分支金鑰。將 `kms:Encrypt`新增至您的 AWS KMS 存取角色以重新啟用輪換。如需所需許可的完整清單，請參閱金鑰政策。

> **Note**
>
> 值 `0` 不表示目前 AWS IoT Core 操作中斷。您的資料會保持加密並可存取。不過，您應該立即更新 AWS KMS 存取角色許可，以確保分支金鑰可以按排程輪換。

若要在分支金鑰輪換無法繼續時收到通知，請設定 `CMK.BranchKeyRotationHealth` 指標的 CloudWatch 警示：

```
aws cloudwatch put-metric-alarm --region us-west-2 \
  --alarm-name "IoTCore-CMK-BranchKeyRotationHealth-Alert" \
  --alarm-description "Alert when IoT Core branch key rotation health is degraded" \
  --metric-name "CMK.BranchKeyRotationHealth" \
  --namespace "AWS/IoT" \
  --statistic "Minimum" \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator "LessThanThreshold" \
  --alarm-actions "arn:aws:sns:us-west-2:111122223333:iot-alerts"
```

AWS CloudTrail 事件

您也可以監控 KMS 金鑰的 AWS IoT Core使用情況，以加密解密操作。 AWS IoT Core 會在您的 KMS 金鑰上建立 `DescribeKey`、`ReEncrypt`、 和 `DecryptGenerateDataKeyWithoutPlaintext`操作，以加密/解密屬於您靜態存放 AWS 帳戶的資料。

`DescribeKey`、、 `DecryptReEncrypt`和 有 CloudTrail 事件`GenerateDataKeyWithoutPlaintext`。這些事件會監控 呼叫 AWS KMS 的操作 AWS IoT Core ，以存取客戶受管金鑰加密的資料。

`Decrypt` 範例

```
{
    "eventVersion": "1.09",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "**********************",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "*****"
            },
            "attributes": {
                "creationDate": "2024-09-16T20:23:39Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "iot.amazonaws.com"
    },
    "eventTime": "2024-09-16T20:32:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "iot.amazonaws.com",
    "userAgent": "iot.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "kms-arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "aws-crypto-ec:vendor": "iot.amazonaws.com",
            "branch-key-id": "111122223333",
            "type": "branch:ACTIVE"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "1afb6d98-8388-455d-8b48-e62c9e0cf7f4",
    "eventID": "b59a5f16-0d98-46d8-a590-0e040a48b39b",
    "readOnly": true,
    "resources": [
        {
```

```
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

的身分和存取管理 AWS IoT

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證 （ 登入 ） 和授權 （ 具有許可 ） 來使用 AWS IoT 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用 IAM 身分來驗證
- 使用政策管理存取權
- AWS IoT 如何使用 IAM
- AWS IoT 身分型政策範例
- AWS 的 受管政策 AWS IoT
- 對 AWS IoT 身分和存取進行故障診斷

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 對 AWS IoT 身分和存取進行故障診斷)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 AWS IoT 如何使用 IAM)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 AWS IoT 身分型政策範例)

使用 IAM 身分來驗證

身分可以是裝置 AWS IoT (X.509) 憑證、Amazon Cognito 身分或 IAM 使用者或群組。本主題僅討論 IAM 身分。如需 AWS IoT 支援的其他身分的詳細資訊，請參閱 用戶端身分驗證。

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務 和 資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色 （主控台） 或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱《Amazon Simple Storage Service 開發人員指南》中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS IoT 如何使用 IAM

在您使用 IAM 管理對 的存取之前 AWS IoT，您應該了解哪些 IAM 功能可與 搭配使用 AWS IoT。若要全面了解 AWS IoT 和其他 AWS 服務如何與 IAM 搭配使用，請參閱《IAM 使用者指南》中的與 IAM AWS 搭配使用的 服務。

主題

- AWS IoT 身分型政策
- AWS IoT 資源型政策
- 以 AWS IoT 標籤為基礎的授權
- AWS IoT IAM 角色

AWS IoT 身分型政策

使用 IAM 身分類型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下會允許或拒絕動作。 AWS IoT 支援特定動作、資源及條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

下表列出 IAM IoT 動作、相關聯的 AWS IoT API，以及動作操作的資源。

政策動作	AWS IoT API	Resources
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** ARN 中 AWS 帳戶 指定的 必須是憑證要傳輸到的帳戶。
iot:AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	無
iot:AttachPolicy	AttachPolicy	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* 或 `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:AttachSecurityProfile	AttachSecurityProfile	`arn:aws:iot:` *region*`:`*account-id* `:securityprofile/` *security-profile-name*

政策動作	AWS IoT API	Resources
		arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:AttachThingPrincipal	AttachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id* **Note** ARN 中 AWS 帳戶 指定的 必須是憑證要傳輸到的帳戶。
iot:CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	無
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
iot:CreateDimension	CreateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*

政策動作	AWS IoT API	Resources
iot:CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateJobTemplate	CreateJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*
iot:CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:CreatePolicyVersion	CreatePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** 這必須是 AWS IoT 政策，而不是 IAM 政策。
iot:CreateRoleAlias	CreateRoleAlias	(參數：roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*

政策動作	AWS IoT API	Resources
iot:CreateSecurityProfile	CreateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:CreateThingGroup	CreateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 適用正在建立的群組和父群組 (若使用)
iot:CreateThingType	CreateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:CreateTopicRule	CreateTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DeleteDimension	DeleteDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-template-id*

政策動作	AWS IoT API	Resources
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DeleteSecurityProfile	DeleteSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeleteThingType	DeleteThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*

政策動作	AWS IoT API	Resources
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (參數：authorizerName) 無
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	無
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	無
iot:DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	無

政策動作	AWS IoT API	Resources
iot:DescribeJobTemplate	DescribeJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-template-id*
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	無
iot:DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 或 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DetachSecurityProfile	DetachSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:GetIndexingConfiguration	GetIndexingConfiguration	無
iot:GetJobDocument	GetJobDocument	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 或 arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
iot:ListAuthorizers	ListAuthorizers	無
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	無
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	無
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	無
iot:ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* 若使用 thingGroupName 參數
iot:ListJobTemplates	ListJobs	無
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	*

政策動作	AWS IoT API	Resources
iot:ListPolicyVersions	ListPolicyVersions	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:ListPrincipalPolicies	ListPrincipalPolicies	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListPrincipalThings	ListPrincipalThings	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListRoleAliases	ListRoleAliases	無
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:ListThingGroups	ListThingGroups	無
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	無
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	無
iot:ListThingTypes	ListThingTypes	*

政策動作	AWS IoT API	Resources
iot:ListThings	ListThings	*
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:ListTopicRules	ListTopicRules	*
iot:ListV2LoggingLevels	ListV2LoggingLevels	無
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	無
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*

政策動作	AWS IoT API	Resources
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
iot:SetV2LoggingLevel	SetV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:SetV2LoggingOptions	SetV2LoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
iot:StartThingRegistrationTask	StartThingRegistrationTask	無
iot:StopThingRegistrationTask	StopThingRegistrationTask	無
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	無
iot:TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
iot:UpdateDimension	UpdateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:UpdateEventConfigurations	UpdateEventConfigurations	無
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	無
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:UpdateSecurityProfile	UpdateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

中的政策動作在動作之前 AWS IoT 使用下列字首：`iot:`。例如，若要授予某人許可，以列出在 中 AWS 帳戶 向 `ListThings` API 註冊的所有 IoT 物件，請在其政策中包含 `iot:ListThings`動作。政策陳述式必須包含 `Action`或 `NotAction` element。 AWS IoT 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 Describe 文字的所有動作，請包含以下動作：

```
"Action": "iot:Describe*"
```

若要查看 AWS IoT 動作清單，請參閱《IAM 使用者指南》中的 定義的動作 AWS IoT。

Device Advisor 動作

下表列出 IAM IoT Device Advisor 動作、相關的 AWS IoT Device Advisor API，以及動作所操控的資源。

政策動作	AWS IoT API	Resources
iotdeviceadvisor:CreateSuiteDefinition	CreateSuiteDefinition	無
iotdeviceadvisor:DeleteSuiteDefinition	DeleteSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:GetSuiteDefinition	GetSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:GetSuiteRun	GetSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-run-id*

政策動作	AWS IoT API	Resources
iotdeviceadvisor:GetSuiteRunReport	GetSuiteRunReport	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/ *suite-definition-id* /*suite-run-id*
iotdeviceadvisor:ListSuiteDefinitions	ListSuiteDefinitions	無
iotdeviceadvisor:ListSuiteRuns	ListSuiteRuns	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:ListTagsForResource	ListTagsForResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdeviceadvisor:StartSuiteRun	StartSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:TagResource	TagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

政策動作	AWS IoT API	Resources
iotdeviceadvisor:UntagResource	UntagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdeviceadvisor:UpdateSuiteDefinition	UpdateSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:StopSuiteRun	StopSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

AWS IoT Device Advisor 中的政策動作在動作之前使用下列字首：`iotdeviceadvisor:`。例如，若要授予某人許可，以列出 AWS 帳戶 在其中使用 ListSuiteDefinitions API 註冊的所有套件定義，請在其政策中包含 `iotdeviceadvisor:ListSuiteDefinitions`動作。

Resources

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

```
"Resource": "*"
```

AWS IoT 資源

政策動作	AWS IoT API	Resources
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
		Note ARN 中 AWS 帳戶 指定的 必須是憑證要傳輸到的帳戶。
iot:AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	無
iot:AttachPolicy	AttachPolicy	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* 或 `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:AttachThingPrincipal	AttachThingPrincipal	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** ARN 中 AWS 帳戶 指定的 必須是憑證要傳輸到的帳戶。

政策動作	AWS IoT API	Resources
iot:CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	無
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
iot:CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateJobTemplate	CreateJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*

政策動作	AWS IoT API	Resources
iot:CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
CreatePolicyVersion	iot:CreatePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** 這必須是 AWS IoT 政策，而不是 IAM 政策。
iot:CreateRoleAlias	CreateRoleAlias	(參數：roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:CreateThingGroup	CreateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 適用正在建立的群組和父群組 (若使用)
iot:CreateThingType	CreateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:CreateTopicRule	CreateTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
iot:DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeleteThingType	DeleteThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

政策動作	AWS IoT API	Resources
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (參數：authorizerName) 無
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	無
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	無
iot:DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	無
iot:DescribeJobTemplate	DescribeJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*

政策動作	AWS IoT API	Resources
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	無
iot:DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 或 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

政策動作	AWS IoT API	Resources
iot:GetIndexingConfiguration	GetIndexingConfiguration	無
iot:GetJobDocument	GetJobDocument	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 或 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	無
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*

政策動作	AWS IoT API	Resources
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	無
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	無
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	無
iot:ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* 若使用 thingGroupName 參數
iot:ListJobTemplates	ListJobTemplates	無
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListPolicyVersions	ListPolicyVersions	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListPrincipalPolicies	ListPrincipalPolicies	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
iot:ListPrincipalThings	ListPrincipalThings	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*

政策動作	AWS IoT API	Resources
iot:ListRoleAliases	ListRoleAliases	無
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:ListThingGroups	ListThingGroups	無
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	無
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	無
iot:ListThingTypes	ListThingTypes	*
iot:ListThings	ListThings	*
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:ListTopicRules	ListTopicRules	*

政策動作	AWS IoT API	Resources
iot:ListV2LoggingLevels	ListV2LoggingLevels	無
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	無
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	*
iot:SetV2LoggingLevel	SetV2LoggingLevel	*

政策動作	AWS IoT API	Resources
iot:SetV2LoggingOptions	SetV2LoggingOptions	*
iot:StartThingRegistrationTask	StartThingRegistrationTask	無
iot:StopThingRegistrationTask	StopThingRegistrationTask	無
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	無
iot:TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateEventConfigurations	UpdateEventConfigurations	無
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	無

政策動作	AWS IoT API	Resources
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

如需 ARNs 格式的詳細資訊，請參閱 Amazon Resource Name (ARNs) AWS 和服務命名空間。

有些 AWS IoT 動作無法對特定資源執行，例如用於建立資源的動作。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

若要查看 AWS IoT 資源類型及其 ARNs的清單，請參閱《IAM 使用者指南》中的 定義的資源 AWS IoT。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 AWS IoT定義的動作。

Device Advisor 資源

若要定義 AWS IoT Device Advisor IAM 政策的資源層級限制，請針對套件定義和套件執行使用以下資源 ARN 格式。

套件定義資源 ARN 格式

```
arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id
```

套件執行資源 ARN 格式

```
arn:aws:iotdeviceadvisor:region:account-id:suiterun/suite-definition-id/suite-run-id
```

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

AWS IoT 會定義自己的一組條件金鑰，也支援使用一些全域條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

AWS IoT 條件索引鍵

AWS IoT 條件索引鍵	描述	Type
aws:RequestTag/${*tag-key*}	在使用者對 AWS IoT提出的請求中存在的標籤金鑰。	String
aws:ResourceTag/${*tag-key*}	連接至 AWS IoT 資源之標籤的標籤索引鍵元件。	String
aws:TagKeys	與請求中資源相關聯的所有標籤鍵名稱清單。	String

若要查看 AWS IoT 條件金鑰清單，請參閱《IAM 使用者指南》中的 的條件金鑰 AWS IoT。若要了解您可以使用條件金鑰的動作和資源，請參閱 定義的動作 AWS IoT。

範例

若要檢視 AWS IoT 身分型政策的範例，請參閱 AWS IoT 身分型政策範例。

AWS IoT 資源型政策

以資源為基礎的政策是 JSON 政策文件，指定指定委託人可以在 AWS IoT 資源上執行的動作，以及在哪些條件下執行的動作。

AWS IoT 不支援 IAM 資源型政策。不過，它支援以 AWS IoT 資源為基礎的政策。如需詳細資訊，請參閱AWS IoT Core 政策。

以 AWS IoT 標籤為基礎的授權

您可以將標籤連接至 AWS IoT 資源，或將請求中的標籤傳遞至 AWS IoT。如需根據標籤控制存取，請使用 `iot:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。如需詳細資訊，請參閱搭配 IAM 政策使用標籤。如需標記 AWS IoT 資源的詳細資訊，請參閱 標記您的 AWS IoT 資源。

若要檢視身分型原則範例，以根據該資源上的標籤來限制存取資源，請參閱根據標籤檢視 AWS IoT 資源。

AWS IoT IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的實體。

搭配 使用臨時登入資料 AWS IoT

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫 AssumeRole 或 GetFederationToken 等 AWS STS API 操作來取得臨時安全登入資料。

AWS IoT 支援使用臨時登入資料。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成 動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

AWS IoT 不支援服務連結角色。

服務角色

此功能可讓服務代表您擔任服務角色。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

AWS IoT 身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 AWS IoT 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱 IAM 使用者指南中的在 JSON 索引標籤上建立政策。

主題

- 政策最佳實務
- 使用 AWS IoT 主控台
- 允許使用者檢視他們自己的許可
- 根據標籤檢視 AWS IoT 資源
- 根據標籤檢視 AWS IoT Device Advisor 資源

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 AWS IoT 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。

- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 IAM 安全最佳實務。

使用 AWS IoT 主控台

若要存取 AWS IoT 主控台，您必須擁有一組最低的許可。這些許可必須允許您列出和檢視 中 AWS IoT 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

為了確保這些實體仍然可以使用 AWS IoT 主控台，請將下列 AWS 受管政策連接至實體：`AWSIoTFullAccess`。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合您嘗試執行之 API 操作的動作就可以了。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

根據標籤檢視 AWS IoT 資源

您可以在身分型政策中使用條件，以根據標籤控制存取 AWS IoT 資源。此範例會示範如何建立政策，允許檢視物件。但是，只有在物件標籤 `Owner` 的值是該使用者的使用者名稱時，才會授予該許可。此政策也會授予在主控台完成此動作的必要許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListBillingGroupsInConsole",
            "Effect": "Allow",
            "Action": "iot:ListBillingGroups",
            "Resource": "*"
        },
        {
            "Sid": "ViewBillingGroupsIfOwner",
            "Effect": "Allow",
           "Action": "iot:DescribeBillingGroup",
            "Resource": "arn:aws:iot:*:*:billinggroup/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
        }
        }
    ]
}
```

您可以將此政策連接到您帳戶中的 IAM 使用者。如果名為 的使用者`richard-roe`嘗試檢視 AWS IoT 帳單群組，則帳單群組必須加上標籤 `Owner=richard-roe`或 `owner=richard-roe`。否則，他便會被拒絕存取。條件標籤鍵 `Owner` 符合 `Owner` 和 `owner`，因為條件索引鍵名稱不區分大小寫。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

根據標籤檢視 AWS IoT Device Advisor 資源

您可以在身分類型政策中使用條件，根據標籤來控制存取 AWS IoT Device Advisor 資源。以下範例顯示如何建立政策，允許檢視特定套件定義。不過，只在套件定義標籤已將 `SuiteType` 設定為值 `MQTT` 時，才會授予許可。此政策也會授予在主控台完成此動作的必要許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
        }
        }
    ]
}
```

AWS 的 受管政策 AWS IoT

若要新增許可給使用者、群組和角色，使用 AWS 受管政策比自行撰寫政策更容易。建立 IAM 客戶受管政策需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見的使用案例，並可在您的 AWS 帳戶中使用。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 服務會維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會在 AWS 受管政策中新增其他許可以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，服務很可能會更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外， AWS 支援跨多個 服務之任務函數的受管政策。例如，ReadOnlyAccess AWS 受管政策提供所有 AWS 服務和資源的唯讀存取權。當服務啟動新功能時， AWS 會為新的操作和資源新增唯讀許可。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中有關任務職能的AWS 受管政策。

> **Note**
>
> AWS IoT 適用於 AWS IoT 和 IAM 政策。此主題僅討論 IAM 政策，此政策定義控制平面和資料平面 API 操作的政策動作。另請參閱AWS IoT Core 政策。

AWS 受管政策：AWSIoTConfigAccess

您可將 `AWSIoTConfigAccess` 政策連接到 IAM 身分。

此政策授予相關的身分許可，允許存取所有 AWS IoT 組態操作。此政策會影響資料處理和儲存。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTConfigAccess。

許可詳細資訊

此政策包含以下許可。

- `iot` – 擷取 AWS IoT 資料並執行 IoT 組態動作。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AcceptCertificateTransfer",
                "iot:AddThingToThingGroup",
                "iot:AssociateTargetsWithJob",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
```

```
                "iot:AttachThingPrincipal",
                "iot:CancelCertificateTransfer",
                "iot:CancelJob",
                "iot:CancelJobExecution",
                "iot:ClearDefaultAuthorizer",
                "iot:CreateAuthorizer",
                "iot:CreateCertificateFromCsr",
                "iot:CreateJob",
                "iot:CreateKeysAndCertificate",
                "iot:CreateOTAUpdate",
                "iot:CreatePolicy",
                "iot:CreatePolicyVersion",
                "iot:CreateRoleAlias",
                "iot:CreateStream",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:CreateThingType",
                "iot:CreateTopicRule",
                "iot:DeleteAuthorizer",
                "iot:DeleteCACertificate",
                "iot:DeleteCertificate",
                "iot:DeleteJob",
                "iot:DeleteJobExecution",
                "iot:DeleteOTAUpdate",
                "iot:DeletePolicy",
                "iot:DeletePolicyVersion",
                "iot:DeleteRegistrationCode",
                "iot:DeleteRoleAlias",
                "iot:DeleteStream",
                "iot:DeleteThing",
                "iot:DeleteThingGroup",
                "iot:DeleteThingType",
                "iot:DeleteTopicRule",
                "iot:DeleteV2LoggingLevel",
                "iot:DeprecateThingType",
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
```

```
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachPrincipalPolicy",
                "iot:DetachThingPrincipal",
                "iot:DisableTopicRule",
                "iot:EnableTopicRule",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
```

```
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:RegisterCACertificate",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RejectCertificateTransfer",
                "iot:RemoveThingFromThingGroup",
                "iot:ReplaceTopicRule",
                "iot:SearchIndex",
                "iot:SetDefaultAuthorizer",
                "iot:SetDefaultPolicyVersion",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:SetV2LoggingOptions",
                "iot:StartThingRegistrationTask",
                "iot:StopThingRegistrationTask",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:TransferCertificate",
                "iot:UpdateAuthorizer",
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:UpdateEventConfigurations",
                "iot:UpdateIndexingConfiguration",
                "iot:UpdateRoleAlias",
                "iot:UpdateStream",
                "iot:UpdateThing",
                "iot:UpdateThingGroup",
                "iot:UpdateThingGroupsForThing",
                "iot:UpdateAccountAuditConfiguration",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DeleteAccountAuditConfiguration",
                "iot:StartOnDemandAuditTask",
                "iot:CancelAuditTask",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:CreateScheduledAudit",
                "iot:UpdateScheduledAudit",
                "iot:DeleteScheduledAudit",
                "iot:DescribeScheduledAudit",
```

```
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:CreateSecurityProfile",
                "iot:DescribeSecurityProfile",
                "iot:UpdateSecurityProfile",
                "iot:DeleteSecurityProfile",
                "iot:AttachSecurityProfile",
                "iot:DetachSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSIoTConfigReadOnlyAccess

您可將 `AWSIoTConfigReadOnlyAccess` 政策連接到 IAM 身分。

此政策授予相關的身分許可，允許以唯讀方式存取所有 AWS IoT 組態操作。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTConfigReadOnlyAccess。

許可詳細資訊

此政策包含以下許可。

- `iot` – 執行 IoT 組態操作的唯獨操作。

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
```

```
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:SearchIndex",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:DescribeSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSIoTDataAccess

您可將 AWSIoTDataAccess 政策連接到 IAM 身分。

此政策會授予相關的身分許可，以允許存取 AWS IoT 所有資料操作。資料操作則可透過 MQTT 或 HTTP 通訊協定傳送資料。若要在 AWS 管理主控台中檢視此政策，請參閱 AWSIoTDataAccess。

許可詳細資訊

此政策包含以下許可。

- iot – 擷取 AWS IoT 資料並允許完整存取 AWS IoT 簡訊動作。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:ListNamedShadowsForThing"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSIoTFullAccess

您可將 AWSIoTFullAccess 政策連接到 IAM 身分。

此政策授予相關的身分許可，允許完整存取所有 AWS IoT 組態和簡訊操作。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTFullAccess。

許可詳細資訊

此政策包含以下許可。

- `iot` – 擷取 AWS IoT 資料並允許完整存取 AWS IoT 組態和簡訊動作。
- `iotjobsdata` – 擷取 AWS IoT 任務資料，並允許完整存取 AWS IoT 任務資料平面 API 操作。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:*",
                "iotjobsdata:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 受管政策：AWSIoTLogging

您可將 `AWSIoTLogging` 政策連接到 IAM 身分。

此政策授予相關的身分許可，允許建立 Amazon CloudWatch Logs 日誌群組，以及將日誌串流至群組。此政策會連接至您的 CloudWatch 記錄角色。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTLogging。

許可詳細資訊

此政策包含以下許可。

- `logs` – 擷取 CloudWatch 日誌。此外，也允許建立 CloudWatch 日誌群組，以及將日誌串流至群組。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS 受管政策：AWSIoTOTAUpdate

您可將 `AWSIoTOTAUpdate` 政策連接到 IAM 身分。

此政策會授予相關聯的身分許可，允許建立 AWS IoT 任務、 AWS IoT 程式碼簽署任務，以及描述 AWS 程式碼簽署者任務的存取權。若要在 中檢視此政策 AWS 管理主控台，請參閱 `AWSIoTOTAUpdate`。

許可詳細資訊

此政策包含以下許可。

- iot – 建立 AWS IoT 任務和程式碼簽署任務。
- signer – 執行 AWS 程式碼簽署者任務的建立。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateJob",
            "signer:DescribeSigningJob"
        ],
        "Resource": "*"
    }
}
```

AWS 受管政策：AWSIoTRuleActions

您可將 AWSIoTRuleActions 政策連接到 IAM 身分。

此政策會授予關聯的身分許可，允許存取 AWS IoT 規則動作中 AWS 服務支援的所有 。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTRuleActions。

許可詳細資訊

此政策包含以下許可。

- iot - 執行發佈規則動作訊息的動作。
- dynamodb - 將訊息插入 DynamoDB 表格，或將訊息拆分至 DynamoDB 表格中多個欄。
- s3 - 將物件存放在 Amazon S3 儲存貯體中。
- kinesis - 向 Amazon Kinesis 串流物件發送訊息。
- firehose - 在 Firehose 串流物件中插入記錄。

- `cloudwatch` - 變更 CloudWatch 警示狀態，或將訊息資料發送到 CloudWatch 指標。
- `sns` - 執行使用 Amazon SNS 發佈通知的操作。此操作的範圍為 AWS IoT SNS 主題。
- `sqs` - 插入要新增至 SQS 佇列的訊息。
- `es` - 向 OpenSearch Service 服務發送訊息。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "kinesis:PutRecord",
            "iot:Publish",
            "s3:PutObject",
            "sns:Publish",
            "sqs:SendMessage*",
            "cloudwatch:SetAlarmState",
            "cloudwatch:PutMetricData",
            "es:ESHttpPut",
            "firehose:PutRecord"
        ],
        "Resource": "*"
    }
}
```

AWS 受管政策：AWSIoTThingsRegistration

您可將 `AWSIoTThingsRegistration` 政策連接到 IAM 身分。

此政策授予相關的身分許可，允許使用 `StartThingRegistrationTask` API 大量註冊物件。此政策會影響資料處理和儲存。若要在 中檢視此政策 AWS 管理主控台，請參閱 `AWSIoTThingsRegistration`。

許可詳細資訊

此政策包含以下許可。

- `iot` - 進行大量註冊時，執行建立物件並附加政策和憑證的動作。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateCertificateFromCsr",
                "iot:CreatePolicy",
                "iot:CreateThing",
                "iot:DescribeCertificate",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachThingPrincipal",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RemoveThingFromThingGroup",
                "iot:UpdateCertificate",
                "iot:UpdateThing",
                "iot:UpdateThingGroupsForThing",
                "iot:AddThingToBillingGroup",
                "iot:DescribeBillingGroup",
                "iot:RemoveThingFromBillingGroup"
```

```
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS IoT AWS 受管政策的更新

檢視自此服務開始追蹤這些變更 AWS IoT 以來， AWS 受管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱 AWS IoT 文件歷史記錄頁面上的 RSS 摘要。

變更	描述	日期
AWSIoTFullAccess – 更新現有政策	AWS IoT 新增了許可，允許使用者使用 HTTP 通訊協定存取 AWS IoT 任務資料平面 API 操作。 新的 IAM 政策字首 `iotjobsdata:` 為您提供更精細的存取控制，以存取 AWS IoT 任務資料平面端點。對於控制平面 API 操作，請依舊使用 `iot:` 字首。如需詳細資訊，請參閱AWS IoT Core HTTPS 通訊協定的 政策。	2022 年 5 月 11 日
AWS IoT 開始追蹤變更	AWS IoT 已開始追蹤其 AWS 受管政策的變更。	2022 年 5 月 11 日

對 AWS IoT 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 AWS IoT 和 IAM 時可能遇到的常見問題。

主題

- 我無權在 中執行動作 AWS IoT
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS IoT 資源

我無權在 中執行動作 AWS IoT

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在 IAM 使用者 mateojackson 嘗試使用主控台檢視物件資源的詳細資訊，但並無 iot:*DescribeThing* 許可權限時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing on resource: MyIoTThing
```

於此情況下，必須更新 mateojackson 使用者的政策，允許使用 iot:*DescribeThing* 動作存取物件資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

使用 AWS IoT Device Advisor

如果您使用 AWS IoT Device Advisor，當使用者mateojackson嘗試使用主控台檢視套件定義的詳細資訊，但沒有 iotdeviceadvisor:*GetSuiteDefinition*許可時，會發生下列範例錯誤。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:GetSuiteDefinition on resource: MySuiteDefinition
```

於此情況下，必須更新 mateojackson 使用者的政策，允許使用 iotdeviceadvisor:*GetSuiteDefinition* 動作存取 *MySuiteDefinition* 資源。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS IoT。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS IoT中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 AWS IoT 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 是否 AWS IoT 支援這些功能，請參閱 AWS IoT 如何使用 IAM。
- 若要了解如何在您擁有 AWS 帳戶 的 資源間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 IAM 使用者中提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

記錄和監控

監控是維護 和 AWS 解決方案的可靠性、可用性 AWS IoT 和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時更輕鬆地偵錯。如需記錄和監視程序的相關資訊，請參閱 監控 AWS IoT

監控工具

AWS 提供可用來監控的工具 AWS IoT。您可設定部分這些工具以為您執行監控工作。部分工具將需要手動操作。建議您盡可能自動化監控任務。

自動化監控工具

您可以使用下列自動化監控工具，在發生錯誤時監看 AWS IoT 和報告 ：

- Amazon CloudWatch 警示：監看指定時段內的單一指標，並根據與多個時段內給定之閾值相對的指標值來執行一或多個動作。此動作是傳送到 Amazon Simple Notification Service (Amazon SNS) 主題或 Amazon EC2 Auto Scaling 政策的通知。CloudWatch 警示不會只因為處於特定狀態而叫用動作。狀態必須已變更，且在指定的期間數內維持此狀態。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 AWS IoT 警示和指標。
- Amazon CloudWatch Logs – 從 AWS CloudTrail 或其他來源監控、存放和存取您的日誌檔案。Amazon CloudWatch Logs 也可讓您查看 Device Advisor 測試案例採取的關鍵步驟 AWS IoT 、產生的事件和從裝置或在測試執行 AWS IoT Core 期間傳送的 MQTT 訊息。這些日誌可讓您在裝置上偵錯並採取更正動作。如需詳細資訊，請參閱 AWS IoT 使用 CloudWatch Logs 進行監控。如需 Amazon CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的監控記錄檔。
- Amazon CloudWatch Events：匹配事件並將它們路由至一或多個目標函式或串流以進行變更、擷取狀態資訊，以及採取修正動作。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch Events？。
- AWS CloudTrail 日誌監控 – 在帳戶之間共用日誌檔案、透過將日誌檔案傳送到 CloudTrail CloudWatch Logs 來即時監控 CloudTrail 日誌檔案、在 Java 中寫入日誌處理應用程式，以及驗證您的日誌檔案在 CloudTrail 交付後並未變更。如需詳細資訊，請參閱 使用 記錄 AWS IoT API 呼叫 AWS CloudTrail，亦請參閱《AWS CloudTrail 使用者指南》中的使用 CloudTrail 記錄檔案。

手動監控工具

監控的另一個重要部分 AWS IoT 包括手動監控 CloudWatch 警示未涵蓋的項目。 AWS IoT、CloudWatch 和其他 AWS 服務主控台儀表板可讓您at-a-glance檢視 AWS 環境的狀態。我們建議您也檢查 上的日誌檔案 AWS IoT。

- AWS IoT 儀表板會顯示：
  - 憑證授權機構憑證
  - 憑證
  - 政策
  - 規則
  - 物件
- CloudWatch 首頁顯示：

- 目前警示與狀態。
- 警示與資源的圖形。
- 服務運作狀態。

您可以使用 CloudWatch 執行下列動作：

- 建立自訂儀表板來監控您關心的服務。
- 用於疑難排解問題以及探索驅勢的圖形指標資料。
- 搜尋和瀏覽您的所有 AWS 資源指標。
- 建立與編輯要通知發生問題的警示。

AWS IoT 核心的合規驗證

我們的新 AWS 註冊體驗並非針對受管制的工作負載而設計。如果您正在使用我們新的 AWS 註冊體驗，但想要將 AWS 用於受管制的工作負載，您可以註冊 AWS （進階） 或為您的 AWS 環境啟用進階功能。

若要了解 是否 AWS 服務 在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

AWS IoT Core 中的彈性

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體隔離和隔離的可用區域，這些可用區域以低延遲、高輸送量和高度備援聯網連接。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

AWS IoT Core 會將裝置的相關資訊儲存在裝置登錄檔中。它也會存放憑證授權機構憑證、裝置憑證和裝置影子資料。發生硬體或網路失敗時，此資料會自動跨可用區域複寫，但不會跨區域複寫。

AWS IoT Core 會在裝置登錄檔更新時發佈 MQTT 事件。您可以使用這些訊息來備份您的登錄資料，並將其儲存在某處，例如 DynamoDB 資料表。您負責儲存為您或您自行 AWS IoT Core 建立的憑證。

裝置影子會儲存裝置的狀態資料，並在裝置重新上線時重新傳送。 AWS IoT Device Advisor 會儲存測試套件組態的相關資訊。發生硬體或網路失敗時，此資料會自動複寫。

AWS IoT Core 資源是區域特定的，除非您特別這樣做， AWS 區域 否則不會跨 複寫。

如需安全性最佳實務的相關資訊，請參閱 中的安全最佳實務 AWS IoT Core。

AWS IoT Core 搭配界面 VPC 端點使用

透過 AWS IoT Core，您可以使用介面 VPC 端點，在虛擬私有雲端 (VPC) 內建立 IoT 控制平面端點和 IoT 資料端點。 https://docs.aws.amazon.com/vpc/latest/userguide/vpce-interface.html#create-interface-endpoint介面 VPC 端點採用一種 AWS 技術 AWS PrivateLink，可讓您 AWS 使用私有 IP 地址存取在 上執行的服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud。

若要將遠端網路上 欄位中的裝置，例如公司網路連接到 Amazon VPC，請參閱Network-to-Amazon VPC 連線矩陣中列出的選項。

目錄

- 建立 AWS IoT Core 控制平面的 VPC 端點
- 為 AWS IoT Core 資料平面建立 VPC 端點
- 為 AWS IoT Core 登入資料提供者建立 VPC 端點
- 建立 Amazon VPC 介面端點
- 設定私有託管區域
- 透過 VPC AWS IoT Core 端點控制對 的存取
- 限制
- 使用 擴展 VPC 端點 AWS IoT Core
- 搭配使用自訂網域與 VPC 端點
- 的 VPC 端點可用性 AWS IoT Core
- 搭配界面 VPC 端點使用 AWS IoT Device Management 安全通道

建立 AWS IoT Core 控制平面的 VPC 端點

您可以為 AWS IoT Core 控制平面 API 建立 VPC 端點，將您的裝置連線至 AWS IoT 服務和其他 AWS 服務。若要開始使用 VPC 端點，請建立介面 VPC 端點，然後選取 AWS IoT Core 做為 AWS 服務。如果您使用的是 CLI，請先呼叫 describe-vpc-endpoint-services，以確保您選擇特定 中存在 AWS IoT Core 的可用區域 AWS 區域。例如，在 us-east-1 中，此命令將如下所示：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.api
```

請參閱以下詳細說明，為 AWS IoT Core 控制平面建立 Amazon VPC 介面端點。

為 AWS IoT Core 資料平面建立 VPC 端點

您可以為 AWS IoT Core 資料平面 API 建立 VPC 端點，將您的裝置連線至 AWS IoT 服務和其他 AWS 服務。若要開始使用 VPC 端點，請建立介面 VPC 端點，然後選取 AWS IoT Core 做為 AWS 服務。如果您使用的是 CLI，請先呼叫 describe-vpc-endpoint-services，以確保您選擇特定 中存在 AWS IoT Core 的可用區域 AWS 區域。例如，在 us-east-1 中，此命令將如下所示：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.data
```

> **Note**
>
> 已停用自動建立 DNS 記錄的 VPC 功能。若要連接這些端點，您必須手動建立私有 DNS 記錄。如需有關私有 VPC DNS 記錄的詳細資訊，請參閱介面端點的私有 DNS。如需 AWS IoT Core VPC 限制的詳細資訊，請參閱 限制 。

若要將 MQTT 用戶端連線至 VPC 端點介面：

- 您必須手動在連接到 VPC 的私有託管區域中建立 DNS 記錄。若要開始使用，請參閱建立私有託管區域。

- 在私有託管區域內，為 VPC 端點的每個彈性網路介面 IP 建立別名記錄。如果多個 VPC 端點有多個網路介面 IP，請在所有加權記錄中建立具有相同權重的加權 DNS 記錄。依描述欄位中的 VPC 端點 ID 進行篩選時，可從 DescribeNetworkInterfaces API 呼叫中取得這些 IP 地址。

請參閱以下詳細說明，以建立 Amazon VPC 介面端點和設定資料平面的私有託管區域。 AWS IoT Core

為 AWS IoT Core 登入資料提供者建立 VPC 端點

您可以建立 VPC 端點 AWS IoT Core 登入資料提供者，以使用用戶端憑證型身分驗證來連接裝置，並取得 AWS Signature 第 4 版格式的臨時 AWS 登入資料。若要開始使用 AWS IoT Core 登入資料提供者的 VPC 端點，請執行 create-vpc-endpoint CLI 命令來建立介面 VPC 端點，然後選取 AWS IoT

Core 登入資料提供者做為 AWS 服務。為了確保您選擇特定 中存在 AWS IoT Core 的可用區域 AWS 區域，請先執行 describe-vpc-endpoint-services 命令。例如，在 us-east-1 中，此命令將如下所示：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.iot.credentials
```

> **Note**
>
> 已停用自動建立 DNS 記錄的 VPC 功能。若要連接這些端點，您必須手動建立私有 DNS 記錄。如需有關私有 VPC DNS 記錄的詳細資訊，請參閱介面端點的私有 DNS。如需 AWS IoT Core VPC 限制的詳細資訊，請參閱 限制 。

若要將 HTTP 用戶端連線至 VPC 端點介面：

- 您必須手動在連接到 VPC 的私有託管區域中建立 DNS 記錄。若要開始使用，請參閱建立私有託管區域。

- 在私有託管區域內，為 VPC 端點的每個彈性網路介面 IP 建立別名記錄。如果多個 VPC 端點有多個網路介面 IP，請在所有加權記錄中建立具有相同權重的加權 DNS 記錄。依描述欄位中的 VPC 端點 ID 進行篩選時，可從 DescribeNetworkInterfaces API 呼叫中取得這些 IP 地址。

請參閱以下詳細說明，以建立 Amazon VPC 介面端點和設定登入資料提供者的私有託管區域。 AWS IoT Core

建立 Amazon VPC 介面端點

您可以建立介面 VPC 端點，以連線至由 提供支援 AWS 的服務 AWS PrivateLink。使用下列程序來建立連線至 AWS IoT Core 資料平面或 AWS IoT Core 憑證提供者的介面 VPC 端點。如需詳細資訊，請參閱使用介面 VPC 端點存取 AWS 服務。

> **Note**
>
> 為 AWS IoT Core 資料平面和 AWS IoT Core 登入資料提供者建立 Amazon VPC 介面端點的程序類似，但您必須進行端點特定變更，才能使連線正常運作。

使用 VPC 端點主控台建立介面 VPC 端點

1. 導覽至 VPC 端點主控台，在左側選單的虛擬私有雲端下，選擇端點，然後選擇建立端點。
2. 在建立端點頁面中，指定下列資訊。
   - 選擇服務類別的 AWS 服務。
   - 針對 Service Name (服務名稱)，輸入關鍵字 `iot` 進行搜尋。在顯示的 `iot` 服務清單中，選擇端點。

     如果您為 AWS IoT Core 控制平面建立 VPC 端點，請選擇 的 AWS IoT Core 控制平面 API 端點 AWS 區域。端點的格式為 `com.amazonaws.`*`region`*`.iot.api`。

     如果您為 AWS IoT Core 資料平面建立 VPC 端點，請選擇您區域 AWS IoT Core 的資料平面 API 端點。端點的格式為 `com.amazonaws.`*`region`*`.iot.data`。

     如果您為 AWS IoT Core 登入資料提供者建立 VPC 端點，請選擇您區域的 AWS IoT Core 登入資料提供者端點。端點的格式為 `com.amazonaws.`*`region`*`.iot.credentials`。

     如果您為聯邦資訊處理標準 (FIPS) 區域建立 VPC 端點，請選擇 的 FIPS API 端點 AWS 區域。端點的格式為 `com.amazonaws.`*`region`*`.iot-fips.api`。這僅適用於控制平面。

     > **Note**
     >
     > 中國區域中 AWS IoT Core 資料平面的服務名稱格式為 `cn.com.amazonaws.`*`region`*`.iot.data`。中國區域中 AWS IoT Core 控制平面的服務名稱格式為 `com.amazonaws.`*`region`*`.iot.api`。

   - 針對 VPC 和 Subnets (子網路)，選擇要在其中建立端點的 VPC，以及要在其中建立端點網路的可用區域 (AZ)。
   - 針對啟用 DNS 名稱，請確定未針對 AWS IoT Core 資料平面和 AWS IoT Core 憑證提供者選取此端點的啟用。 AWS IoT Core 資料平面和 AWS IoT Core 登入資料提供者都不支援私有 DNS 名稱。

     對於 AWS IoT Core 控制平面，預設會選取啟用 DNS 名稱。這可確保對控制平面公有 AWS IoT Core 端點的任何請求都會改為透過 VPC 端點路由。啟用此功能時，您不需要設定私有託管區域。
   - 針對 Security group (安全群組)，選擇要與端點網路介面建立關聯的安全群組。
   - 您可以選擇性地新增或移除標籤。標籤是您用來與端點建立關聯的名稱值對。
3. 若要建立 VPC 端點，請選擇 Create endpoint (建立端點)。

建立 AWS PrivateLink 端點之後，您會在端點的詳細資訊索引標籤中看到 DNS 名稱清單。您可以使用您在本節中建立的其中一個 DNS 名稱來設定私有託管區域。如果您使用的是 AWS IoT Core 控制平面，則不需要設定私有託管區域。

設定私有託管區域

> **Note**
>
> 如果您使用 AWS IoT Core 控制平面並選取啟用 DNS 名稱，則不需要設定私有託管區域。如果您停用它，必須遵循此程序來設定私有託管區域。

您可以使用您在上一節中建立的其中一個 DNS 名稱來設定私有託管區域。

對於 AWS IoT Core 資料平面

DNS 名稱必須是您的網域組態名稱或`IoT:Data-ATS`端點。DNS 名稱範例可以是： *xxx*`-ats.data.iot.`*region*`.amazonaws.com`。

對於 AWS IoT Core 登入資料提供者

DNS 名稱必須是您的`iot:CredentialProvider`端點。範例 DNS 名稱可以是：*xxxx*`.credentials.iot.`*region*`.amazonaws.com`。

對於 AWS IoT Core 控制平面

DNS 名稱必須是您的 AWS IoT Core 控制平面端點。 AWS IoT Core 控制平面的範例 DNS 名稱為 *xxxx*`.api.iot.`*region*`.amazonaws.com`。

> **Note**
>
> 為 AWS IoT Core 資料平面和 AWS IoT Core 登入資料提供者設定私有託管區域的程序類似，但您必須進行端點特定變更，才能使連線正常運作。

建立私有託管區域

使用 Route 53 主控台建立私有託管區域

1. 導覽至 Route 53 Hosted zones (託管區域) 主控台，然後選擇 Create hosted zone (建立託管區域)。

2. 在 Create hosted zone (建立託管區域) 頁面中，指定下列資訊。
   - 針對網域名稱，輸入 `iot:Data-ATS`或 端點的`iot:CredentialProvider`端點地址。下列 AWS CLI 命令顯示如何透過公有網路取得端點： `aws iot describe-endpoint --endpoint-type iot:Data-ATS`或 `aws iot describe-endpoint --endpoint-type iot:CredentialProvider`。

     > **Note**
     >
     > 如果您使用自訂網域，請參閱搭配使用自訂網域與 VPC 端點。 AWS IoT Core 登入資料提供者不支援自訂網域。

   - 針對 Type (類型)，選擇 Private Hosted Zone (私有託管區域)。
   - 或者，您可以新增或移除要與託管區域建立關聯的標籤。
3. 若要建立私有託管區域，請選擇 Create hosted zone (建立託管區域)。

如需詳細資訊，請參閱建立私有託管區域。

建立記錄

在建立了私有託管區域之後，您可以建立記錄，告訴 DNS 您想要流量路由至該網域的方式。

建立記錄

1. 在顯示的託管區域清單，選擇您先前建立的私有託管區域，然後選擇 Create record (建立記錄)。
2. 使用精靈方法來建立記錄。如果主控台呈現 Quick create (快速建立) 方法，請選擇 Switch to wizard (切換至精靈)。
3. 為 Routing policy (路由政策) 選擇 Simple Routing (簡易路由)，然後選擇 Next (下一步)。
4. 在 Configure records (設定記錄) 頁面中，選擇 Define simple record (定義簡易記錄)。
5. 在 Define simple record (定義簡易記錄) 頁面中：
   - 針對記錄名稱，輸入`iot:Data-ATS`端點或`iot:CredentialProvider`端點。這必須與私有託管區域名稱相同。
   - 對於記錄類型，如果您只想要 IPv4 支援，請將值保留為 `A - Routes traffic to an IPv4 address and some AWS resources`。如果您只想要 IPv6 支援，請將值保留為 `AAAA - Routes traffic to an IPv6 address and some AWS resources`。如果您想要雙堆疊支援 (IPv4 和 IPv6)，請使用相同的記錄名稱和值/路由流量，`AAAA`在託管區域中建立兩個記錄 (`A`和 。

- 針對 Value/Route traffic to (值/路由流量至)，選擇 Alias to VPC endpoint (VPC 端點的別名)。接著，選擇您的 Region (區域)，然後從顯示的端點清單中選擇您先前建立的端點，如???所述。

6. 選擇 Define simple record (定義簡易記錄) 來建立您的記錄。

透過 VPC AWS IoT Core 端點控制對 的存取

您可以使用 VPC 條件內容金鑰，將裝置存取限制 AWS IoT Core 為僅允許透過 VPC 端點存取 。AWS IoT Core 支援下列 VPC 相關內容金鑰：

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core 不支援 VPC 端點的端點政策。

例如，以下政策授予許可，以 AWS IoT Core 使用符合物件名稱的用戶端 ID 連線至 ，並發佈至物件名稱前綴的任何主題，條件是連接至具有特定 VPC 端點 ID 之 VPC 端點的裝置。此政策會拒絕與公有 IoT 資料端點的連線嘗試。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
            }
```

```
        }

        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

限制

AWS IoT Core 控制平面端點、AWS IoT Core 資料端點和AWS IoT Core 登入資料提供者端點目前支援 VPC 端點。只有在使用 AWS IoT Core 控制平面時，聯邦資訊處理標準 (FIPS) 端點才支援 VPC 端點。

IoT 控制平面 VPC 端點的限制

本節涵蓋 IoT 控制平面 VPC 端點的限制。

- VPC 端點只會提供 ATS 憑證。
- 控制平面端點不支援自訂網域。
- 如需 FIPS 安全政策的相關資訊，請參閱 FIPS 安全政策。

IoT 資料 VPC 端點的限制

本節涵蓋 IoT 資料 VPC 端點的限制。

- MQTT 持續作用期間限制為 230 秒。保持存活期間超過自動減少到 230 秒的時間。
- 每個 VPC 端點支援總共 100,000 個並行連網裝置。如果您需要更多連線，請參閱 使用 擴展 VPC 端點 AWS IoT Core 。
- VPC 端點只會為 ATS 憑證提供服務，但自訂網域除外。

- 不支援 VPC 端點政策。
- 對於為 AWS IoT Core 資料平面建立的 VPC 端點， AWS IoT Core 不支援使用區域或區域公有 DNS 記錄。

登入資料提供者端點的限制

本節涵蓋登入資料提供者 VPC 端點的限制。

- VPC 端點只會提供 ATS 憑證。
- 不支援 VPC 端點政策。
- 登入資料提供者端點不支援自訂網域。
- 對於為 AWS IoT Core 登入資料提供者建立的 VPC 端點， AWS IoT Core 不支援使用區域或區域公有 DNS 記錄。

使用 擴展 VPC 端點 AWS IoT Core

AWS IoT Core 介面 VPC 端點透過單一介面端點限制為 100，000 個連線裝置。如果您的使用案例要求更多並行連線到代理程式，則我們建議使用多個 VPC 端點，並跨您的介面端點手動路由裝置。建立私有 DNS 記錄以將流量路由到 VPC 端點時，請確保建立的加權記錄數量與 VPC 端點數量一樣多，以將流量分配到多個端點。

搭配使用自訂網域與 VPC 端點

如果您想要搭配 VPC 端點使用自訂網域，您必須在私有託管區域中建立自訂網域名稱記錄，並在 Route53 中建立路由記錄。如需詳細資訊，請參閱建立私有託管區域。

> **Note**
>
> 自訂網域僅支援 AWS IoT Core 資料端點。

的 VPC 端點可用性 AWS IoT Core

AWS IoT Core 介面 VPC 端點適用於所有AWS IoT Core 支援的 區域。中國區域和 不支援憑證提供者的 AWS IoT Core AWS IoT Core 介面 VPC 端點 AWS GovCloud (US) Regions。

搭配界面 VPC 端點使用 AWS IoT Device Management 安全通道

AWS IoT Device Management 安全通道支援介面 VPC 端點。您可以使用 VPC 端點來保留 VPC 與 AWS 網路 AWS IoT Secure Tunneling 之間的流量，而不需要網際網路閘道、NAT 裝置、VPN 連線或 AWS Direct Connect 連線。

介面 VPC 端點採用 AWS PrivateLink技術，這項技術可讓您使用私有 IP 地址來私下存取 服務。如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的使用介面 VPC 端點存取 AWS 服務。

目錄

- 先決條件
- 透過 VPC 端點接收通道通知
- 建立 VPC 端點以進行安全通道
- 在 Proxy 伺服器上設定 VPC 端點政策
- 後續步驟

先決條件

為 建立 VPC 端點之前 AWS IoT Secure Tunneling，請確認您有下列項目：

- 具有建立 VPC 端點所需許可 AWS 的帳戶。
- 您 AWS 帳戶中的 VPC。
- 了解 AWS IoT Device Management 安全通道概念。
- 熟悉 VPC 端點政策和 AWS Identity and Access Management (IAM)

透過 VPC 端點接收通道通知

若要透過 VPC 端點接收通道通知，您的裝置可以透過 VPC 端點連線至 AWS IoT Core 資料平面，並訂閱安全通道預留 MQTT 主題。

如需如何在 AWS IoT Core 資料平面中建立和設定 VPC 端點的指示，請參閱《 AWS IoT 開發人員指南》中的搭配使用 AWS IoT Core 與介面 VPC 端點。

建立 VPC 端點以進行安全通道

您可以為安全通道控制平面和代理伺服器建立 VPC 端點。

建立用於安全通道的 VPC 端點

1. 遵循《Amazon VPC 開發人員指南》中建立介面端點的步驟
2. 針對服務名稱，根據您的端點類型選擇下列其中一個選項：

   控制平面

   - 標準： `com.amazonaws.<region>.iot.tunneling.api`
   - FIPS （適用於 FIPS 區域）： `com.amazonaws.<region>.iot-fips.tunneling.api`

   代理伺服器

   - 標準： `com.amazonaws.<region>.iot.tunneling.data`
   - FIPS （適用於 FIPS 區域）： `com.amazonaws.<region>.iot-fips.tunneling.data`

   將 *<region>* 取代為您的 AWS 區域。例如 `us-east-1`。
3. 根據您的網路需求，完成 VPC 端點建立程序中剩餘的步驟。

在 Proxy 伺服器上設定 VPC 端點政策

除了用來授權通道連線的用戶端存取字符型授權之外，您還可以使用 VPC 端點政策進一步限制裝置如何使用 VPC 端點連線到安全通道 Proxy Server。VPC 端點政策遵循類似 IAM 的語法，並在 VPC 端點本身上設定。

請注意，代理伺服器 VPC 端點政策唯一支援的 IAM 動作是 `iot:ConnectToTunnel`。

以下是不同 VPC 端點政策的範例。

Proxy 伺服器 VPC 端點政策範例

下列範例顯示常見使用案例的 Proxy Server VPC 端點政策組態。

Example- 預設政策

此政策允許 VPC 中的裝置連線到任何帳戶中建立 AWS 端點 AWS 區域 之相同 中的任何通道。

```
{
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Example- 限制存取特定 AWS 帳戶

此政策允許 VPC 端點僅連線到特定 AWS 帳戶中的通道。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*",
                "arn:aws:iot:us-east-1:444455556666:tunnel/*"
            ]
        }
    ]
}
```

Example- 依通道端點限制連線

您可以限制 VPC 端點存取，僅允許裝置連線到通道的來源或目的地端。

僅限來源：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "source"
                }
```

```
            }
        }
    ]
}
```

僅限目的地：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
        }
    ]
}
```

Example- 根據資源標籤限制存取

此政策允許 VPC 端點僅連線到以特定鍵值對標記的通道。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Environment": "Production"
                }
            }
        }
    ]
}
```

Example- 合併政策條件

此政策示範結合多個政策元素。它允許連線到特定 AWS 帳戶中的任何通道，但前提是通道標記為 `AllowConnectionsThroughPrivateLink`且`true`用戶端未連線到通道的目的地端。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/AllowConnectionsThroughPrivateLink": "true"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
        }
    ]
}
```

後續步驟

為 建立和設定 VPC 端點之後 AWS IoT Secure Tunneling，請考慮下列事項：

- 透過端點連線裝置，以測試您的 VPC 端點組態。
- 透過 Amazon CloudWatch 指標監控 VPC 端點用量。

- 根據您的安全需求，視需要檢閱和更新 VPC 端點政策。

如需 AWS IoT Device Management 安全通道的詳細資訊，請參閱AWS IoT Secure Tunneling。

中的基礎設施安全 AWS IoT

作為受管服務的集合， AWS IoT 受到 Amazon Web Services：安全程序概觀白皮書中所述的 AWS 全球網路安全程序的保護。

您可以使用 AWS 發佈的 API 呼叫， AWS IoT 透過網路存取 。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。如需詳細資訊，請參閱中的傳輸安全性 AWS IoT Core。

請求必須使用存取金鑰 ID 和與 IAM 委託人相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 產生臨時安全憑證來簽署請求。

使用 AWS IoT Core 對生產機群或裝置進行安全監控

IoT 機群可由大量的裝置組成具有多樣化的功能、長時間在線上，並且散佈在多個地理位置。這些特點使機群的設定變得複雜且極易出錯。由於裝置通常受限於運算能力、記憶體和儲存功能，因此限制了裝置本身的加密和其他形式的安全性的使用。此外，裝置通常會使用具有已知漏洞的軟體。這些因素讓 IoT 機群成為對駭客極具吸引力的目標，且難以持續保護裝置機群。

AWS IoT Device Defender 透過提供識別安全問題和偏離最佳實務的工具來解決這些挑戰。您可以使用 AWS IoT Device Defender 來分析、稽核和監控連線裝置，以偵測異常行為，並降低安全風險。 AWS IoT Device Defender 可以稽核裝置機群，以確保其遵守安全最佳實務，並偵測裝置上的異常行為。這可讓您在 AWS IoT 裝置機群中強制執行一致的安全政策，並在裝置遭到入侵時快速回應。如需詳細資訊，請參閱AWS IoT Device Defender。

AWS IoT Device Advisor 會視需要推送更新並修補您的機群。 AWS IoT Device Advisor 會自動更新測試案例。您選取的測試案例一律搭配最新版本。如需詳細資訊，請參閱Device Advisor。

中的安全最佳實務 AWS IoT Core

本節包含 的安全最佳實務相關資訊 AWS IoT Core。如需產業 IoT 解決方案安全規則的相關資訊，請參閱產業 IoT 解決方案的十大安全黃金法則。

在 中保護 MQTT 連線 AWS IoT

AWS IoT Core 是一種受管雲端服務，可讓連線的裝置輕鬆安全地與雲端應用程式和其他裝置互動。AWS IoT Core 支援 HTTP、WebSocket 和 MQTT，這是一種輕量型通訊協定，專門設計來容忍間歇性連線。如果您 AWS IoT 使用 MQTT 連線至 ，則每個連線都必須與稱為用戶端 ID 的識別符建立關聯。MQTT 用戶端 ID 是獨一無二的，可識別 MQTT 連線。如果使用已針對另一個連線宣告的用戶端 ID 建立新連線， AWS IoT 訊息中介裝置會捨棄舊連線，以允許新連線。每個 AWS 帳戶 和每個 中的用戶端 IDs都必須是唯一的 AWS 區域。這表示您不需要在 外部 AWS 帳戶 或 內跨區域強制執行用戶端 IDs 的全域唯一性 AWS 帳戶。

在您的裝置機群上捨棄 MQTT 連線的影響和嚴重性，取決於多個因素。其中包含：

- 您的使用案例 （ 例如，您的裝置傳送的資料 AWS IoT、多少資料，以及傳送資料的頻率 ）。
- 您的 MQTT 用戶端組態 (例如，自動重新連線設定、關聯的退避計時，以及 MQTT 持久性工作階段的使用)。
- 裝置資源限制。
- 連線中斷的根本原因，以及其積極性和持久性。

為了避免用戶端 ID 衝突及其潛在的負面影響，請確定每個裝置或行動應用程式都有 AWS IoT 或 IAM 政策，以限制哪些用戶端 IDs 可用於與 AWS IoT 訊息中介裝置的 MQTT 連線。例如，您可以使用 IAM 政策，藉由使用已在使用中的用戶端 ID 來防止裝置意外關閉其他裝置的連線，如需詳細資訊，請參閱Authorization。

機群中的所有裝置都必須具有僅授權預期動作權限的登入資料，其中包括 （ 但不限於 ） AWS IoT MQTT 動作，例如發佈訊息或訂閱具有特定範圍和內容的主題。特定的許可政策可能因您的使用案例而異。識別最符合您業務和安全性需求的許可政策。

為了簡化許可政策的建立和管理，您可以使用 AWS IoT Core 政策變數 和 IAM 政策變數。政策變數可以放置在政策中，當政策接受評估時，該變數可由來自裝置的請求值取代。透過使用政策變數，您可以建立一個單一政策，以授與許可給多個裝置。您可以根據 AWS IoT 您的帳戶組態、身分驗證機制和用於連線至 AWS IoT 訊息中介裝置的網路通訊協定，來識別使用案例的相關政策變數。不過，若要撰寫最佳的許可政策，您需要考慮您使用案例的特定情形和威脅模型。

例如，如果您在 AWS IoT 登錄檔中註冊裝置，您可以在 AWS IoT 政策中使用物件政策變數，根據物件名稱、物件類型和物件屬性值等物件屬性來授予或拒絕許可。物件名稱是從物件連線時所傳送的 MQTT 連線訊息中的用戶端 ID 取得 AWS IoT。當物件使用 TLS 交互身分驗證 AWS IoT 透過 MQTT 連線至 ，或使用已驗證的 Amazon Cognito 身分透過 WebSocket 通訊協定連線至 MQTT 時，會取代

物件政策變數。您可以使用 AttachThingPrincipal API，將憑證和驗證的 Amazon Cognito 身分連接到物件。iot:Connection.Thing.ThingName 是一種實用的物件政策變數，可強制執行用戶端 ID 限制。下列範例 AWS IoT 政策要求已註冊物件的名稱做為 MQTT 連線至 AWS IoT 訊息中介裝置的用戶端 ID：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

如果您想要識別持續的用戶端 ID 衝突，您可以啟用和使用 CloudWatch Logs AWS IoT。對於 AWS IoT 訊息中介裝置因用戶端 ID 衝突而中斷連線的每個 MQTT 連線，會產生類似以下的日誌記錄：

```
{
    "timestamp": "2019-04-28 22:05:30.105",
    "logLevel": "ERROR",
    "traceId": "02a04a93-0b3a-b608-a27c-1ae8ebdb032a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "clientId01",
    "principalId": "1670fcf6de55adc1930169142405c4a2493d9eb5487127cd0091ca0193a3d3f6",
    "sourceIp": "203.0.113.1",
    "sourcePort": 21335,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID"
}
```

您可以使用 CloudWatch Logs 篩選條件 (例如 {$.reason= "DUPLICATE_CLIENT_ID" }) 來搜尋用戶端 ID 衝突的執行個體，或設定 CloudWatch 指標篩選條件與對應的 CloudWatch 警示，以供持續監控和報告。

您可以使用 AWS IoT Device Defender 來識別過度寬鬆 AWS IoT 和 IAM 政策。 AWS IoT Device Defender 也提供稽核檢查，通知您機群中的多個裝置是否使用相同的用戶端 ID 連線到 AWS IoT 訊息中介裝置。

您可以使用 AWS IoT Device Advisor 來驗證您的裝置是否可以可靠地連線至 AWS IoT Core 並遵循安全最佳實務。

另請參閱

- AWS IoT Core
- AWS IoT的安全性功能
- AWS IoT Core 政策變數
- IAM 政策變數
- Amazon Cognito 身分
- AWS IoT Device Defender
- 的 CloudWatch Logs AWS IoT

讓裝置的時鐘保持同步

在裝置上保持準確的時間是很重要的。X.509 憑證具到期日期和時間。裝置上的時鐘用來驗證伺服器憑證是否仍然有效。如果您正在建置商用 IoT 裝置，請記住您的產品在售出前可能會長期存放。在這段期間，即時時鐘可能會漂移，電池可能會放電，因此在工廠設定時間並不足夠。

對大多數系統而言，這表示裝置的軟體必須包含網路時間通訊協定 (NTP) 用戶端。裝置應該等到與 NTP 伺服器同步時，才會嘗試連線到 AWS IoT Core。如果無法這麼做，系統應該提供方法讓使用者能夠設定裝置的時間，以便讓後續連線成功。

一旦裝置與 NTP 伺服器同步，就可以開啟與 AWS IoT Core的連線。允許的時脈偏移會取決於您嘗試對連線執行的操作而定。

驗證伺服器憑證

裝置與 互動的第一件事 AWS IoT 是開啟安全連線。當您將裝置連線到 時 AWS IoT，請確定您正在與交談 AWS IoT ，而不是另一個伺服器模擬 AWS IoT。每個 AWS IoT 伺服器都會佈建為網域發行的憑證`iot.amazonaws.com`。此憑證 AWS IoT 是由信任的憑證授權機構核發給 ，該授權機構會驗證我們網域的身分和擁有權。

當裝置連線時，最先 AWS IoT Core 做的就是傳送伺服器憑證給裝置。裝置可以驗證其預期連線至 `iot.amazonaws.com`，以及該連線端的伺服器是否擁有來自該網域之受信任授權單位的憑證。

TLS 憑證採用 X.509 格式，並包含各種資訊，例如組織的名稱、位置、網域名稱和有效期間。有效期間以稱為 `notBefore` 和 `notAfter` 的一對時間值來指定。這類服務會對其伺服器憑證 AWS IoT Core 使用有限的有效期間（例如一年），並在舊憑證過期之前開始提供新的憑證。

使用每個裝置單一身分

使用每個用戶端單一身分。裝置通常使用 X.509 用戶端憑證。Web 與行動應用程式使用 Amazon Cognito 身分。這可讓您將精細的許可套用至您的裝置。

例如，您有由行動電話裝置組成的應用程式，該裝置從兩個不同的智慧型家用物件 (燈泡和電熱器) 接收狀態更新。燈泡會傳送其電池的電量狀態，恆溫器會傳送報告溫度的訊息。

AWS IoT 會個別驗證裝置，並個別處理每個連線。您可以使用授權政策套用微調的存取控制。您可以定義電熱器的政策，以允許電熱器發佈至主題空間。您可以為燈泡定義個別的政策，以允許燈泡發佈至不同的主題空間。最後，您可以定義行動應用程式的政策，只允許它連線和訂閱電熱器和燈泡的主題，以接收來自這些裝置的訊息。

盡可能套用最低權限的政策，並盡可能降低每個裝置的許可範圍。所有裝置或使用者在 中都應有一個 AWS IoT 政策 AWS IoT ，只允許它與已知的用戶端 ID 連線，並發佈和訂閱已識別和固定的主題集。

使用秒 AWS 區域 作為備份

請考慮在一秒內將資料的副本 AWS 區域 儲存為備份。請注意，不再提供名為 的災難復原 AWS IoT AWS 解決方案。雖然相關聯的 GitHub 程式庫仍可存取，但已於 2023 年 7 月 AWS 棄用，不再提供維護或支援。若要實作您自己的解決方案或探索其他支援選項，請造訪聯絡 AWS。如果有與您的帳戶相關聯的 AWS 技術客戶經理，請聯絡他們尋求協助。

使用及時佈建

手動建立和佈建每個裝置可能很耗時。 AWS IoT 提供一種方式來定義範本，以在首次連線時佈建裝置 AWS IoT。如需詳細資訊，請參閱即時佈建。

執行 AWS IoT Device Advisor 測試的許可

下列政策範本顯示執行 AWS IoT Device Advisor 測試案例所需的最低許可和 IAM 實體。您需要將 *`your-device-role-arn`* 取代為在先決條件下建立的裝置角色 Amazon Resource Name (ARN)。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/your-thing-
group",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iotdeviceadvisor.amazonaws.com"
            }
        }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iotjobsdata:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iotjobsdata:GetPendingJobExecutions",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
```

```
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        }
    ]
}
```

Device Advisor 跨服務預防混淆代理人

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多權限的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了避免這種情況， AWS 提供工具，協助您保護所有 服務的資料，其服務主體已獲得您帳戶中資源的存取權。

若要限制 Device Advisor 為資源提供另一項服務的許可，我們推薦在資源政策中使用 aws:SourceArn 和 aws:SourceAccount 全域條件內容索引鍵。如果同時使用全域條件內容金鑰，則在相同政策陳述式中使用 aws:SourceAccount 值和 aws:SourceArn 值中的帳戶時，必須使用相同的帳戶 ID。

aws:SourceArn 值必須是套件定義資源的 ARN。套件定義資源是指使用 Device Advisor 建立的測試套件。

防範混淆代理人問題最有效的方法，是使用 aws:SourceArn 全域條件內容金鑰，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 aws:SourceArn 全域條件內容金鑰，同時使用萬用字元 (*) 表示 ARN 的未知部分。例如 arn:aws:iotdeviceadvisor:*:*account-id*:suitedefinition/*

下列範例示範如何使用 Device Advisor 中的 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，來預防混淆代理人問題。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "ConfusedDeputyPreventionExamplePolicy",
        "Effect": "Allow",
        "Principal": {
            "Service": "iotdeviceadvisor.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iotdeviceadvisor:us-
east-1:123456789012:suitedefinition/ygp6rxa3tzvn"
        },
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
        }
        }
    }
}
```

AWS 訓練和認證

進行下列課程，以了解 AWS IoT 安全的關鍵概念：AWS IoT Security Primer。

監控 AWS IoT

監控是維護 和 AWS 解決方案的可靠性、可用性 AWS IoT 和效能的重要部分。

我們強烈建議您從 AWS 解決方案的所有部分收集監控資料，以便在發生多點故障時更輕鬆地偵錯。請在一開始先建立可回答下列問題的監視計劃。如果您不確定如何回答這些問題，您仍然可以繼續啟用記錄並建立效能基準。

- 監控目標是什麼？
- 監控哪些資源？
- 監控這些資源的頻率為何？
- 將使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

您的下一個步驟是啟用記錄，並在不同的負載條件下測量各種時間 AWS IoT 的效能，以建立環境中正常效能的基準。當您監控時 AWS IoT，請保留歷史監控資料，以便與目前的效能資料進行比較。這可協助您識別正常效能模式和效能異常情況，並策劃解決這些情況的方法。

若要建立 的基準效能 AWS IoT，您應該監控這些指標以開始。您可以隨時監控更多度量。

- PublishIn.Success
- PublishOut.Success
- Subscribe.Success
- Ping.Success
- Connect.Success
- GetThingShadow.Accepted
- UpdateThingShadow.Accepted
- DeleteThingShadow.Accepted
- RulesExecuted

本節中的主題可協助您開始記錄和監視 AWS IoT。

主題

- 設定 AWS IoT 記錄
- 使用 Amazon CloudWatch 監控 AWS IoT 警示和指標
- AWS IoT 使用 CloudWatch Logs 進行監控
- 將裝置端日誌上傳到 Amazon CloudWatch
- 使用 記錄 AWS IoT API 呼叫 AWS CloudTrail

設定 AWS IoT 記錄

您必須使用 AWS IoT 主控台、CLI 或 API 啟用記錄，才能監控和記錄 AWS IoT 活動。您可以在 AWS IoT 三個層級設定 的記錄：帳戶層級、事件層級或資源特定層級。事件層級和資源特定的記錄僅適用於 V2 記錄。使用 V1 記錄的客戶必須執行 V2 遷移，才能存取這些功能。請參閱詳細資訊。

考慮如何設定 AWS IoT 記錄時，帳戶層級記錄組態會決定記錄 AWS IoT 活動的方式，除非另有說明。從一開始，您可能想要取得預設日誌層級為 INFO 或 DEBUG 的詳細日誌。檢閱初始日誌之後，您可以將預設日誌層級變更為較不詳細的層級，例如帳戶或事件層級的 WARN 或 ERROR，並針對可能需要更多注意的資源設定更詳細的資源特定日誌層級。您可以隨時更改日誌層級。

本主題涵蓋雲端登入 AWS IoT。如需裝置端日誌和監控的相關資訊，請參閱將裝置端日誌上傳至 CloudWatch。

如需記錄和監控的資訊 AWS IoT Greengrass，請參閱 中的記錄和監控 AWS IoT Greengrass。

設定 V2 登入 AWS IoT

決定您的記錄版本

如果未啟用 V2GetV2LoggingOptions API 會傳回 NotConfiguredException。當 V1 記錄正在使用，或未設定記錄時，就會發生此錯誤。

了解 V2 記錄功能

V2 記錄提供兩個關鍵功能：事件層級記錄和資源特定的記錄。事件層級記錄可透過可自訂的日誌層級和 CloudWatch 日誌群組目的地來啟用目標記錄組態。資源特定的記錄可讓您依物件群組、來源 IP、用戶端 ID 或主體 ID 篩選日誌。這些功能共同提供 IoT 操作的精細控制和全面的可見性，透過消除不必要的記錄活動來改善日誌可搜尋性並降低成本。

從 V1 移轉至 V2

您可以透過 AWS CLI 或 SDK，使用 SetV2LoggingOptions API 遷移至 V2 記錄。SetV2LoggingOptions 遷移後， AWS IoT 會自動將所有日誌路由到 CloudWatch 日誌群組 'AWSIotLogsV2'。重要：如果您的下游應用程式或資源使用來自「AWSIotLogs」的資訊，請更新它們以使用對應的日誌群組路徑。

設定記錄角色和政策

您必須先建立 IAM 角色和政策，授予代表您將 AWS IoT 日誌活動寫入 CloudWatch 日誌群組的 AWS IoT 許可 AWS IoT，才能啟用登入。您也可以使用 AWS IoT 主控台的日誌區段中所需的政策來產生 IAM 角色。

> **Note**
>
> 啟用 AWS IoT 記錄之前，請確定您了解 CloudWatch Logs 存取許可。具有 CloudWatch Logs 存取權限的使用者可從您的裝置查看除錯資訊。如需更多資訊，請參閱 Amazon CloudWatch Logs 的驗證與存取控制。
> 如果您 AWS IoT Core 預期 因為負載測試而有高流量模式，請考慮停用 IoT 記錄，以防止限流。若偵測到高流量，我們的服務可能會停用您的帳戶登入。

以下說明如何為 AWS IoT Core 資源建立記錄角色和政策。

建立記錄角色

若要建立記錄角色，請開啟 IAM 主控台的角色中樞，然後選擇 Create role (建立角色)。

1. 在 Select trusted entity (選取信任的實體) 下，選取 AWS service (服務)。然後在 Use case (使用案例) 下選擇 IoT。如果未顯示 IoT，請從 Use cases for other AWS services: (其他 服務的使用案例：) 下拉式清單中輸入並搜尋 IoT。選取下一步。
2. 在 Add permissions (新增許可) 頁面上，您會看到自動連接到服務角色的政策。選擇下一步。
3. 在 Name, review, and create (命名、檢閱和建立) 頁面上，輸入角色的 Role name (角色名稱) 和 Role description (角色描述)，然後選擇 Create role (建立角色)。

記錄角色政策

下列政策文件提供了角色政策及信任政策，用以允許 AWS IoT 代表您提交日誌項目至 CloudWatch。如果您使用自訂 CloudWatch 日誌群組設定事件層級記錄，則必須更新角色政策以包含自訂資源 ARN。

如果您也允許 AWS IoT Core for LoRaWAN 提交日誌項目，您會看到為您建立的政策文件，可記錄這兩個活動。

> **Note**
>
> 這些文件是在您建立記錄角色時為您建立的。文件具有變數 *${partition}#${region}* 和 *${accountId}* ，您必須將其取代為值。
>
> - 將分割區取代為區域的分割區。
> - 將區域取代為您使用 AWS 的區域。請確定您使用與在裝置上設定 CLI AWS 相同的 AWS 區域。
> - 將 account-id 取代為 AWS 您的帳戶 ID。

角色政策：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "iot:GetLoggingOptions",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingOptions",
                "iot:GetV2LoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:ListV2LoggingLevels",
                "iot:DeleteV2LoggingLevel"
```

```
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:AWSIotLogsV2:*"
            ]
        }
    ]
}
```

僅記錄 AWS IoT Core 活動的信任政策：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

AWS IoT 由於 IAM 角色許可不足，記錄可能無法將記錄發佈至 CloudWatch Logs。發生這種情況時，請檢查 CloudWatch 記錄指標以調查故障並進行故障診斷。

在 AWS IoT （主控台）中設定記錄

本節說明如何使用 AWS IoT 主控台設定 AWS IoT 記錄。您可以設定帳戶層級、事件層級和資源特定的記錄。

若要設定 AWS IoT 記錄：

1. 登入 AWS IoT 主控台。如需詳細資訊，請參閱開啟 AWS IoT 主控台。
2. 在左側導覽窗格中，選擇日誌（先前是設定下的區段）。
3. 設定帳戶層級記錄：帳戶層級記錄會套用至所有 AWS IoT 機群（裝置或端點），除非被事件層級或資源特定設定覆寫。

a. 在帳戶層級記錄下，選取管理帳戶層級記錄以進行更新。

b. 選取「啟用記錄」核取方塊，以開始將日誌傳送至 CloudWatch。取消勾選「啟用記錄」時，AWS IoT 不會將任何日誌傳送至 CloudWatch 日誌群組，無論事件層級或資源層級記錄組態為何。

c. 在 IAM 日誌角色下，從下拉式清單中選取現有角色。您可以檢視角色詳細資訊以檢查角色許可。或者，選取建立新角色以設定新的 IAM 角色。記錄角色提供允許 代表您 AWS IoT 將日誌項目提交至 CloudWatch 的政策。如果您使用自訂 CloudWatch 日誌群組設定事件層級記錄，則必須更新角色政策，以包含此日誌群組的 ARN。

d. 選擇對應至您要顯示在 CloudWatch Logs 中日誌項目詳細資訊層級的預設日誌層級。注意：日誌層級「DEBUG」提供最詳細的資訊，但會增加 CloudWatch 成本。無法在帳戶層級設定 CloudWatch 日誌群組目的地。不過，您可以為個別事件類型指定自訂日誌群組，如下節所述。

e. 選擇更新記錄以儲存您的變更。

4. 事件層級記錄可讓您選擇性地擷取相關事件的日誌，並將其導向至專用的 CloudWatch 日誌群組。這可讓您靈活地根據使用案例組織日誌，以獲得更好的可探索性、與不同的對象共用日誌，以及啟用日誌並根據事件重要性設定日誌層級，以降低 CloudWatch 成本。

   設定事件層級記錄：事件層級記錄會擷取特定 AWS IoT 事件，例如用戶端身分驗證嘗試。這些設定會覆寫帳戶層級記錄。

   a. 在事件層級記錄區段下，選取管理事件層級記錄以進行更新。

   b. 根據預設，事件類型會繼承帳戶層級的記錄組態。注意：設定資源特定的記錄時，它會覆寫帳戶和事件層級設定。

   c. 若要修改個別事件的設定，請按一下對應事件列中的值。您可以同時調整日誌層級和 CloudWatch 日誌群組目的地。指定自訂 CloudWatch 日誌群組目的地時，您必須驗證 IAM 角色政策是否包含新日誌群組的許可。未更新角色政策將 AWS IoT 阻止 將日誌寫入自訂日誌群組。進行選擇後，請按一下核取記號以確認您的選擇。「已修改」欄會顯示「是」，指出待定的變更。

   d. 按一下更新記錄以套用您的變更，或選擇取消以捨棄。

5. 設定資源特定覆寫：資源特定覆寫會將記錄設定套用至選取的資源。資源可以是物件群組、來源 IP、用戶端 ID 或主體 ID。資源特定的記錄組態會覆寫帳戶層級和事件層級設定。啟用時，它會在指定資源的已設定記錄層級產生所有事件類型的日誌。例如，您可以為特定物件設定偵錯層級記錄，同時保留所有其他物件的資訊層級記錄。

   a. 在資源特定覆寫區段中選取新增資源特定覆寫。

b. 選擇日誌目標：物件群組、來源 IP、用戶端 ID 或主體 ID。
c. 輸入所選目標類型的對應日誌目標值。
d. 從資源特定日誌層級區段的下拉式功能表中選取所需的日誌層級。
e. 按一下提交以新增覆寫，或按一下取消以捨棄變更。
f. 若要修改現有的資源特定覆寫，請選取資源旁的核取方塊，然後按一下「移除」來刪除覆寫，或按一下「編輯」來修改。

啟用記錄之後，請造訪 在 CloudWatch 主控台中檢視 AWS IoT 日誌 以進一步了解檢視記錄項目。

在 AWS IoT (CLI) 中設定帳戶和事件層級登入

本節說明如何使用 AWS IoT CLI 設定 的全域記錄。

您可以選擇性地設定事件層級記錄。事件層級記錄會在事件層級擷取記錄資訊，例如身分驗證和授權或憑證建立事件。您可以在事件層級自訂日誌層級和 CloudWatch 日誌群組目的地。相較於帳戶層級記錄，事件層級記錄在更目標層級運作，因此會覆寫帳戶層級記錄設定。這種階層方法可讓您根據不同類型的事件的操作重要性和成本考量來維護不同的記錄策略。

> **Note**
>
> 您需要您想要使用的角色的 Amazon Resource Name (ARN)。如果您需要建立用於記錄的角色，請在繼續前參閱 建立記錄角色。為任何事件類型指定自訂 CloudWatch 日誌群組時，請確定您的記錄角色具有目標日誌群組的必要許可。
> 用於進行 API 呼叫的委託人必須擁有您記錄角色的 傳遞角色許可。

您也可以使用 API 中對應至此處所示 CLI 命令的方法來使用 AWS API 執行此程序。

使用 CLI 設定 的預設記錄 AWS IoT

1. 請使用 set-v2-logging-options 命令來設定您的帳戶的記錄選項。

```
aws iot set-v2-logging-options \
    --event-configurations event-configuration-list \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

其中：

--role-arn

角色 ARN，授予在 CloudWatch Logs 中寫入日誌的 AWS IoT 許可。初始設定需要 Role-arn 組態。

--default-log-level

欲使用的日誌層級。有效值為：ERROR、WARN、DEBUG、 INFO或 DISABLED。初始設定需要 Default-log-level組態。

--no-disable-all-logs

啟用所有 AWS IoT 記錄的選用參數。使用此參數可在目前該參數停用時啟用記錄功能。

--disable-all-logs

選用參數，可停用所有 AWS IoT 記錄。使用此參數可在該參數目前啟用時停用記錄功能。

--event-configurations

此參數是選用的，可讓您自訂個別事件類型的記錄設定：

- eventType：覆寫帳戶層級設定的事件類型
- logLevel：使用 DEBUG、INFO、 ERROR、WARN 或 DISABLED 覆寫帳戶層級設定
- logDestination：指定用於日誌交付的自訂 CloudWatch 日誌群組

您可以為每個事件類型分別設定記錄層級和日誌目的地。如果未指定，事件將繼承帳戶層級設定

```
aws iot set-v2-logging-options \
    --event-configurations "[{\"eventType\":\"Publish-In\",\"logLevel\":\"INFO\",
\"logDestination\":\"examplePublishInLogGroup\"}]"
```

2. 使用 get-v2-logging-options 命令取得目前的記錄選項。

```
aws iot get-v2-logging-options \
    --verbose
```

其中：

--verbose

選用參數，可讓您擷取所有 eventTypes 及其組態。

啟用記錄之後，請造訪 在 CloudWatch 主控台中檢視 AWS IoT 日誌 以進一步了解檢視記錄項目。

> **Note**
>
> AWS IoT 繼續支援較舊的命令 (set-logging-options 和 get-logging-options) 來設定和取得您帳戶的全域記錄。請注意，這些指令在使用時所產生的記錄包含純文字，而非 JSON 有效承載，且記錄延遲通常會更高。這些較舊指令的實施將不會進一步改進。我們建議您使用「v2」版本來設定記錄選項，並盡可能變更使用舊版的舊版應用程式。

在 AWS IoT (CLI) 中設定資源特定覆寫

本節說明如何 AWS IoT 使用 CLI 設定 的資源特定覆寫。資源特定覆寫可讓您為物件群組、用戶端 ID、來源 IP 或主體 ID 識別的特定資源指定記錄層級。啟用資源特定記錄時，它會覆寫帳戶層級和事件層級設定。所有事件類型都會在設定的日誌層級為指定的資源產生日誌，即使這些事件在事件層級組態中已停用。

物件群組可以包含其他物件群組來建立階層關係。此程序說明如何設定單一物件群組的記錄。您可以將此程序套用至階層中的父項物件群組，以設定階層中所有物件群組的記錄。您也可以將此過程應用於子事件組，以覆蓋其父項的記錄組態。

物件可以是物件群組的成員。此成員資格允許物件繼承套用至物件群組的組態、政策和設定。物件群組用於管理和套用設定到多個物件，而不是個別處理每個物件。當您的用戶端 ID 符合物件名稱時，AWS IoT Core 會自動將用戶端工作階段與對應的物件資源建立關聯。這可讓用戶端工作階段繼承套用至物件所屬物件群組的組態和設定，包括記錄層級。如果您的用戶端 ID 與物件名稱不相符，您可以啟用獨佔物件附件來建立關聯。如需詳細資訊，請參閱 ???。

除物件群組之外，您還可以記錄目標，如裝置的用戶端 ID、來源 IP 和主體 ID。

> **Note**
>
> 您需要您想要使用的角色的 Amazon Resource Name (ARN)。如果您需要建立用於記錄的角色，請在繼續前參閱 建立記錄角色。
> 用於進行 API 呼叫的委託人必須擁有您記錄角色的 傳遞角色許可。

您也可以使用 API 中對應至此處所示 CLI 命令的方法來使用 AWS API 執行此程序。

使用 CLI 設定 的資源特定覆寫 AWS IoT

1. 使用下列命令設定資源特定記錄之前，請先啟用帳戶層級記錄：aws iot set-v2-logging-options 命令
2. 使用 set-v2-logging-level命令來設定資源特定的覆寫。如需物件群組組態，請參閱下列範例：

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

--log-target

您設定欲記錄的資源類型與名稱。targetType 值必須是以下其中之一：THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。log-target 參數值可以是文字，如前面的命令範例所示，也可以是 JSON 字串，例如下列範例。

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
              --log-level log_level
```

--log-level

為指定資源產生日誌時所用的日誌層級。有效值為：DEBUG、INFO、ERROR、WARN 和 DISABLED。

3. 使用 list-v2-logging-levels 命令列出目前設定的日誌層級。

```
aws iot list-v2-logging-levels
```

4. 使用 delete-v2-logging-level 命令可刪除特定資源的日誌層級，如下列範例所示：

```
aws iot delete-v2-logging-level \
              --target-type "THING_GROUP" \
              --target-name "thing_group_name"
```

```
aws iot delete-v2-logging-level \
              --target-type=CLIENT_ID
              --target-name=ClientId1
```

--target-type

target-type 值必須是以下其中之一：THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。

--target-name

要移除日誌層級之物件群組的名稱。

日誌層級

這些日誌層級會決定記錄的事件，並套用至預設和資源特定的日誌層級。

ERROR

導致操作失敗的錯誤。

範例：由於憑證過期而無法驗證裝置。

日誌內僅含 ERROR 資訊。

WARN

會導致系統內出現不一致，但可能不會造成操作失敗的情況。

範例：接近訊息速率限制。

日誌內僅含 ERROR 和 WARN 資訊。

INFO

物件流的高層級資訊。

範例：用戶端已成功訂閱 MQTT 主題。

日誌包含 INFO、ERROR、WARN 資訊。

DEBUG

可能有助於偵錯問題的資訊。

範例：IoT 規則引擎偵測到發佈至規則主題「規則/測試」的訊息，並成功開始執行。規則是使用 RepublishAction 設定。

日誌包含 DEBUG、INFO、ERROR、WARN 資訊。

DISABLED

已停用所有記錄功能。

使用 Amazon CloudWatch 監控 AWS IoT 警示和指標

您可以使用 AWS IoT CloudWatch 監控 ，該 CloudWatch 會收集原始資料並將其處理 AWS IoT 為可讀且近乎即時的指標。這些統計資料會保存兩週的期間，以便您存取歷史資訊，並更清楚 web 應用程式或服務的執行方式。依預設， AWS IoT 指標資料會以一分鐘的間隔自動傳送至 CloudWatch。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch、Amazon CloudWatch Events 及 Amazon CloudWatch Logs？。

使用 AWS IoT 指標

報告的指標 AWS IoT 提供您可以用不同方式分析的資訊。以下使用案例是根據案例，您有十個項目，每天連接到網際網路。每個天：

- 十個物件大約同時連接到 AWS IoT 。
- 每個真的訂閱主題篩選，然後等待一小時才能中斷。在此期間，事與其他應用程式通訊和進一步了解世界各地的狀態。
- 每個動作會根據其新發現有一些感知發佈資料 `UpdateThingShadow`。
- 每個物件都會中斷連線 AWS IoT。

為了協助您開始使用，這些主題會探索可能遇到的一些問題。

- 如何每天都在我的物件連線失敗時收到通知？
- 如何每天都在我的物件並未推送資料時收到通知？
- 如何每天都在我的物件的影子更新遭拒時收到通知？
- 如何為任務建立 CloudWatch 警示？

CloudWatch 警示和指標的詳細資訊

- 建立 CloudWatch 警示以監控 AWS IoT
- AWS IoT 指標和維度

建立 CloudWatch 警示以監控 AWS IoT

您可以建立 CloudWatch 警報，在警示變更狀態時傳送 Amazon SNS 訊息。警示會在您指定的期間，監看單一指標。當指標值在數個期間內超過指定的閾值時，會執行一或多個動作。此動作可為傳送至 Amazon SNS 主題或 Auto Scaling 政策的通知。警示僅會觸發持續狀態變更的動作。CloudWatch 警示不會只因處於特定狀態就觸發動作，狀態必須已遭變更且已維持了一段指定的時間。

下列主題說明使用 CloudWatch 警示的一些範例。

- 如何每天都在我的物件連線失敗時收到通知？
- 如何每天都在我的物件並未推送資料時收到通知？
- 如何每天都在我的物件的影子更新遭拒時收到通知？
- 如何為任務建立 CloudWatch 警示？

您可以在 中查看 CloudWatch AWS IoT 指標和維度 警示可以監控的所有指標。

如何每天都在我的物件連線失敗時收到通知？

1. 建立名為 `things-not-connecting-successfully` 的 Amazon SNS 主題，並記錄其 Amazon 資源名稱 (ARN)。此程序會將您主題的 ARN 稱為 *`sns-topic-arn`* 。

   如需如何建立 Amazon SNS 通知的詳細資訊，請參閱 Amazon SNS 入門。

2. 建立警示。

```
aws cloudwatch put-metric-alarm \
    --alarm-name ConnectSuccessAlarm \
    --alarm-description "Alarm when my Things don't connect successfully" \
    --namespace AWS/IoT \
    --metric-name Connect.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 測試警示。

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. 確認警示顯示於 CloudWatch 主控台中。

如何每天都在我的物件並未推送資料時收到通知？

1. 建立名為 `things-not-publishing-data` 的 Amazon SNS 主題，並記錄其 Amazon 資源名稱 (ARN)。此程序會將您主題的 ARN 稱為 *`sns-topic-arn`* 。

   如需如何建立 Amazon SNS 通知的詳細資訊，請參閱 Amazon SNS 入門。

2. 建立警示。

```
aws cloudwatch put-metric-alarm \
    --alarm-name PublishInSuccessAlarm\
    --alarm-description "Alarm when my Things don't publish their data \
    --namespace AWS/IoT \
    --metric-name PublishIn.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 測試警示。

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. 確認警示顯示於 CloudWatch 主控台中。

如何每天都在我的物件的影子更新遭拒時收到通知？

1. 建立名為 `things-shadow-updates-rejected` 的 Amazon SNS 主題，並記錄其 Amazon 資源名稱 (ARN)。此程序會將您主題的 ARN 稱為 *`sns-topic-arn`* 。

   如需如何建立 Amazon SNS 通知的詳細資訊，請參閱 Amazon SNS 入門。

2. 建立警示。

```
aws cloudwatch put-metric-alarm \
    --alarm-name UpdateThingShadowSuccessAlarm \
    --alarm-description "Alarm when my Things Shadow updates are getting rejected"
 \
    --namespace AWS/IoT \
    --metric-name UpdateThingShadow.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 測試警示。

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value ALARM
```

4. 確認警示顯示於 CloudWatch 主控台中。

如何為任務建立 CloudWatch 警示？

任務服務提供 CloudWatch 指標，以供您監控任務。您可以建立 CloudWatch 警示來監控任何 任務指標 。

下列命令會建立 CloudWatch 警示，以監控任務 *`SampleOTAJob`* 的任務執行失敗次數總計，並會在超過 20 個任務執行失敗時，通知您。警示每 300 秒會檢查報告的值，以監控任務指標

FailedJobExecutionTotalCount。當單一報告值大於 20 時，便會啟動警示，表示自任務啟動以來，失敗的任務執行次數超過 20 次。當警示關閉時，會傳送通知給所提供的 Amazon SNS 主題。

```
aws cloudwatch put-metric-alarm \
    --alarm-name TotalFailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when total number of failed job execution exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionTotalCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 20 \
    --comparison-operator GreaterThanThreshold \
    --period 300 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions
```

下列命令會建立 CloudWatch 警示，以監控在指定期間內任務 *SampleOTAJob* 失敗的任務執行次數。然後，當在此期間有超過五個任務執行失敗時，會通知您。警示每 3600 秒會檢查報告的值，以監控任務指標 FailedJobExecutionCount。當單一報告值大於 5 時，便會啟動警示，表示在過去一小時內，失敗的任務執行次數超過 5 次。當警示關閉時，會傳送通知給所提供的 Amazon SNS 主題。

```
aws cloudwatch put-metric-alarm \
    --alarm-name FailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when number of failed job execution per hour exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --period 3600 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions-per-hour
```

AWS IoT 指標和維度

當您與 互動時 AWS IoT，服務每分鐘都會將指標和維度傳送至 CloudWatch。您可以使用 AWS IoT、使用 CloudWatch 主控台或 AWS CLI 來檢視這些指標。

若要使用 CloudWatch 主控台檢視指標，請開啟 CloudWatch 主控台。在導覽窗格中，選擇 Metrics (指標)，然後選擇 All metrics (所有指標)。在瀏覽索引標籤中，搜尋 AWS IoT 以檢視指標清單。指標會先依服務命名空間分組，再依各命名空間內不同的維度組合分類。

若要使用 檢視指標 AWS CLI，請執行下列命令。

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch 會顯示下列指標群組 AWS IoT：

- AWS IoT 指標
- AWS IoT Core 登入資料提供者指標
- 身分驗證指標
- 伺服器憑證 OCSP 堆疊指標
- 規則指標
- 規則動作指標
- HTTP 動作特定指標
- 訊息代理程式指標
- Device Shadow 指標
- 記錄指標
- 任務指標
- Device Defender Audit 指標
- Device Defender Detect 指標
- 裝置佈建指標
- LoRaWAN 指標
- 機群索引指標
- 指標的維度

AWS IoT 指標

指標	說明
AddThingToDynamicThingGroupsFailed	與將物件新增至動態物件群組相關聯的失敗物件數量。此 DynamicThingGroupName 維度包含無法新增項目的動態群組名稱。
NumLogBatchesFailedToPublishThrottled	因調節錯誤而無法發佈的單批日誌事件。
NumLogEventsFailedToPublishThrottled	批次中因調節錯誤而無法發佈的日誌事件數。

AWS IoT Core 登入資料提供者指標

指標	說明
CredentialExchangeSuccess	對 AWS IoT Core 憑證提供者提出成功的 AssumeRoleWithCertificate 請求數。

身分驗證指標

> **Note**
>
> 身分驗證指標會顯示在 CloudWatch 主控台的通訊協定指標下。

指標	說明
Connection.AuthNError	由於身分驗證失敗而 AWS IoT Core 拒絕的連線嘗試次數。此指標只會考慮傳送與 端點相符之伺服器名稱指示 (SNI) 字串的連線 AWS 帳戶。此指標包括來自外部來源的連線嘗試，例如網際網路掃描工具或探測

指標	說明
	活動。Protocol 維度包含用於傳送連線嘗試的通訊協定。

伺服器憑證 OCSP 堆疊指標

指標	說明
RetrieveOCSPStapleData.Success	OCSP 回應已成功接收和處理。此回應將包含在已設定網域的 TLS 交握期間。DomainConfiguratio nName 維度包含已啟用伺服器憑證 OCSP 堆疊的已設定網域名稱。

規則指標

指標	說明
ParseError	JSON 剖析錯誤數目，發生於對規則所接聽主題發佈的訊息中。RuleName 維度會包含規則的名稱。
RuleExecutionThrottled	規則引擎因惡意行為而調節的訊息數量，例如過度的基本擷取會將請求發佈至不存在的規則。RuleName 維度包含欲觸發規則的名稱。
RuleEvaluation (Status=T hrottled)	已調節的規則評估數目。Status 維度指定請求的狀態。目前，唯一支援的狀態為 Throttled ，表示請求已節流。
RuleNotFound	找不到欲觸發的規則。RuleName 維度會包含規則的名稱。
RulesExecuted	執行的 AWS IoT 規則數目。
TopicMatch	對規則所接聽主題發佈的傳入訊息數目。RuleName 維度會包含規則的名稱。

規則動作指標

指標	說明
Failure	失敗規則動作呼叫次數。RuleName 維度包含可指定動作之規則的名稱。ActionType 維度包含已呼叫動作的類型。
Success	成功規則動作呼叫次數。RuleName 維度包含可指定動作之規則的名稱。ActionType 維度包含已呼叫動作的類型。
ErrorActionFailure	失敗的錯誤動作數量。RuleName 維度包含可指定動作之規則的名稱。ActionType 維度包含已呼叫動作的類型。
ErrorActionSuccess	成功的錯誤動作數量。RuleName 維度包含可指定動作之規則的名稱。ActionType 維度包含已呼叫動作的類型。

HTTP 動作特定指標

指標	描述
HttpCode_Other	如果來自下游 Web 服務/應用程式的回應狀態碼不是 2xx、4xx 或 5xx，則會產生此指標。
HttpCode_4XX	如果來自下游 Web 服務/應用程式的回應狀態碼介於 400 到 499 之間，則會產生此指標。
HttpCode_5XX	如果來自下游 Web 服務/應用程式的回應狀態碼介於 500 到 599 之間，則會產生此指標。
HttpInvalidUrl	在取代替換範本之後，如果端點 URL 未以 https:// 開頭，則會產生此指標。

指標	描述
HttpRequestTimeout	如果下游 Web 服務/應用程式未在請求逾時限制內傳回回應，則會產生此指標。如需詳細資訊，請參閱 Service Quotas。
HttpUnknownHost	如果 URL 有效，但服務不存在或無法連線，則會產生此指標。

訊息代理程式指標

> **Note**
>
> 訊息代理程式指標會顯示於 CloudWatch 主控台的 Protocol Metrics (通訊協定指標) 之下。

指標	說明
Connect.AuthError	訊息代理程式無法授權的連線要求數目。Protocol 維度包含用來傳送 CONNECT 訊息的協定。
Connect.ClientError	由於 MQTT 訊息不符合 AWS IoT 配額 中定義的要求，連線請求已拒絕的數量。Protocol 維度包含用來傳送 CONNECT 訊息的協定。
Connect.ClientIDThrottle	由於用戶端超過特定用戶端 ID 所允許之連線要求率而調節的連線要求數目。Protocol 維度包含用來傳送 CONNECT 訊息的協定。
Connect.ServerError	因發生內部錯誤而失敗的連線要求數目。Protocol 維度包含用來傳送 CONNECT 訊息的協定。
Connect.Success	訊息代理程式成功的連線次數。Protocol 維度包含用來傳送 CONNECT 訊息的協定。

指標	說明
Connect.Throttle	因帳戶超過允許連線要求率而調節的連線要求數目。Protocol 維度包含用來傳送 CONNECT 訊息的協定。
Ping.Success	訊息代理程式收到的 ping 訊息數目。Protocol 維度包含用來傳送 ping 訊息的協定。
PublishIn.AuthError	訊息代理程式無法授權的發佈要求數目。Protocol 維度包含用來發佈訊息的協定。HTTP 發佈不支援此指標。
PublishIn.ClientError	訊息中介裝置拒絕的發佈請求數量，因為訊息不符合 AWS IoT 配額 中定義的要求。Protocol 維度包含用來發佈訊息的協定。
PublishIn.ServerError	訊息代理程式因發生內部錯誤而無法處理的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishIn.Success	訊息代理程式成功處理的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishIn.Throttle	因用戶端超過允許傳入訊息率而調節的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。HTTP 發佈不支援此指標。
PublishOut.AuthError	AWS IoT無法授權之訊息代理程式所提出的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishOut.ClientError	訊息中介裝置提出的發佈請求數量遭到拒絕，因為訊息不符合 AWS IoT 配額 中定義的要求。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishOut.ServerError	訊息代理程式因發生內部錯誤而無法處理的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。

指標	說明
PublishOut.Success	訊息代理程式成功發出的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishOut.Throttle	因用戶端超過所允許之傳出訊息率而調節的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishRetained.AuthError	訊息代理程式無法授權帶 RETAIN 旗標集的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishRetained.ServerError	訊息代理程式因發生內部錯誤而無法處理的保留發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishRetained.Success	訊息代理程式成功處理帶 RETAIN 旗標集的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
PublishRetained.Throttle	因用戶端超過允許傳入訊息率而調節的帶 RETAIN 旗標集的發佈要求數目。Protocol 維度包含用來傳送 PUBLISH 訊息的協定。
Queued.Success	訊息代理程式與其中斷持久性工作階段連線的用戶端成功處理的儲存訊息數目。具有持久性工作階段的用戶端中斷連線時，會儲存 QoS 為 1 的訊息。
Queued.Throttle	具有持久性工作階段的用戶端中斷連線時，無法儲存和限流的訊息數目。當用戶端超過每個帳戶每秒的佇列訊息數限制時，就會發生這種情況。具有持久性工作階段的用戶端中斷連線時，會儲存 QoS 為 1 的訊息。
Queued.ServerError	因為內部錯誤而未針對持久性工作階段儲存的訊息數目。當具有持久性工作階段的用戶端中斷連線時，會儲存服務品質 (QoS) 為 1 的訊息。

指標	說明
Subscribe.AuthError	由用戶端提出且無法授權的訂閱要求數目。Protocol 維度包含用來傳送 SUBSCRIBE 訊息的協定。
Subscribe.ClientError	由於SUBSCRIBE 訊息不符合 AWS IoT 配額 中定義的要求而拒絕的訂閱請求數量。Protocol 維度包含用來傳送 SUBSCRIBE 訊息的協定。
Subscribe.ServerError	因發生內部錯誤而拒絕的訂閱要求數目。Protocol 維度包含用來傳送 SUBSCRIBE 訊息的協定。
Subscribe.Success	訊息代理程式成功處理的訂閱要求數目。Protocol 維度包含用來傳送 SUBSCRIBE 訊息的協定。
Subscribe.Throttle	由於您的 超過允許的訂閱請求率限制而調節的訂閱請求數量 AWS 帳戶。這些限制包括每個帳戶的每秒訂閱數、每個帳戶的訂閱數，以及AWS IoT Core 訊息中介裝置和通訊協定限制和配額中所述的每個連線訂閱數。Protocol 維度包含用來傳送 SUBSCRIBE 訊息的協定。
Throttle.Exceeded	當 MQTT 用戶端在每個連線層級限制上每秒封包受到調節時，此指標會顯示在 CloudWatch 中。此指標不適用於 HTTP 連線。
Unsubscribe.ClientError	因為UNSUBSCRIBE 訊息不符合 中定義的要求而拒絕的取消訂閱請求數量AWS IoT 配額。Protocol 維度包含用來傳送 UNSUBSCRIBE 訊息的協定。
Unsubscribe.ServerError	因發生內部錯誤而拒絕的取消訂閱要求數目。Protocol 維度包含用來傳送 UNSUBSCRIBE 訊息的協定。
Unsubscribe.Success	訊息代理程式成功處理的取消訂閱要求數目。Protocol 維度包含用來傳送 UNSUBSCRIBE 訊息的協定。

指標	說明
Unsubscribe.Throttle	因用戶端超過允許取消訂閱要求率而拒絕的取消訂閱要求數目。Protocol 維度包含用來傳送 UNSUBSCRIBE 訊息的協定。

Device Shadow 指標

> **Note**
>
> 裝置影子指標會顯示在 CloudWatch 主控台的 Protocol Metrics (通訊協定指標) 之下。

指標	說明
DeleteThingShadow.Accepted	成功處理的 DeleteThingShadow 要求數目。Protocol 維度包含用來提出要求的協定。
GetThingShadow.Accepted	成功處理的 GetThingShadow 要求數目。Protocol 維度包含用來提出要求的協定。
ListThingShadow.Accepted	成功處理的 ListThingShadow 要求數目。Protocol 維度包含用來提出要求的協定。
UpdateThingShadow.Accepted	成功處理的 UpdateThingShadow 要求數目。Protocol 維度包含用來提出要求的協定。

記錄指標

指標	說明
CloudwatchLogs:LogGroupCreationFailed	CreateLogGroup 請求失敗的數量。如果不存在，AWS IoT Logging 會建立 CloudWatch 日誌群組。LogGroup 維度包含用於提出請求的日誌群組。

指標	說明
CloudwatchLogs:LogStreamCreationFailed	CreateLogStream 請求失敗的數量。AWS IoT Logging 會建立 CloudWatch 日誌串流來推送日誌事件。LogGroup 維度包含用於提出請求的日誌群組。
CloudwatchLogs:PutRetentionPolicyFailed	PutRetentionPolicy 請求失敗的數量。如果不存在，AWS IoT Logging 會建立 CloudWatch 日誌群組，並將保留政策設定為 30 天。LogGroup 維度包含用於提出請求的日誌群組。
CloudwatchLogs:PutLogEventsFailed	PutLogEvents 請求失敗的數量。LogGroup 維度包含用於提出請求的 CloudWatch 日誌群組。

任務指標

指標	說明
CanceledJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 CANCELED 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
CanceledJobExecutionTotalCount	針對指定任務，狀態為 CANCELED 的任務執行總數。JobId 維度包含任務的 ID。
ClientErrorCount	執行任務時產生的用戶端錯誤數目。JobId 維度包含任務的 ID。
FailedJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 FAILED 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
FailedJobExecutionTotalCount	針對指定任務，狀態為 FAILED 的任務執行總數。JobId 維度包含任務的 ID。

指標	說明
InProgressJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 IN_PROGRESS 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
InProgressJobExecutionTotalCount	針對指定任務，狀態為 IN_PROGRESS 的任務執行總數。JobId 維度包含任務的 ID。
RejectedJobExecutionTotalCount	針對指定任務，狀態為 REJECTED 的任務執行總數。JobId 維度包含任務的 ID。
RemovedJobExecutionTotalCount	針對指定任務，狀態為 REMOVED 的任務執行總數。JobId 維度包含任務的 ID。
QueuedJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 QUEUED 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
QueuedJobExecutionTotalCount	針對指定任務，狀態為 QUEUED 的任務執行總數。JobId 維度包含任務的 ID。
RejectedJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 REJECTED 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
RemovedJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 REMOVED 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
ServerErrorCount	執行任務時產生的伺服器錯誤數目。JobId 維度包含任務的 ID。

指標	說明
SucceededJobExecutionCount	其狀態已在 CloudWatch 決定的期間內變更為 SUCCESS 之任務執行的數量。(如需有關 CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 指標。) JobId 維度包含任務的 ID。
SucceededJobExecutionTotalCount	針對指定任務，狀態為 SUCCESS 的任務執行總數。JobId 維度包含任務的 ID。

Device Defender Audit 指標

指標	說明
NonCompliantResources	在檢查中發現不相容的資源數量。在每次稽核執行的檢查時，系統都會回報不合規的資源數量。
ResourcesEvaluated	評估符合的資源數目。系統會為每次執行的稽核檢查報告評估的資源數量。
MisconfiguredDeviceDefenderNotification	當您的 SNS AWS IoT Device Defender 組態設定錯誤時， 會通知您。 Dimensions (尺寸)

Device Defender Detect 指標

指標	說明
NumOfMetricsExported	針對雲端、裝置端或自訂指標匯出的指標數量。系統會針對特定指標報告為帳戶匯出的指標數量。此指標僅適用於使用指標匯出的客戶。
NumOfMetricsSkipped	針對雲端、裝置端或自訂指標略過的指標數量。由於提供給 Device Defender Detect 發佈至 mqtt 主題的許

指標	說明
	可不足，系統針對特定指標報告為帳戶略過的指標數量。此指標僅適用於使用指標匯出的客戶。
NumOfMetricsExceedingSizeLimit	由於大小超過 MQTT 訊息大小限制，略過匯出雲端、裝置端或自訂指標的指標數量。系統會報告因為大小超過 MQTT 訊息大小限制，而針對特定指標略過匯出的指標數量。此指標僅適用於使用指標匯出的客戶。
Violations	自從上次執行評估起，發現新違反安全性描述檔行為的數目。系統會回報帳戶、特定安全性設定檔，以及特定安全性設定檔特定行為的新違規次數。
ViolationsCleared	自上次評估執行以來已解決的安全性設定檔行為的違規次數。系統會報告已解決的帳戶、特定安全性設定檔以及特定安全性設定檔特定行為的違規次數。
ViolationsInvalidated	自上次執行評估起，資訊不再可用的安全性描述檔行為違反之數目 (因為報告裝置停止報告，或因為某些原因不再受到監控)。系統會報告整個帳戶，特定安全性設定檔以及特定安全性設定檔特定行為的無效違規次數。
MisconfiguredDeviceDefenderNotification	當您的 SNS AWS IoT Device Defender 組態設定錯誤時， 會通知您。 Dimensions (尺寸)

裝置佈建指標

AWS IoT 機群佈建指標

指標	說明
ApproximateNumberOfThingsRegistered	已由機群佈建註冊的事項計數。 雖然計數通常相當準確，但 AWS IoT Core 的分配式架構將會很難保持已註冊事項的精確計數。

指標	說明
	此指標使用的統計數字為： • Max (最大值)，報告已註冊的事項總數。如需在 CloudWatch 彙總視窗期間註冊的事項計數，請參閱 `RegisterThingFailed` 指標。 維度：ClaimCertificateId
`CreateKeysAndCertificateFailed`	呼叫 `CreateKeysAndCertificate` MQTT API 時發生的失敗次數。 在成功 (值 = 0) 和失敗 (值 = 1) 兩種情況下，皆會發出指標。此指標可用來追蹤於 CloudWatch 支援的彙總視窗期間 (例如 5 分鐘或 1 小時) 建立及註冊的憑證數量。 此指標可用的統計數字為： • Sum (總和)，報告失敗的呼叫數量。 • SampleCount，報告成功和失敗的呼叫總數。
`CreateCertificateFromCsrFailed`	呼叫 `CreateCertificateFromCsr` MQTT API 時發生的失敗次數。 在成功 (值 = 0) 和失敗 (值 = 1) 兩種情況下，皆會發出指標。此指標可用來追蹤於 CloudWatch 支援的彙總視窗期間 (例如 5 分鐘或 1 小時) 註冊的事項數量。 此指標可用的統計數字為： • Sum (總和)，報告失敗的呼叫數量。 • SampleCount，報告成功和失敗的呼叫總數。

指標	說明
RegisterThingFailed	呼叫 RegisterThing MQTT API 時發生的失敗次數。 在成功 (值 = 0) 和失敗 (值 = 1) 兩種情況下，皆會發出指標。此指標可用來追蹤於 CloudWatch 支援的彙總視窗期間 (例如 5 分鐘或 1 小時) 註冊的事項數量。如需註冊事項的總數，請參閱 ApproximateNumberOfThingsRegistered 指標。 此指標可用的統計數字為： • Sum (總和)，報告失敗的呼叫數量。 • SampleCount，報告成功和失敗的呼叫總數。 維度：TemplateName

即時佈建指標

指標	說明
ProvisionThing.ClientError	裝置因用戶端錯誤而無法佈建的次數。例如，指定於範本中的政策不存在。
ProvisionThing.ServerError	裝置因伺服器錯誤而無法佈建的次數。客戶可在等待後重試佈建裝置，而若問題仍然存在，其可聯絡 AWS IoT 。
ProvisionThing.Success	成功佈建的裝置的次數。

LoRaWAN 指標

下表顯示 的指標 AWS IoT Core for LoRaWAN。如需詳細資訊，請參閱 AWS IoT Core for LoRaWAN 指標。

AWS IoT Core for LoRaWAN 指標

指標	說明
作用中裝置/閘道	您帳戶中作用中 LoRaWAN 裝置和閘道的數量。
上行訊息計數	您 中所有作用中閘道和裝置在指定期間內傳送的上行訊息數量 AWS 帳戶。上行訊息是從您的裝置傳送到的訊息 AWS IoT Core for LoRaWAN。
下行訊息計數	您 中所有作用中閘道和裝置在指定期間內傳送的下行訊息數量 AWS 帳戶。下行訊息是從 傳送到 AWS IoT Core for LoRaWAN 您的裝置的訊息。
訊息損失率	新增裝置並連線至 後 AWS IoT Core for LoRaWAN，您的裝置可以啟動上行訊息，以開始與雲端交換訊息。接著您可以使用此指標來追蹤上行訊息損失率。
聯結指標	新增裝置和閘道之後，您可以執行聯結程序，讓裝置可以傳送上行資料並與之通訊 AWS IoT Core for LoRaWAN。您可以使用此指標取得您 AWS 帳戶中所有作用中裝置的聯結指標相關資訊。
平均接收訊號強度指示器 (RSSI)	您可以使用此指標來監控一段指定期間內的 RSSI (接收的訊號強度指示器) 平均數。RSSI 是一種測量方式，會指出訊號強度是否足以提供良好的無線連線。此值為負值，對於強式連線，必須更接近零。
平均訊號雜訊比 (SNR)	您可以使用此指標來監控一段指定期間內的 SNR (訊號雜訊比) 平均數。SNR 是一種測量方式，會指出與雜訊程度相比，收到的訊號強度是否足以提供良好的無線連線。SNR 值為正數，且必須大於零，以表示訊號功率比雜訊功率強。
閘道可用性	您可以使用此指標來取得一段指定期間內，此閘道的可用性相關資訊。此指標會顯示一段指定期間內，此閘道的 Websocket 連線時間。

即時佈建指標

指標	說明
ProvisionThing.ClientError	裝置因用戶端錯誤而無法佈建的次數。例如，指定於範本中的政策不存在。
ProvisionThing.ServerError	裝置因伺服器錯誤而無法佈建的次數。客戶可在等待後重試佈建裝置，而若問題仍然存在，其可聯絡 AWS IoT 。
ProvisionThing.Success	成功佈建的裝置的次數。

機群索引指標

AWS IoT 機群索引指標

指標	說明
NamedShadowCountForDynamicGroupQueryLimitExceeded	對於動態物件群組中並非特定於資料來源的查詢術語，每個物件最多處理 25 個已命名影子。當物件違反此限制時，會發出 NamedShadowCountForDynamicGroupQueryLimitExceeded 事件類型。

指標的維度

指標使用命名空間，並提供下列維度的指標。

維度	說明
ActionType	觸發請求，並由規則指定的動作類型。
BehaviorName	被監控的 Device Defender Detect 安全性設定檔行為名稱。
ClaimCertificateId	用來佈建裝置的 certificateId 宣告。

維度	說明
CheckName	其結果正受到監控的 Device Defender 稽核檢查的名稱。
JobId	正在監控其進度或訊息連線成功/失敗的任務 ID。
Protocol	用來提出要求的協定。有效值為：MQTT 或 HTTP
RuleName	要求所觸發規則的名稱。
ScheduledAuditName	其檢查結果正受到監控之 Device Defender 排程稽核的名稱。若回報的結果是依隨需執行的稽核結果，則此值為 OnDemand。
SecurityProfileName	其行為正受到監控之 Device Defender Detect 安全性描述檔的名稱。
TemplateName	佈建範本的名稱。
SourceArn	參考用於偵測的安全設定檔或用於稽核的帳戶。
RoleArn	是指 Device Defender 嘗試擔任的角色。
TopicArn	參考嘗試發佈至 的 SNS 主題 Device Defender。

維度	說明
`Error`	提供嘗試發佈至 SNS 主題時所收到錯誤的簡短描述。可能值為： • "KMSKeyNotFound"：表示主題不存在 KMS 金鑰。 • "InvalidTopicName"： 表示 SNS 主題無效。 • "KMSAccessDenied"：表示角色沒有主題 KMS 金鑰的許可。 • 「AuthorizationError」：表示提供的角色並未授權 Device Defender 發佈至 SNS 主題。 • 「SNSTopicNotFound」：表示提供的 SNS 主題不存在。 • 「FailureToAssumeRole」：表示提供的角色並未授權 Device Defender 擔任該角色。 • 「CrossRegionSNSTopic」：表示 SNS 主題存在於不同的區域中。

AWS IoT 使用 CloudWatch Logs 進行監控

AWS IoT 啟用記錄功能時， 會透過訊息代理程式和規則引擎，在訊息從您的裝置傳遞時 AWS IoT 傳送每個訊息的進度事件。在 CloudWatch 主控台中，CloudWatch 日誌會顯示於名為 AWSIotLogs 的日誌群組中。

如需 CloudWatch Logs 的詳細資訊，請參閱 CloudWatch Logs。如需 Support AWS IoT CloudWatch Logs 的相關資訊，請參閱 CloudWatch Logs AWS IoT 日誌項目 。

在 CloudWatch 主控台中檢視 AWS IoT 日誌

> **Note**
>
> `AWSIotLogsV2` 日誌群組於下列狀況下才會顯示於 CloudWatch 主控台中：
>
> • 您已啟用登入 AWS IoT。如需如何啟用登入的詳細資訊 AWS IoT，請參閱 設定 AWS IoT 記錄

- 有些日誌項目已由 AWS IoT 操作寫入。

在 CloudWatch 主控台中檢視您的 AWS IoT 日誌

1. 請瀏覽 https://console.aws.amazon.com/cloudwatch/。在導覽窗格中，選擇 Log groups (日誌群組)。
2. 在 Filter (篩選條件) 文字方塊中，輸入 **`AWSIotLogsV2`**，然後按下 Enter 鍵。
3. 請按兩下 `AWSIotLogsV2` 日誌群組。
4. 選擇 Search All (全部搜尋)。系統會顯示為您的帳戶產生的 AWS IoT 日誌完整清單。
5. 選擇展開圖示可觀看個別的串流。

您也可以在 Filter events (篩選條件事件) 中輸入一個查詢。以下是一些可以試試看的有趣查詢：

- `{ $.logLevel = "INFO" }`

  尋找日誌層級為 `INFO` 的所有日誌。
- `{ $.status = "Success" }`

  尋找日誌狀態為 `Success` 的所有日誌。
- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  尋找狀態為 `Success` 且事件類型為 `GetThingShadow` 的所有日誌。

如需有關建立篩選條件表達式的詳細資訊，請參閱 CloudWatch Logs 查詢。

CloudWatch Logs AWS IoT 日誌項目

的每個元件都會 AWS IoT 產生自己的日誌項目。每個日誌項目都有 `eventType` 指定造成日誌項目產生的作業。本節旨在說明下列 AWS IoT 元件所產生的記錄項目。

主題

- Message broker 日誌項目
- 伺服器憑證 OCSP 日誌項目
- Device Shadow 日誌項目

- Rules engine 日誌項目
- 任務日誌項目
- 裝置佈建日誌項目
- 動態物件群組日誌項目
- 機群索引日誌項目
- 常用 CloudWatch Logs 屬性

Message broker 日誌項目

AWS IoT 訊息中介裝置會產生下列事件的日誌項目：

主題

- Connect 記錄項目
- Disconnect 日誌輸入項
- Ping 日誌項目
- Connection.AuthNError 日誌項目
- DeleteConnection 日誌項目
- GetRetainedMessage 日誌項目
- ListRetainedMessage 日誌項目
- ListSubscriptions 日誌項目
- GetConnection 日誌項目
- Publish-In 日誌項目
- Publish-Out 日誌項目
- 佇列日誌項目
- 訂閱日誌項目
- 取消訂閱日誌項目
- SendDirectMessage 日誌項目

Connect 記錄項目

當 MQTT 用戶端連線Connect時， AWS IoT 訊息中介裝置會使用 eventType 的 產生日誌項目。

連線成功日誌項目範例

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.11",
    "sourcePort": 13490
}
```

連線失敗日誌項目範例

```
{
    "timestamp": "2026-07-09 23:26:48.827",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.1",
    "sourcePort": 11238,
    "reason": "throttled",
    "details": "Exceeded Connect requests per second per account limit"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Connect日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

reason

Connect 操作失敗的原因。只有在 status為 時，才會出現此欄位Failure。有效值為 throttled、client_id_throttled 或 INTERNAL_SERVER_ERROR。

詳細資訊

錯誤的簡要說明。只有在 status為 時，才會出現此欄位Failure。

原因代碼

下列原因代碼和詳細資訊會出現在Connect失敗日誌項目中。

Reason	詳細資訊
throttled	超過每個帳戶每秒的 Connect 請求數限制
client_id_throttled	超過每個用戶端 ID 限制每秒的 Connect 請求數

Disconnect 日誌輸入項

當 MQTT 用戶端中斷連線Disconnect時， AWS IoT 訊息中介裝置會使用 eventType 的 產生日誌項目。

Disconnect 日誌項目的連線範例

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
```

```
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID",
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Disconnect日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

reason

用戶端中斷連線的原因。

詳細資訊

錯誤的簡要說明。

disconnectReason

用戶端中斷連線的原因。

Ping 日誌項目

AWS IoT 當訊息中介裝置從連線用戶端處理 MQTT PINGREQ 並傳送對應的 PINGRESP Ping時，會產生具有 eventType 的 日誌項目。每個 Ping 操作都會發出單一日誌項目。

> **Important**
>
> Ping 日誌項目為選擇加入，預設為停用。發射閾值符合每個項目的結果：成功項目會在DEBUG關卡發出，失敗項目則會在ERROR關卡發出。因此，每當套用至連線的事件層級或資源特定覆寫設定為 ERROR或更詳細層級 (WARN、 或 DEBUG) 時，您就會看到 Ping 失敗項目INFO，但只有在該層級設定為 時才會發佈 Ping 成功項目DEBUG。若要設定這些設定，請參閱 在 AWS IoT (CLI) 中設定帳戶和事件層級登入和 在 AWS IoT (CLI) 中設定資源特定覆寫。

成功時， logLevel是 DEBUG，而 status是 Success。失敗時 （ 例如，用戶端連線在傳送 PINGRESP 之前終止時）， logLevel是 ERROR，而 status是 Failure。

Ping 日誌項目範例 （ 成功 ）

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "DEBUG",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Ping",
    "protocol": "MQTT",
    "latency": 128,
    "requestTimestamp": "2025-10-31 15:37:23.348",
    "responseTimestamp": "2025-10-31 15:37:23.476",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

Ping 日誌項目範例 （ 失敗 ）

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "ERROR",
```

```
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Ping",
    "protocol": "MQTT",
    "requestTimestamp": "2025-10-31 15:37:23.476",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "CONNECTION_ALREADY_CLOSED",
    "details": "Connection has already closed"
}
```


除了 常用 CloudWatch Logs 屬性，Ping 記錄項目包含下列屬性：

protocol

用來提出要求的協定。值一律為 MQTT。

延遲

從接收 PINGREQ 到傳送 PINGRESP 之間的時間，以毫秒為單位。只有在 status為 時，才會出現此欄位Success。

requestTimestamp

從用戶端 AWS IoT Core 接收 PINGREQ 時的時間戳記。

responseTimestamp

將 PINGRESP AWS IoT Core 傳送至用戶端時的時間戳記。只有在 status為 時，才會出現此欄位Success。

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

reason

Ping 操作失敗的原因。只有在 `status`為 時，才會出現此欄位`Failure`。

詳細資訊

錯誤的簡要說明。只有在 `status`為 時，才會出現此欄位`Failure`。

Connection.AuthNError 日誌項目

當用戶端連線嘗試因身分驗證失敗而遭到拒絕`Connection.AuthNError`時， AWS IoT 訊息中介裝置會產生具有 `eventType` 的日誌項目。此日誌項目`logLevel`的 一律為 `ERROR`，而 一律`status`為 `Failure`。此日誌項目對應至 `Connection.AuthNError` CloudWatch 指標。

> **Important**
>
> `Connection.AuthNError` 日誌項目為選擇加入，預設為停用。它們會在`ERROR`日誌層級發出，因此只有在套用至連線的事件層級或資源特定覆寫設定為 `ERROR`或更詳細的層級 (`WARN`、或) 時`INFO`，才會發佈`Connection.AuthNError`日誌項目`DEBUG`。若要設定這些設定，請參閱 在 AWS IoT (CLI) 中設定帳戶和事件層級登入和 在 AWS IoT (CLI) 中設定資源特定覆寫。

> **Note**
>
> 此日誌項目會發佈到 AWS 帳戶 擁有 TLS 交握中提供的伺服器名稱指示 (SNI) 的 。如果用戶端未提供 SNI，或 SNI 無法解析至帳戶，則不會發出日誌項目或指標。

Connection.AuthNError 日誌項目範例

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
```

```
    "status": "Failure",
    "eventType": "Connection.AuthNError",
    "protocol": "MQTT",
    "authenticationType": "AWS_X509",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "targetIp": "52.94.236.10",
    "targetPort": 8883,
    "serverNameIndication": "a1b2c3d4e5f6g7-ats.iot.us-east-1.amazonaws.com",
    "reason": "DEVICE_CERTIFICATE_NOT_REGISTERED",
    "details": "Device certificate is not registered"
}
```


除了 常用 CloudWatch Logs 屬性，Connection.AuthNError 記錄項目包含下列屬性：

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

authenticationType

用於連線嘗試的身分驗證類型。有效值為 AWS_X509、AWS_SIGV4、CUSTOM_AUTH、CUSTOM_AUTH_X509。

clientId

提出請求的用戶端 ID，如果有的話。

principalId

提出請求的委託人 ID，如果有的話。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

targetIp

連線的目的地 IP 地址。此欄位可能不會出現在所有日誌項目中。

targetPort

連線的目的地連接埠。

serverNameIndication

用戶端提供的 TLS 伺服器名稱指示 (SNI) 延伸主機名稱。

reason

用戶端身分驗證失敗的原因。the section called “Connection.AuthNError 錯誤代碼” 如需可能值的清單，請參閱 。

詳細資訊

身分驗證失敗的簡短描述。

Connection.AuthNError 錯誤代碼

下表列出可用於Connection.AuthNError日誌項目的reason錯誤代碼和對應的details訊息。未來可能會新增其他錯誤代碼。

錯誤碼	錯誤訊息
CUSTOM_AUTHORIZER_ LAMBDA_EXECUTION_ERROR	自訂授權方 Lambda 調用失敗
CUSTOM_AUTHORIZER_ NOT_FOUND	找不到自訂授權方
CUSTOM_AUTHORIZER_ PARAMETER_INVALID	自訂身分驗證參數無效
DEVICE_CERTIFICATE _INACTIVE	裝置憑證非作用中
DEVICE_CERTIFICATE _NOT_REGISTERED	裝置憑證未註冊
DEVICE_CERTIFICATE _REVOKED	裝置憑證已撤銷

錯誤碼	錯誤訊息
DOMAIN_CONFIGURATION_DISABLED	提供的伺服器名稱指示 (SNI) 的網域組態已停用
DOMAIN_CONFIGURATION_INVALID	網域組態中的端點類型無效
INTERNAL_SERVER_ERROR	內部伺服器錯誤
NOT_AUTHENTICATED	提供的登入資料無法驗證
SECURITY_TOKEN_EXPIRED	安全性字符已過期
SECURITY_TOKEN_INVALID	請求中包含的安全字符無效
SECURITY_TOKEN_SIGNATURE_MISMATCH	請求簽章不符合提供的簽章
SNI_MISUSED	用戶端使用未與此帳戶關聯的 SNI

DeleteConnection 日誌項目

訊息 AWS IoT 中介裝置會在刪除 MQTT 用戶端連線`DeleteConnection`時，使用 `eventType` 的 產生日誌項目。

DeleteConnection 日誌項目範例

```
{
    "timestamp": "2025-08-09 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteConnection",
    "protocol": "HTTP",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

除了 常用 CloudWatch Logs 屬性 之外，DeleteConnection日誌項目還包含下列屬性：

clientId

將中斷連線的用戶端 ID。

principalId

提出請求的委託人 ID。如需有關如何使用委託人 ID 識別委託人的資訊，請參閱《IAM 使用者指南》中的比較 IAM 身分和登入資料。

protocol

用來提出要求的協定。有效值為 HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

GetRetainedMessage 日誌項目

AWS IoT 訊息中介裝置會在GetRetainedMessage呼叫 GetRetainedMessage時產生具有 eventType 的日誌項目。

GetRetainedMessage 日誌項目範例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetRetainedMessage",
    "protocol": "HTTP",
    "topicName": "a/b/c",
    "qos": "1",
    "lastModifiedDate": "2017-08-07 18:47:56.664"
}
```

除了 常用 CloudWatch Logs 屬性 之外，GetRetainedMessage日誌項目還包含下列屬性：

lastModifiedDate

儲存保留訊息的 Epoch 日期和時間，以毫秒為單位 AWS IoT。

protocol

用來提出要求的協定。有效值：HTTP。

qos

發佈請求中使用的服務品質 (QoS) 等級。有效值為 0 或 1。

topicName

訂閱主題的名稱。

ListRetainedMessage 日誌項目

AWS IoT 訊息中介裝置會在ListRetainedMessages呼叫 ListRetainedMessage時產生具有 eventType 的日誌項目。

ListRetainedMessage 日誌項目範例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ListRetainedMessage",
    "protocol": "HTTP"
}
```

除了 常用 CloudWatch Logs 屬性 之外，ListRetainedMessage日誌項目還包含下列屬性：

protocol

用來提出要求的協定。有效值：HTTP。

ListSubscriptions 日誌項目

AWS IoT 訊息中介裝置會在收到列出用戶端訂閱的請求ListSubscriptions時，使用 eventType 的 產生日誌項目。

ListSubscriptions 日誌項目範例

```
{
    "timestamp": "2025-08-22 19:57:46.587",
    "logLevel": "INFO",
    "traceId": "5bea658a-0752-bf20-770f-ad17c8dabb95",
    "accountId": "987654321",
    "status": "Success",
    "eventType": "ListSubscriptions",
    "protocol": "HTTP",
    "clientId": "abc",
    "principalId": "AIDAZPOK3C3545MCMYATF",
    "sourceIp": "52.94.133.137",
    "sourcePort": 13490,
    "maxResults": 100
}
```

除了 常用 CloudWatch Logs 屬性，ListSubscriptions 記錄項目包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值：HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

maxResults

請求的結果數量上限。

GetConnection 日誌項目

訊息 AWS IoT 中介裝置會在收到取得連線資訊的請求`GetConnection`時，使用 `eventType` 的 產生日誌項目。

GetConnection 日誌項目範例

```
{
    "timestamp": "2025-08-22 19:19:50.679",
    "logLevel": "INFO",
    "traceId": "7a04de1a-d1a3-50bc-3ca5-96b1ee293dae",
    "accountId": "987654321",
    "status": "Success",
    "eventType": "GetConnection",
    "protocol": "HTTP",
    "clientId": "aClient",
    "principalId": "AIDAZPOK3C3545MCMYATF",
    "sourceIp": "52.94.133.137",
    "sourcePort": 31983
}
```

除了 常用 CloudWatch Logs 屬性，`GetConnection` 記錄項目包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值：`HTTP`。

sourceIp

API 請求產生所在的 IP 地址。

sourcePort

API 請求產生所在的連接埠。

Publish-In 日誌項目

當 AWS IoT 訊息中介裝置收到 MQTT 訊息時，它會產生具有 之 eventType的日誌項目Publish-In。

發佈成功日誌項目範例

```
{
        "timestamp": "2017-08-10 15:39:30.961",
        "logLevel": "INFO",
        "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "accountId": "111122223333",
        "status": "Success",
        "eventType": "Publish-In",
        "protocol": "MQTT",
        "topicName": "$aws/things/MyThing/shadow/get",
        "clientId": "abf27092886e49a8a5c1922749736453",
        "principalId":
 "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
        "sourceIp": "203.0.113.12",
        "sourcePort": 13490,
        "retain": "True"
    }
```

發佈失敗日誌項目範例

```
{
    "timestamp": "2026-07-09 22:01:12.858",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Publish-In",
    "protocol": "MQTT",
    "topicName": "testing/topic",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.4",
    "sourcePort": 10872,
    "reason": "throttled",
    "details": "Exceeded Inbound publish requests per second per account limit"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Publish-In日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

保留

當「保留」旗標訊息的值設定為 True 時所使用的屬性。如果訊息沒有設定「保留」旗標，則此屬性不會顯示在日誌條目中。如需詳細資訊，請參閱 MQTT 保留訊息。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

topicName

訂閱主題的名稱。

reason

Publish-In 操作失敗的原因。只有在 status為 時，才會出現此欄位Failure。有效值為 throttled、connection_throttled、CLIENT_ERROR 或 INTERNAL_SERVER_ERROR。

詳細資訊

錯誤的簡要說明。只有在 status為 時，才會出現此欄位Failure。

原因代碼

下列原因代碼和詳細資訊會出現在Publish-In失敗日誌項目中。

Reason	詳細資訊
throttled	超過每個帳戶每秒的傳入發佈請求數限制

Reason	詳細資訊
throttled	超過每個帳戶每秒的傳出發佈請求數限制
throttled	超過每個帳戶限制每秒的規則評估
throttled	超過每個帳戶每秒的保留訊息傳入發佈請求限制
throttled	超過每個主題每秒的保留訊息傳入發佈請求限制
throttled	超過每個帳戶限制的保留訊息數量上限
connection_throttled	超過每個連線限制每秒的發佈請求數
CLIENT_ERROR	超過傳入未確認的 QoS 1 發佈請求上限

Publish-Out 日誌項目

當訊息代理程式發佈 MQTT 訊息時，會產生 eventType 為 Publish-Out 的日誌項目。

發佈成功日誌項目範例

```
{
    "timestamp": "2017-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/get",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.13",
    "sourcePort": 13490
}
```

發佈失敗日誌項目範例

```
{
    "timestamp": "2026-07-14 21:12:33.026",
```

```
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "testing/topic",
    "clientId": "subscriberClientThrottled",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "connection_throttled",
    "details": "Exceeded Publish requests per second per connection limit"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Publish-Out日誌項目還包含下列屬性：

clientId

接收有關該 MQTT 主題之訊息的訂閱用戶端的 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

topicName

訂閱主題的名稱。

reason

Publish-Out 操作失敗的原因。只有在 status為 時，才會出現此欄位Failure。有效值為 throttled、connection_throttled、CLIENT_ERROR 或 INTERNAL_SERVER_ERROR。

詳細資訊

錯誤的簡要說明。只有在 status為 時，才會出現此欄位Failure。

原因代碼

下列原因代碼和詳細資訊會出現在Publish-Out失敗日誌項目中。

Reason	詳細資訊
throttled	超過每個帳戶每秒的傳出發佈請求數限制
connection_throttled	超過每個連線限制每秒的發佈請求數
connection_throttled	超過傳出未確認的 QoS 1 發佈請求上限
CLIENT_ERROR	用戶端端點不可寫入
CLIENT_ERROR	用戶端未連線

佇列日誌項目

當具有持久性工作階段的裝置中斷連線時，MQTT 訊息代理程式會儲存裝置的訊息，並 AWS IoT 產生 eventType 為 的日誌項目Queued。如需 MQTT 持久性工作階段的詳細資訊，請參閱 MQTT 持久性工作階段 。

佇列的伺服器錯誤日誌項目範例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "details": "Server Error"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Queued伺服器錯誤日誌項目還包含下列屬性：

clientId

訊息排入佇列的用戶端 ID。

詳細資訊

Server Error

伺服器錯誤會造成訊息無法儲存。

protocol

用來提出要求的協定。這個值將永遠為 MQTT。

qos

請求的服務品質 (QoS) 等級。該值將始終為 1，因為具有 QoS 為 0 的訊息不會儲存。

topicName

訂閱主題的名稱。

佇列成功日誌項目範例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Success"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Queued成功日誌項目還包含下列屬性：

clientId

訊息排入佇列的用戶端 ID。

protocol

用來提出要求的協定。這個值將永遠為 MQTT。

qos

請求的服務品質 (QoS) 等級。該值將始終為 1，因為具有 QoS 為 0 的訊息不會儲存。

topicName

訂閱主題的名稱。

佇列限流日誌項目範例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "reason": "connection_throttled",
    "details": "Throttled while queueing offline message"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Queued限流日誌項目還包含下列屬性：

clientId

訊息排入佇列的用戶端 ID。

詳細資訊

錯誤的簡要說明。當 status為 時Failure，值為 Throttled while queueing offline message。

protocol

用來提出要求的協定。這個值將永遠為 MQTT。

qos

請求的服務品質 (QoS) 等級。該值將始終為 1，因為具有 QoS 為 0 的訊息不會儲存。

reason

connection_throttled

用戶端超出 Publish requests per second per connection 限制，因此訊息不會儲存。

throttled

帳戶超過 Queued messages per second per account 限制，因此訊息不會儲存。

topicName

訂閱主題的名稱。

訂閱日誌項目

當 MQTT 用戶端訂閱主題Subscribe時， AWS IoT 訊息中介裝置會使用 eventType的 產生日誌項目。

MQTT 3 訂閱成功日誌項目範例

```
{
    "timestamp": "2017-08-10 15:39:04.413",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/#",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.14",
    "sourcePort": 13490
}
```

訂閱失敗日誌項目範例

```
{
    "timestamp": "2026-07-09 23:07:45.633",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
```

```
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "subscribe/test",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.2",
    "sourcePort": 10494,
    "reason": "throttled",
    "details": "Exceeded Subscriptions per second per account limit"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Subscribe日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

protocol

用來提出要求的協定。這個值將永遠為 MQTT。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

topicName

訂閱主題的名稱。

reason

Subscribe 操作失敗的原因。只有在 status為 時，才會出現此欄位Failure。有效值為 throttled、CLIENT_ERROR 或 INTERNAL_SERVER_ERROR。

詳細資訊

錯誤的簡要說明。只有在 status為 時，才會出現此欄位Failure。

MQTT 5 訂閱日誌項目範例

```
{
 "timestamp": "2022-11-30 16:24:15.628",
 "logLevel": "INFO",
 "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
 "accountId": "111122223333",
 "status": "Success",
 "eventType": "Subscribe",
 "protocol": "MQTT",
 "topicName": "test/topic1,$invalid/reserved/topic",
 "subscriptions": [
  {
   "topicName": "test/topic1",
   "reasonCode": 1
  },
  {
   "topicName": "$invalid/reserved/topic",
   "reasonCode": 143
  }
 ],
 "clientId": "abf27092886e49a8a5c1922749736453",
 "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
 "sourceIp": "203.0.113.15",
 "sourcePort": 13490
}
```

對於 MQTT 5 訂閱操作，除了 常用 CloudWatch Logs 屬性和 訂閱日誌項目屬性之外，MQTT 5 Subscribe日誌項目還包含下列屬性：

訂閱

「訂閱」請求中所要求主題與個別 MQTT 5 原因代碼之間的對應清單。如需更多詳細資訊，請參閱 MQTT 原因代碼。

原因代碼

下列原因代碼和詳細資訊會出現在Subscribe失敗日誌項目中。

Reason	詳細資訊
throttled	超過每個帳戶每秒訂閱數限制
throttled	超過每個帳戶限制的訂閱數
throttled	超過每個連線限制的訂閱數
throttled	超過每個群組的共用訂閱限制
throttled	超過每個群組每秒的訂閱和取消訂閱請求限制
CLIENT_ERROR	訂閱封包會重複使用來自先前未確認請求的封包 ID
CLIENT_ERROR	用戶端未連線
CLIENT_ERROR	超過每個訂閱請求上限
CLIENT_ERROR	無效的主題篩選條件

取消訂閱日誌項目

當 MQTT 用戶端取消訂閱 MQTT 主題Unsubscribe時，AWS IoT 訊息中介裝置會使用 eventType 的 產生日誌項目。

取消訂閱成功日誌項目範例

```
{
    "timestamp": "2024-08-20 22:53:32.844",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Unsubscribe",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.16",
    "sourcePort": 13490
}
```

取消訂閱失敗日誌項目範例

```
{
    "timestamp": "2026-01-07 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Unsubscribe",
    "protocol": "MQTT",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.3",
    "sourcePort": 26051,
    "reason": "throttled",
    "details": "Exceeded Unsubscribe requests per second per account limit"
}
```

除了 常用 CloudWatch Logs 屬性 之外，Unsubscribe日誌項目還包含下列屬性：

protocol

用來提出要求的協定。這個值將永遠為 MQTT。

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

reason

Unsubscribe 操作失敗的原因。只有在 status為 時，才會出現此欄位Failure。有效值為 throttled、CLIENT_ERROR 或 INTERNAL_SERVER_ERROR。

詳細資訊

錯誤的簡要說明。只有在 `status`為 時，才會出現此欄位`Failure`。

原因代碼

下列原因代碼和詳細資訊會出現在`Unsubscribe`失敗日誌項目中。

Reason	詳細資訊
`throttled`	超過每個帳戶每秒的取消訂閱請求數限制
`throttled`	超過每個群組每秒的訂閱和取消訂閱請求數限制
`CLIENT_ERROR`	取消訂閱封包會重複使用來自先前未確認請求的封包 ID
`CLIENT_ERROR`	用戶端未連線

SendDirectMessage 日誌項目

AWS IoT 訊息中介裝置會在發出 SendDirectMessage API 請求`SendDirectMessage`時，使用 `eventType` 的 產生日誌項目。

AWS IoT Core 當您啟用 IoT 記錄時， 會將 SendDirectMessage 事件記錄到 CloudWatch Logs。如需設定記錄的詳細資訊，請參閱 設定 AWS IoT 記錄。

SendDirectMessage 成功日誌項目範例

```
{
    "timestamp": "2026-01-20 20:44:00.123",
    "logLevel": "INFO",
    "traceId": "1a2b3c4d-1234-3214-abcd-1a2b3c4d5e6f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "SendDirectMessage",
    "protocol": "HTTP",
    "topicName": "commands/reboot",
    "clientId": "mydevice-002",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
```

```
    "confirmation": true,
    "sourceIp": "203.0.113.45",
    "sourcePort": 52847,
    "timeout": 3
}
```

SendDirectMessage 失敗日誌項目範例

```
{
    "timestamp": "2026-01-15 20:30:00.123",
    "logLevel": "ERROR",
    "traceId": "1a2b3c4d-1234-5678-abcd-1a2b3c4d5e6f",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "SendDirectMessage",
    "protocol": "HTTP",
    "topicName": "commands/reboot",
    "clientId": "myDevice",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "confirmation": true,
    "sourceIp": "203.0.113.45",
    "sourcePort": 52847,
    "reason": "DELIVERY_CONFIRMATION_TIMEOUT",
    "details": "The delivery confirmation was not received within the specified timeout
 period",
    "timeout": 2
}
```

除了 常用 CloudWatch Logs 屬性，SendDirectMessage 記錄項目包含下列屬性：

clientId

接收直接訊息的用戶端 ID。

topicName

傳送訊息的主題。

protocol

用來提出要求的協定。有效值：HTTP。

確認

是否請求交付確認。有效值為 true 或 false。

timeout

等待來自接收用戶端的交付確認的時間上限，以秒為單位。只有在 confirmation為 時才會出現true。

reason

失敗的原因代碼。

詳細資訊

錯誤的簡要說明。

principalId

提出請求的委託人 ID。

sourceIp

請求來源所在的 IP 地址。

sourcePort

請求來源所在的埠口。

原因代碼

下列原因代碼和詳細資訊會出現在SendDirectMessage失敗日誌項目中。

Reason	詳細資訊
MALFORMED_CLIENTID	此用戶端 ID 無效。客戶 IDs 不得超過 128 個字元，且開頭不得為貨幣符號 ($)。輸入不同的用戶端 ID
INVALID_TOPIC	此主題無效。主題大小必須在 256 個位元組內，應包含少於 9 個部分，且不應以貨幣符號 ($) 開頭
INVALID_TIMEOUT_PARAMETER	無效的逾時值。必須是介於 1 到 15 之間的整數
INVALID_CONFIRMATION_PARAMETER	無效的確認值。必須是 'true' 或 'false'

Reason	詳細資訊
RECEIVE_AUTHORIZATION_FAILURE	目標用戶端的授權失敗。確認您的政策授予必要的許可
TARGET_CLIENT_ID_NOT_FOUND	找不到用戶端 ID。請確認用戶端已連線，然後再試一次
TARGET_CLIENT_NOT_CONNECTED	目標用戶端 ID 未連線，但具有作用中的持久性工作階段
SEND_DIRECT_MESSAGE_THROTTLED	由於超過傳送直接訊息請求率，請求遭拒
RECEIVE_SESSION_THROTTLED	請求遭拒，因為請求率超過每個連線每秒的傳出發佈請求
RECEIVER_MAX_UNACKED_QOS1_EXCEEDED	接收者用戶端已達到未確認的 QoS 1 訊息數量上限
DELIVERY_CONFIRMATION_TIMEOUT	未在指定的逾時期間內收到交付確認
DELIVERY_CONFIRMATION_SENDER_DISCONNECT	寄件者在收到交付確認之前中斷連線
CLIENT_ENDPOINT_NOT_WRITABLE	接收器用戶端端點不可寫入
INTERNAL_SERVER_ERROR	內部伺服器錯誤

伺服器憑證 OCSP 日誌項目

AWS IoT Core 會為下列事件產生日誌項目：

主題

- RetrieveOCSPStapleData 日誌項目
- 私有端點的 RetrieveOCSPStapleData 日誌項目

RetrieveOCSPStapleData 日誌項目

AWS IoT Core 當伺服器擷取 OCSP 基本資料`RetrieveOCSPStapleData`時，會產生具有 `eventType` 的 日誌項目。

RetrieveOCSPStapleData 日誌項目範例

以下是 的日誌項目範例`Success`。

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "200",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

以下是 的日誌項目範例`Failure`。

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "A non 2xx HTTP response was received from the OCSP responder.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "444",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```


對於 `RetrieveOCSPStaple`操作，除了 常用 CloudWatch Logs 屬性 之外，日誌項目還包含下列屬性：

reason

操作失敗的原因。

domainConfigName

網域組態的名稱。

connectionDetails

連線詳細資訊的簡短說明。

- httpStatusCode

  OCSP 回應程式傳回的 HTTP 狀態碼，以回應用戶端對伺服器提出的請求。

- ocspResponderUri

  從伺服器憑證擷取的 OCSP AWS IoT Core 回應程式 URI。

- sourceIp

AWS IoT Core 伺服器的來源 IP 地址。

- targetIp

  OCSP 回應程式的目標 IP 地址。

ocspRequestDetails

OCSP 請求的詳細資訊。

- requesterName

  將請求傳送至 OCSP 回應程式之 AWS IoT Core 伺服器的識別符。

- requestCertId

  請求的憑證 ID。這是請求 OCSP 回應的憑證 ID。

ocspResponseDetails

OCSP 回應的詳細資訊。

- responseCertId

  OCSP 回應的憑證 ID。

- ocspResponseStatus

  OCSP 回應的狀態。

- certStatus

  憑證的狀態。

- signature

  由信任實體套用至回應的簽章。

- thisUpdateTime

  指出狀態的時間已知正確。

- nextUpdateTime

  提供憑證狀態的更新資訊時或之前的時間。

- producedAtTime

  OCSP 回應者簽署此回應的時間。

- stapledDataPayloadSize

  主控資料的承載大小。

私有端點的 RetrieveOCSPStapleData 日誌項目

AWS IoT Core 當伺服器擷取 OCSP 基本資料`RetrieveOCSPStapleData`時，會產生具有 `eventType` 的 日誌項目。

私有端點的 RetrieveOCSPStapleData 日誌項目範例

以下是 的日誌項目範例`Success`。


```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responderId": "04:C1:3F:8F:27:D6:49:13:F8:DE:B2:36:9D:85:8E:F8:31:3B:A6:D0"
              "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
```

```
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

以下是 的日誌項目範例Failure。

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "The payload returned by the Lambda function exceeds the maximum response
 size of 7 kilobytes.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
  "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```

對於 RetrieveOCSPStaple操作，除了 RetrieveOCSPStapleData 日誌項目中的 常用 CloudWatch Logs 屬性和 屬性之外，私有端點的日誌項目還包含下列屬性：

lambdaDetails

Lambda 函數的詳細資訊。

- lambdaArn

  Lambda 函數的 ARN。

- sourceArn

網域組態的 ARN。

authorizedResponderArn

如果在網域組態中設定了授權方回應程式的 ARN。

Device Shadow 日誌項目

AWS IoT Device Shadow 服務會產生下列事件的日誌項目：

主題

- DeleteThingShadow 日誌項目
- GetThingShadow 日誌項目
- UpdateThingShadow 日誌項目

DeleteThingShadow 日誌項目

當收到刪除 Device Shadow 的要求時，Device Shadow 服務會產生 `eventType` 為 `DeleteThingShadow` 的日誌項目。

DeleteThingShadow 日誌項目範例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/delete"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`DeleteThingShadow`日誌項目還包含下列屬性：

deviceShadowName

要更新的影子名稱。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

請求所發布的主題名稱。

GetThingShadow 日誌項目

當收到影子的 get 要求時，Device Shadow 服務會產生 eventType 為 GetThingShadow 的日誌項目。

GetThingShadow 日誌項目範例

```
{
    "timestamp": "2017-08-09 17:56:30.941",
    "logLevel": "INFO",
    "traceId": "b575f19a-97a2-cf72-0ed0-c64a783a2504",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "MyThing",
    "topicName": "$aws/things/MyThing/shadow/get"
}
```

除了 常用 CloudWatch Logs 屬性 之外，GetThingShadow日誌項目還包含下列屬性：

deviceShadowName

所要求影子的名稱。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

請求所發布的主題名稱。

UpdateThingShadow 日誌項目

當收到更新 Device Shadow 的要求時，Device Shadow 服務會產生 `eventType` 為 `UpdateThingShadow` 的日誌項目。

UpdateThingShadow 日誌項目範例

```
{
    "timestamp": "2017-08-07 18:43:59.436",
    "logLevel": "INFO",
    "traceId": "d0074ba8-0c4b-a400-69df-76326d414c28",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/update"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`UpdateThingShadow`日誌項目還包含下列屬性：

deviceShadowName

要更新的影子名稱。

protocol

用來提出要求的協定。有效值為 `MQTT` 或 `HTTP`。

topicName

請求所發布的主題名稱。

Rules engine 日誌項目

AWS IoT 規則引擎會產生下列事件的日誌：

主題

- FunctionExecution 日誌項目
- RuleExecution 日誌項目
- RuleMatch 日誌項目

- RuleEvaluation 限流日誌項目
- RuleExecutionThrottled 日誌項目
- RuleNotFound 日誌項目
- StartingRuleExecution 日誌項目

FunctionExecution 日誌項目

當規則的 SQL 查詢呼叫外部函數時，規則引擎會產生 `eventType` 為 `FunctionExecution` 的日誌項目。當規則的動作對 AWS IoT 或其他 Web 服務提出 HTTP 請求 （例如呼叫 `get_thing_shadow`) 時，會呼叫外部 函數。

FunctionExecution 日誌項目範例

```
{
    "timestamp": "2017-07-13 18:33:51.903",
    "logLevel": "DEBUG",
    "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
    "status": "Success",
    "eventType": "FunctionExecution",
    "clientId": "N/A",
    "topicName":"rules/test",
    "ruleName": "ruleTestPredict",
    "ruleAction": "MachinelearningPredict",
    "resources": {
        "ModelId": "predict-model"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`FunctionExecution`日誌項目還包含下列屬性：

clientId

N/A 適用於 `FunctionExecution` 日誌。

principalId

提出請求的委託人 ID。

resources

該規則動作所使用的資源集合。

ruleName

符合的規則名稱。

topicName

訂閱主題的名稱。

RuleExecution 日誌項目

當 AWS IoT 規則引擎觸發規則的動作時，會產生RuleExecution日誌項目。

RuleExecution 日誌項目範例

```
{
    "timestamp": "2017-08-10 16:32:46.070",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "resources": {
        "RepublishTopic": "rules/republish"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

RuleExecution 搭配批次處理日誌項目範例

```
{
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "ruleName": "rule_test",
    "ruleAction": "HttpAction",
    "resources": {
        "Url": "https://example.com",
```

```
        "ConfirmationUrl": "https://example.com"
    },
    "details": "HttpAction made a request to the specified endpoint",
    "batchDetails": {
        "timestamps": [
            "1234567890123",
            "1234567890123",
            "1234567890123"
        ],
        "traceIds": [
            "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537c",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537d"
        ],
        "clientIds": [
            "N/A",
            "N/A",
            "N/A"
        ],
        "topicNames": [
            "topic/ruletest",
            "topic/ruletest",
            "topic/ruletest"
        ],
        "principalIds": [
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
        ],
        "batchSize": 3,
        "batchSizeInBytes": 114
    }
}
```


除了 常用 CloudWatch Logs 屬性，`RuleExecution` 記錄項目包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

resources

該規則動作所使用的資源集合。

ruleAction

受到觸發的動作名稱。

ruleName

符合的規則名稱。

topicName

訂閱主題的名稱。

RuleMatch 日誌項目

當訊息中介裝置收到符合規則的訊息`RuleMatch`時， AWS IoT 規則引擎會產生具有 `eventType` 的日誌項目。

RuleMatch 日誌項目範例

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleMatch",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`RuleMatch`日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

ruleName

符合的規則名稱。

topicName

訂閱主題的名稱。

RuleEvaluation 限流日誌項目

當 AWS IoT 規則引擎調節每個帳戶限制每秒規則評估的規則評估時，它會產生具有 `eventType`的 `RuleEvaluation`和 原因為 的日誌項目`Throttled`。

RuleEvaluation 限流日誌項目範例

```
{
    "timestamp": "2024-11-13 07:32:27.856",
    "logLevel": "ERROR",
    "traceId": "82995222-049b-ee5c-0cd4-15635f052ed3",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleEvaluation",
    "clientId": "57b3b41a-c726-44ab-bcd4-2cc1c2cc4295",
    "topicName": "N/A",
    "ruleName": "N/A",
    "principalId": "N/A",
    "reason": "Throttled",
    "details": "Exceeded Rule Evaluations per second per account limit"
}
```

除了 常用 CloudWatch Logs 屬性，`RuleEvaluation` 記錄項目包含下列屬性：

clientId

提出請求的用戶端 ID。

詳細資訊

錯誤的簡要說明。

principalId

提出請求的委託人 ID。此日誌項目一律為 N/A。

reason

字串 "Throttled"。

ruleName

規則的名稱。此日誌項目一律為 N/A。

topicName

已發佈主題的名稱。此日誌項目一律為 N/A。

RuleExecutionThrottled 日誌項目

當規則引擎因為惡意行為而調節規則評估時，例如過多的基本擷取，將請求發佈至不存在的規則時，AWS IoT 規則引擎會產生具有 之 `eventType`的日誌項目`RuleExecutionThrottled`。

RuleExecutionThrottled 日誌項目範例

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecutionThrottled",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleExecutionThrottled",
    "details": "Exection of Rule example_rule throttled"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`RuleExecutionThrottled`日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

詳細資訊

錯誤的簡要說明。

principalId

提出請求的委託人 ID。

reason

字串「RuleExecutionThrottled」。

ruleName

欲觸發的規則名稱。

topicName

已發佈的主題名稱。

RuleNotFound 日誌項目

當 AWS IoT 規則引擎找不到具有指定名稱的規則時，會產生具有 之 `eventType`的日誌項目`RuleNotFound`。

RuleNotFound 日誌項目範例

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleNotFound",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleNotFound",
    "details": "Rule example_rule not found"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`RuleNotFound`日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

詳細資訊

錯誤的簡要說明。

principalId

提出請求的委託人 ID。

reason

"RuleNotFound" 字串。

ruleName

找不到的規則名稱。

topicName

已發佈的主題名稱。

StartingRuleExecution 日誌項目

當 AWS IoT 規則引擎開始觸發規則的動作時，會產生具有 之 `eventType`的日誌項目`StartingRuleExecution`。

StartingRuleExecution 日誌項目範例

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "DEBUG",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartingRuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch Logs 屬性 之外，`rule-`日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

principalId

提出請求的委託人 ID。

ruleAction

受到觸發的動作名稱。

ruleName

符合的規則名稱。

topicName

訂閱主題的名稱。

任務日誌項目

AWS IoT 任務服務會產生下列事件的日誌項目。從裝置收到 MQTT 或 HTTP 請求時系統會產生日誌項目。

主題

- DescribeJobExecution 日誌項目
- GetPendingJobExecution 日誌項目
- ReportFinalJobExecutionCount 日誌項目
- StartNextPendingJobExecution 日誌項目
- UpdateJobExecution 日誌項目

DescribeJobExecution 日誌項目

AWS IoT Jobs 服務會在服務收到描述任務執行的請求`DescribeJobExecution`時，使用 `eventType` 的 產生日誌項目。

DescribeJobExecution 日誌項目範例

```
{
    "timestamp": "2017-08-10 19:13:22.841",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DescribeJobExecution",
```

```
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/get",
    "clientToken": "myToken",
    "details": "The request status is SUCCESS."
}
```

除了 常用 CloudWatch Logs 屬性 之外，GetJobExecution日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

clientToken

用來確保請求冪等的，是一個唯一、區分大小寫的識別符。如需詳細資訊，請參閱如何確保冪等。

詳細資訊

任務裝置的其他資訊。

jobId

任務執行的任務 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

用來提出請求的主題。

GetPendingJobExecution 日誌項目

AWS IoT Jobs 服務會在服務收到任務執行請求GetPendingJobExecution時，使用 eventType 的產生日誌項目。

GetPendingJobExecution 日誌項目範例

```
{
    "timestamp": "2018-06-13 17:45:17.197",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
```

```
    "status": "Success",
    "eventType": "GetPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "299966ad-54de-40b4-99d3-4fc8b52da0c5",
    "topicName": "$aws/things/299966ad-54de-40b4-99d3-4fc8b52da0c5/jobs/get",
    "clientToken": "24b9a741-15a7-44fc-bd3c-1ff2e34e5e82",
    "details": "The request status is SUCCESS."
}
```

除了 常用 CloudWatch Logs 屬性 之外，GetPendingJobExecution日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

clientToken

用以確保請求冪等的，是一個唯一、區分大小寫的識別符。如需詳細資訊，請參閱如何確保冪等。

詳細資訊

任務裝置的其他資訊。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

訂閱主題的名稱。

ReportFinalJobExecutionCount 日誌項目

AWS IoT Jobs 服務會在任務完成ReportFinalJobExecutionCount時，使用 entryType 的 產生日誌項目。

ReportFinalJobExecutionCount 日誌項目範例

```
{
    "timestamp": "2017-08-10 19:44:16.776",
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ReportFinalJobExecutionCount",
    "jobId": "002",
```

```
    "details": "Job 002 completed. QUEUED job execution count: 0 IN_PROGRESS job
 execution count: 0 FAILED job execution count: 0 SUCCEEDED job execution count: 1
 CANCELED job execution count: 0 REJECTED job execution count: 0 REMOVED job execution
 count: 0"
}
```

除了 常用 CloudWatch Logs 屬性 之外，ReportFinalJobExecutionCount日誌項目還包含下列屬性：

詳細資訊

任務裝置的其他資訊。

jobId

任務執行的任務 ID。

StartNextPendingJobExecution 日誌項目

當它收到啟動下一個待定任務執行的請求時， AWS IoT Jobs 服務會產生具有 之 eventType的日誌項目StartNextPendingJobExecution。

StartNextPendingJobExecution 日誌項目範例

```
{
    "timestamp": "2018-06-13 17:49:51.036",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartNextPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "95c47808-b1ca-4794-bc68-a588d6d9216c",
    "topicName": "$aws/things/95c47808-b1ca-4794-bc68-a588d6d9216c/jobs/start-next",
    "clientToken": "bd7447c4-3a05-49f4-8517-dd89b2c68d94",
    "details": "The request status is SUCCESS."
}
```

除了 常用 CloudWatch Logs 屬性 之外，StartNextPendingJobExecution日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

clientToken

用以確保請求冪等的，是一個唯一、區分大小寫的識別符。如需詳細資訊，請參閱如何確保冪等。

詳細資訊

任務裝置的其他資訊。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

用來提出請求的主題。

UpdateJobExecution 日誌項目

AWS IoT Jobs 服務會在服務收到更新任務執行的請求UpdateJobExecution時，使用 eventType 的 產生日誌項目。

UpdateJobExecution 日誌項目範例

```
{
    "timestamp": "2017-08-10 19:25:14.758",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/update",
    "clientToken": "myClientToken",
    "versionNumber": "1",
    "details": "The destination status is IN_PROGRESS. The request status is SUCCESS."
}
```

除了 常用 CloudWatch Logs 屬性 之外，UpdateJobExecution日誌項目還包含下列屬性：

clientId

提出請求的用戶端 ID。

clientToken

用以確保請求冪等的，是一個唯一、區分大小寫的識別符。如需詳細資訊，請參閱如何確保冪等。

詳細資訊

任務裝置的其他資訊。

jobId

任務執行的任務 ID。

protocol

用來提出要求的協定。有效值為 MQTT 或 HTTP。

topicName

用來提出請求的主題。

versionNumber

任務執行的版本。

裝置佈建日誌項目

AWS IoT 裝置佈建服務會產生下列事件的日誌。

主題

- GetDeviceCredentials 日誌項目
- ProvisionDevice 日誌項目

GetDeviceCredentials 日誌項目

AWS IoT 裝置佈建服務會在用戶端呼叫 GetDeviceCredential時，使用 eventType 的 產生日誌項目GetDeviceCredential。

GetDeviceCredentials 日誌項目範例

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
```

```
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "GetDeviceCredentials",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
}
```

除了 常用 CloudWatch Logs 屬性 之外，GetDeviceCredentials日誌項目還包含下列屬性：

詳細資訊

錯誤的簡要說明。

deviceCertificateId

裝置憑證的 ID。

ProvisionDevice 日誌項目

AWS IoT 裝置佈建服務會在用戶端呼叫 ProvisionDevice時，使用 eventType 的 產生日誌項目ProvisionDevice。

ProvisionDevice 日誌項目範例

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "ProvisionDevice",
  "provisioningTemplateName" : "myTemplate",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
 }
```

除了 常用 CloudWatch Logs 屬性 之外，ProvisionDevice日誌項目還包含下列屬性：

詳細資訊

錯誤的簡要說明。

deviceCertificateId

裝置憑證的 ID。

provisioningTemplateName

佈建範本的名稱。

動態物件群組日誌項目

AWS IoT 動態物件群組會為下列事件產生日誌。

主題

- AddThingToDynamicThingGroupsFailed 日誌項目

AddThingToDynamicThingGroupsFailed 日誌項目

當 AWS IoT 無法將物件新增至指定的動態群組時，會產生具有 之 `eventType`的日誌項目`AddThingToDynamicThingGroupsFailed`。這會在物件符合要在動態物件群組中的準則時發生；不過，它無法加入至動態群組，或從動態群組中移除。發生這種情況的原因是：

- 事情已經屬於群組的最大數量。
- --override-dynamic-groups 選項用於將物件新增到靜態物件群組。它已經從動態物件群組中移除，使之成為可能。

如需詳細資訊，請參閱動態物件群組限制與衝突。

AddThingToDynamicThingGroupsFailed 日誌項目範例

此範例顯示 `AddThingToDynamicThingGroupsFailed` 錯誤的日誌項目。在此範例中，TestThing 符合 `dynamicThingGroupNames` 中列出的動態物件群組中的準則，但無法加入到那些動態群組中，如 `reason` 所述。

```
{
 "timestamp": "2020-03-16 22:24:43.804",
 "logLevel": "ERROR",
 "traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
 "accountId": "57EXAMPLE833",
 "status": "Failure",
 "eventType": "AddThingToDynamicThingGroupsFailed",
 "thingName": "TestThing",
```

```
 "dynamicThingGroupNames": [
  "DynamicThingGroup11",
  "DynamicThingGroup12",
  "DynamicThingGroup13",
  "DynamicThingGroup14"
 ],
 "reason": "The thing failed to be added to the given dynamic thing group(s) because
 the thing already belongs to the maximum allowed number of groups."
}
```

除了 常用 CloudWatch Logs 屬性 之外，AddThingToDynamicThingGroupsFailed日誌項目還包含下列屬性：

dynamicThingGroupNames

無法新增物件的動態物件群組的陣列。

reason

物件無法新增到動態物件群組的原因。

thingName

無法新增至動態物件群組的物件名稱。

機群索引日誌項目

AWS IoT 機群索引會為下列事件產生日誌項目。

主題

- NamedShadowCountForDynamicGroupQueryLimitExceeded 日誌項目

NamedShadowCountForDynamicGroupQueryLimitExceeded 日誌項目

對於動態群組中並非特定於資料來源的查詢術語，每個物件最多處理 25 個已命名影子。當物件違反此限制時，會發出 NamedShadowCountForDynamicGroupQueryLimitExceeded 事件類型。

NamedShadowCountForDynamicGroupQueryLimitExceeded 日誌項目範例

此範例顯示 NamedShadowCountForDynamicGroupQueryLimitExceeded 錯誤的日誌項目。在此範例中，基於 DynamicGroup 的所有值結果可能不正確，如 reason 欄位中所述。

```
{
```

```
"timestamp": "2020-03-16 22:24:43.804",
"logLevel": "ERROR",
"traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
"accountId": "571032923833",
"status": "Failure",
"eventType": "NamedShadowCountForDynamicGroupQueryLimitExceeded",
"thingName": "TestThing",
"reason": "A maximum of 25 named shadows per thing are processed for non-data source
 specific query terms in dynamic groups."
}
```

常用 CloudWatch Logs 屬性

所有 CloudWatch Logs 記錄項目皆包含下列這些屬性：

accountId

您的 AWS 帳戶 ID。

eventType

產生日誌的事件類型。事件類型的值會根據產生該日誌項目的事件而定。每個日誌項目說明都包含該 `eventType` 日誌項目的值。

logLevel

使用的日誌層級。如需詳細資訊，請參閱 the section called “日誌層級”。

status

請求的狀態。

timestamp

用戶端連線至 AWS IoT 訊息代理程式時，人類看得懂的 UTC 時間戳記。

traceId

隨機產生的識別符，可用於建立某個特定請求的所有日誌的關聯性。

將裝置端日誌上傳到 Amazon CloudWatch

您可以將裝置端的歷史日誌上傳到 Amazon CloudWatch，以現場監控和分析裝置的活動。設備端日誌可以包括系統、應用程式和裝置日誌檔案。此程序使用 CloudWatch Logs 規則動作參數，將裝置端日誌發佈至客戶定義的日誌群組。

運作方式

當 AWS IoT 裝置將包含格式化日誌檔案的 MQTT 訊息傳送到 AWS IoT 主題時，程序就會開始。AWS IoT 規則會監控訊息主題，並將日誌檔案傳送至您定義的 CloudWatch Logs 群組。然後，您可以檢閱和分析資訊。

主題

- MQTT 主題
- 規則動作

MQTT 主題

選擇您將用來發佈日誌的 MQTT 主題名稱空間。我們建議將此格式用於共同主題空間 `$aws/rules/things/thing_name/logs`，並將此格式用於錯誤主題 `$aws/rules/things/thing_name/logs/errors`。建議使用日誌和錯誤主題的命名結構，但並非必要。如需詳細資訊，請參閱設計 AWS IoT Core的 MQTT 主題。

透過使用建議的常見主題空間，您可以使用 AWS IoT 基本擷取預留主題。 AWS IoT 基本擷取會將裝置資料安全地傳送至 AWS IoT 規則動作支援 AWS 的服務。基本擷取會從擷取路徑移除發佈/訂閱訊息代理程式，因此更具成本效益。如需詳細資訊，請參閱使用基本擷取減少簡訊費。

如果您使用批次模式來上傳日誌檔案，您的訊息必須遵循包含 UNIX 時間戳記和訊息的特定格式。如需詳細資訊，請參閱 CloudWatch Logs 規則動作中的批次模式 MQTT 訊息格式要求主題。

規則動作

當 從用戶端裝置 AWS IoT 接收 MQTT 訊息時， AWS IoT 規則會監控客戶定義的主題，並將內容發佈到您定義的 CloudWatch 日誌群組。此程序使用 CloudWatch Logs 檔規則動作來監控 MQTT 是否有日誌檔案批次。如需詳細資訊，請參閱 CloudWatch Logs AWS IoT 規則動作。

批次模式

`batchMode` 是 AWS IoT CloudWatch Logs 規則動作中的布林值參數。這個參數是可選的，預設情況下是 off (`false`)。若要批次上傳裝置端日誌檔案，您必須在建立 AWS IoT 規則時開啟此參數 (`true`)。如需詳細資訊，請參閱 AWS IoT 規則動作區段中的 CloudWatch Logs。

使用 AWS IoT 規則上傳裝置端日誌

您可以使用 AWS IoT 規則引擎，將現有裝置端日誌檔案 （ 系統、應用程式和裝置用戶端日誌 ） 的日誌記錄上傳至 Amazon CloudWatch。當裝置端日誌發佈到 MQTT 主題時，CloudWatch Logs 規則動

作會將訊息傳輸到 CloudWatch Logs。此程序概述如何使用規則動作 `batchMode` 參數開啟 (設定為 `true`) 來批次上傳裝置日誌。

若要開始將裝置端日誌上傳到 CloudWatch，請完成下列先決條件。

先決條件

開始之前，請執行以下動作：

- 建立至少一個向 註冊 AWS IoT Core 為 AWS IoT 物件的目標 IoT 裝置。如需詳細資訊，請參閱建立物件。
- 判斷用於擷取和錯誤的 MQTT 主題空間。如需 MQTT 主題和建議命名慣例的詳細資訊，請參閱將裝置端日誌上傳到 Amazon CloudWatch 中的 MQTT 主題 MQTT 主題一節。

如需這些先決條件的詳細資訊，請參閱將裝置端日誌上傳至 CloudWatch。

建立 CloudWatch 日誌群組

若要建立 CloudWatch 日誌群組，請完成以下步驟。根據您偏好透過 AWS 管理主控台 或 AWS Command Line Interface () 執行步驟，選擇適當的索引標籤AWS CLI。

AWS 管理主控台

使用 建立 CloudWatch 日誌群組 AWS 管理主控台

1. 開啟 AWS 管理主控台 並導覽至 CloudWatch。
2. 在導覽列上，選擇 Logs (日誌)，然後選擇 Log groups (日誌群組)。
3. 選擇 Create log group (建立日誌群組)。
4. 更新 Log group name (日誌群組名稱)，並選擇性地更新 Retention settings (保留設定) 欄位。
5. 選擇建立。

AWS CLI

使用 建立 CloudWatch 日誌群組 AWS CLI

1. 若要建立日誌群組，請執行以下命令。如需詳細資訊，請參閱 AWS CLI v2 命令參考 `create-log-group` 中的 。

   將範例 (`uploadLogsGroup`) 中的日誌群組名稱取代為您偏好的名稱。

```
aws logs create-log-group --log-group-name uploadLogsGroup
```

2. 若要確認日誌群組已正確建立，請執行下列命令。

```
aws logs describe-log-groups --log-group-name-prefix uploadLogsGroup
```

輸出範例：

```
{
    "logGroups": [
        {
            "logGroupName": "uploadLogsGroup",
            "creationTime": 1674521804657,
            "metricFilterCount": 0,
            "arn": "arn:aws:logs:us-east-1:111122223333:log-
group:uploadLogsGroup:*",
            "storedBytes": 0
        }
    ]
}
```

建立主題規則

若要建立 AWS IoT 規則，請完成下列步驟。根據您偏好透過 AWS 管理主控台 或 AWS Command Line Interface () 執行步驟，選擇適當的索引標籤AWS CLI。

AWS 管理主控台

使用 建立主題規則 AWS 管理主控台

1. 開啟規則中樞。
   a. 開啟 AWS 管理主控台 並導覽至 AWS IoT。
   b. 在導覽列上，選擇 Message routing (訊息路由)，然後選擇 Rules (規則)。
   c. 選擇建立規則。
2. 輸入規則屬性。
   a. 輸入英數字元的 Rule name (規則名稱)。

b. (選擇性) 輸入 Rule description (規則描述) 和 Tags (標籤)。

c. 選擇下一步。

3. 輸入 SQL 陳述式。

a. 使用您為擷取定義的 MQTT 主題輸入 SQL 敘述句。

例如 `SELECT * FROM '$aws/rules/things/thing_name/logs'`

b. 選擇下一步。

4. 輸入規則動作。

a. 在 Action 1 (動作 1) 選單中，選擇 CloudWatch logs (CloudWatch 日誌)。

b. 選擇 Log group name (日誌群組名稱)，然後選擇您建立的日誌群組。

c. 選取 Use batch mode (使用批次模式)。

d. 指定規則的 IAM 角色。

如果您擁有規則的 IAM 角色，請執行下列動作。

1. 在 IAM role (IAM 角色) 選單上，選擇您的 IAM 角色。

如果您沒有規則的 IAM 角色，請執行下列動作。

1. 選擇 Create new role (建立新角色)。
2. 在 Role name (角色名稱) 中，輸入唯一的名稱，然後選擇 Create (建立)。
3. 確認 IMA role (IAM 角色) 欄位中的 IAM 角色 名稱是否正確。

e. 選擇下一步。

5. 檢閱範本組態。

a. 檢閱 Job 範本的設定，以確認其正確無誤。

b. 完成時，選擇 Create (建立)。

AWS CLI

使用 建立 IAM 角色和主題規則 AWS CLI

1. 建立授予 AWS IoT 規則權限的 IAM 角色。

a. 建立 IAM 政策。

若要建立 IAM 政策，請執行下列命令。請務必更新 `policy-name` 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考[create-policy]中的 。

> **Note**
>
> 如果您使用的是 Microsoft Windows 作業系統，您可能需要將行尾標記 (\) 取代為勾號 (`) 或其他字元。

```
aws iam create-policy \
    --policy-name uploadLogsPolicy \
    --policy-document \
'{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateTopicRule",
            "iot:Publish",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:GetLogEvents"
        ],
        "Resource": "*"
    }
}'
```

b. 將政策 ARN 從輸出複製到文字編輯器中。

輸出範例：

```
{
    "Policy": {
        "PolicyName": "uploadLogsPolicy",
        "PermissionsBoundaryUsageCount": 0,
        "CreateDate": "2023-01-23T18:30:10Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "AAABBBCCCDDDEEEFFFGGG",
        "DefaultVersionId": "v1",
```

```
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:policy/uploadLogsPolicy",
        "UpdateDate": "2023-01-23T18:30:10Z"
    }
}
```

c. 建立 IAM 角色和信任政策。

若要建立 IAM 政策，請執行下列命令。請務必更新 role-name 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考create-role中的 。

```
aws iam create-role \
--role-name uploadLogsRole \
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

d. 將 IAM 政策連接至角色。

若要建立 IAM 政策，請執行下列命令。請務必更新 role-name 和 policy-arn 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考attach-role-policy中的 。

```
aws iam attach-role-policy \
--role-name uploadLogsRole \
--policy-arn arn:aws:iam::111122223333:policy/uploadLogsPolicy
```

e. 檢閱角色。

若要確認 IAM 角色已正確建立，請執行下列命令。請務必更新 role-name 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考get-role中的 。

```
aws iam get-role --role-name uploadLogsRole
```

輸出範例：

```
{
    "Role": {
        "Path": "/",
        "RoleName": "uploadLogsRole",
        "RoleId": "AAABBBCCCDDDEEEFFFGGG",
        "Arn": "arn:aws:iam::111122223333:role/uploadLogsRole",
        "CreateDate": "2023-01-23T19:17:15+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "iot.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {}
    }
}
```

2. 在 中建立 AWS IoT 主題規則 AWS CLI。

   a. 若要建立 AWS IoT 主題規則，請執行下列命令。請務必更新 `--rule-name`、`sql` 陳述式`descriptionroleARN` 、 和 `logGroupName` 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考中的 create-topic-rule。

```
aws iot create-topic-rule \
--rule-name uploadLogsRule \
--topic-rule-payload \
 '{
  "sql":"SELECT * FROM 'rules/things/thing_name/logs'",
```

```
  "description":"Upload logs test rule",
  "ruleDisabled":false,
  "awsIotSqlVersion":"2016-03-23",
  "actions":[
   {"cloudwatchLogs":
    {"roleArn":"arn:aws:iam::111122223333:role/uploadLogsRole",
    "logGroupName":"uploadLogsGroup",
    "batchMode":true}
    }
   ]
}'
```

b. 若要確認規則已正確建立，請執行下列命令。請務必更新 `role-name` 參數值。如需詳細資訊，請參閱 AWS CLI v2 命令參考中的 get-topic-rule。

```
aws iot get-topic-rule --rule-name uploadLogsRule
```

輸出範例：

```
{
    "ruleArn": "arn:aws:iot:us-east-1:111122223333:rule/uploadLogsRule",
    "rule": {
        "ruleName": "uploadLogsRule",
        "sql": "SELECT * FROM rules/things/thing_name/logs",
        "description": "Upload logs test rule",
        "createdAt": "2023-01-24T16:28:15+00:00",
        "actions": [
            {
                "cloudwatchLogs": {
                    "roleArn": "arn:aws:iam::111122223333:role/
uploadLogsRole",
                    "logGroupName": "uploadLogsGroup",
                    "batchMode": true
                }
            }
        ],
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23"
    }
}
```

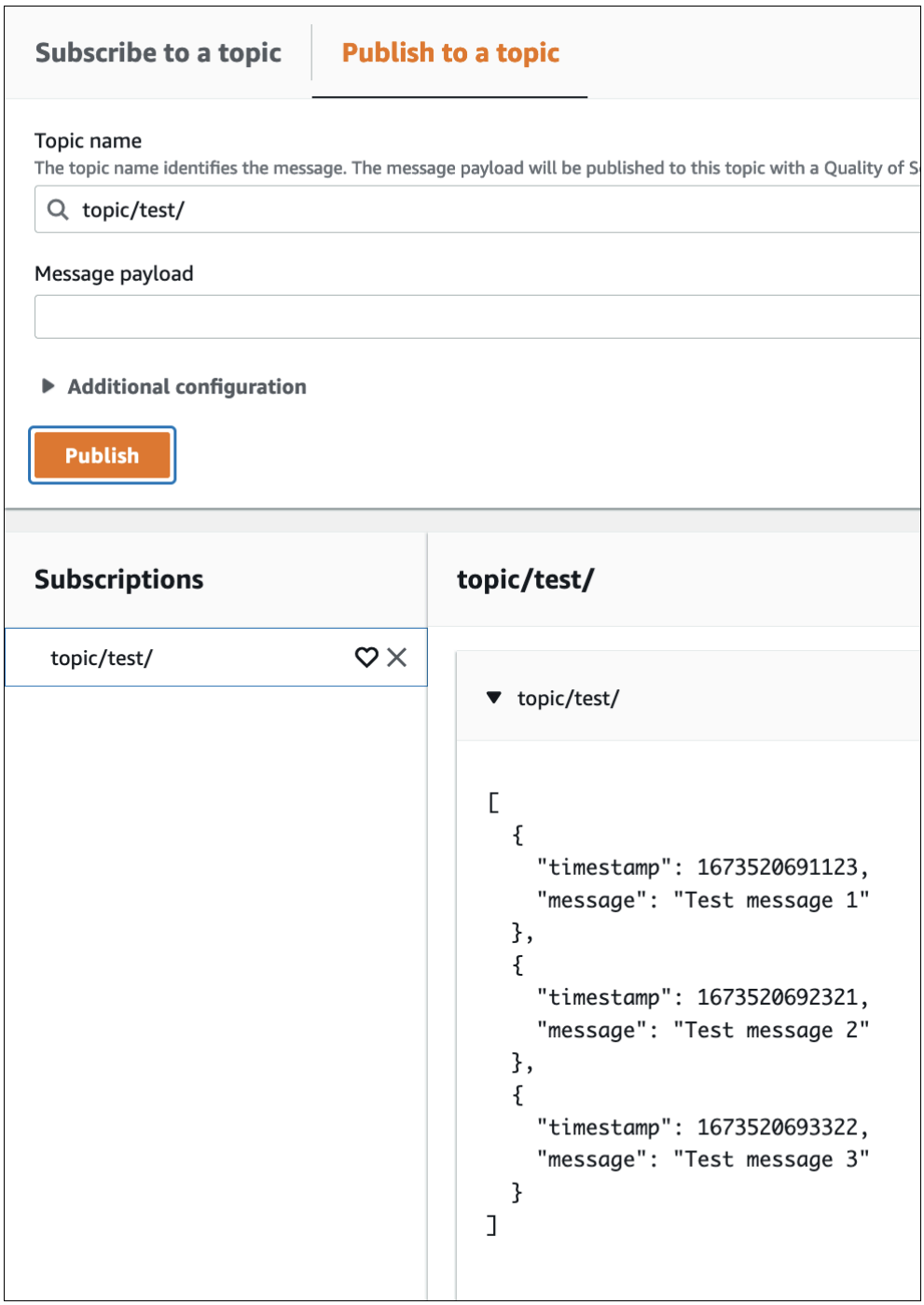
將裝置端日誌傳送至 AWS IoT

將裝置端日誌傳送至 AWS IoT

1. 若要將歷史日誌傳送到 AWS IoT，請與您的裝置通訊，以確保下列事項。
   - 日誌資訊會傳送至此程序的 Prerequisites (必要條件) 區段中所指定的正確主題命名空間。

     例如 `$aws/rules/things/thing_name/logs`
   - MQTT 訊息承載已正確格式化。如需 MQTT 主題和建議命名慣例的詳細資訊，請參閱 中的MQTT 主題一節將裝置端日誌上傳到 Amazon CloudWatch。
2. 確認在 MQTT 用戶端內收到 AWS IoT MQTT 訊息。
   a. 開啟 AWS 管理主控台 並導覽至 AWS IoT。
   b. 若要檢視 MQTT 測試用戶端，請在導覽列上選擇測試、MQTT 測試用戶端。
   c. 對於訂閱主題主題主題篩選條件，輸入主題命名空間。
   d. 選擇 Subscribe (訂閱)。

      MQTT 訊息會顯示在 Subscriptions (訂閱) 與 Topic (主題) 表格中，如下所示。這些訊息可能需要 5 分鐘才會顯示。



檢視日誌資料

檢閱 CloudWatch Logs 中的日誌紀錄

1. 開啟 AWS 管理主控台，然後導覽至 CloudWatch。
2. 在導覽列上，選擇 Logs，Logs Insights。

3. 在選取日誌群組 (Select log group) 功能表中，選擇您在 AWS IoT 規則中指定的日誌群組。
4. 在 Logs insights (日誌深入解析) 頁面上，選擇 Run query (執行查詢)。

使用 記錄 AWS IoT API 呼叫 AWS CloudTrail

AWS IoT 已與 服務整合 AWS CloudTrail，此服務提供使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS IoT。CloudTrail 會將 的所有 API 呼叫擷取 AWS IoT 為事件，包括來自 AWS IoT 主控台的呼叫，以及來自對 AWS IoT APIs的程式碼呼叫。如果您建立線索，您可以將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 的事件 AWS IoT。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。您可以使用 CloudTrail 所收集的資訊來判斷提出的請求 AWS IoT、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱AWS CloudTrail 《使用者指南》。

AWS IoT CloudTrail 中的資訊

當您建立帳戶 AWS 帳戶 時，您的 上會啟用 CloudTrail。當活動在 中發生時 AWS IoT，該活動會與事件歷史記錄中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。您可以在 中檢視、搜尋和下載最近的事件 AWS 帳戶。如需詳細資訊，請參閱《使用 CloudTrail 事件歷史記錄檢視事件》https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events.html。

如需您 AWS 帳戶帳戶中正在進行事件的記錄 (包含 AWS IoT的事件)，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有 AWS 區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中收集的事件資料。如需詳細資訊，請參閱：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案

> **Note**
>
> AWS IoT CloudTrail 不會記錄資料平面動作 （ 裝置端 ）。使用 CloudWatch 監控這些動作。

一般而言，CloudTrail 會記錄進行變更的 AWS IoT 控制平面動作。CreateThing、CreateKeysAndCertificate 和 UpdateCertificate 等呼叫會離開 CloudTrail 項目，而 ListThings 和 ListTopicRules 等呼叫則不會。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

AWS IoT 動作記錄在 AWS IoT API 參考中。 AWS IoT Wireless 動作記錄在 AWS IoT Wireless API 參考中。

了解 AWS IoT 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 `AttachPolicy` 動作的 CloudTrail 日誌項目。

```
{
    "timestamp":"1460159496",
    "AdditionalEventData":"",
    "Annotation":"",
    "ApiVersion":"",
    "ErrorCode":"",
    "ErrorMessage":"",
    "EventID":"8bff4fed-c229-4d2d-8264-4ab28a487505",
    "EventName":"AttachPolicy",
    "EventTime":"2016-04-08T23:51:36Z",
    "EventType":"AwsApiCall",
    "ReadOnly":"",
    "RecipientAccountList":"",
    "RequestID":"d4875df2-fde4-11e5-b829-23bf9b56cbcd",
    "RequestParamters":{
```

```
        "principal":"arn:aws:iot:us-
east-1:123456789012:cert/528ce36e8047f6a75ee51ab7beddb4eb268ad41d2ea881a10b67e8e76924d894",
        "policyName":"ExamplePolicyForIoT"
    },
    "Resources":"",
    "ResponseElements":"",
    "SourceIpAddress":"52.90.213.26",
    "UserAgent":"aws-internal/3",
    "UserIdentity":{
        "type":"AssumedRole",
        "principalId":"AKIAI44QH8DHBEXAMPLE",
        "arn":"arn:aws:sts::12345678912:assumed-role/iotmonitor-us-east-1-beta-
InstanceRole-1C5T1YCYMHPYT/i-35d0a4b6",
        "accountId":"222222222222",
        "accessKeyId":"access-key-id",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"Fri Apr 08 23:51:10 UTC 2016"
            },
            "sessionIssuer":{
                "type":"Role",
                "principalId":"AKIAI44QH8DHBEXAMPLE",
                "arn":"arn:aws:iam::123456789012:role/executionServiceEC2Role/
iotmonitor-us-east-1-beta-InstanceRole-1C5T1YCYMHPYT",
                "accountId":"222222222222",
                "userName":"iotmonitor-us-east-1-InstanceRole-1C5T1YCYMHPYT"
            }
        },
        "invokedBy":{
            "serviceAccountId":"111111111111"
        }
    },
    "VpcEndpointId":""
}
```

的規則 AWS IoT

規則可讓您的裝置與 互動 AWS 服務。規則的分析和動作的執行均是以 MQTT 的主題流為依歸。您可以使用規則來支援下列任務：

- 擴增或篩選從裝置接收的資料
- 將裝置接收的資料寫入 Amazon DynamoDB 資料庫。
- 將檔案儲存至 Amazon S3。
- 使用 Amazon SNS 將推送通知傳送給所有使用者。
- 將資料發佈至 Amazon SQS 佇列。
- 呼叫 Lambda 函數來擷取資料。
- 使用 Amazon Kinesis 來處理來自大量裝置的訊息。
- 將資料傳送至 Amazon OpenSearch Service。
- 擷取 CloudWatch 指標。
- 變更 CloudWatch 警示。
- 將資料從 MQTT 訊息傳送至 Amazon SageMaker AI，以根據機器學習 (ML) 模型進行預測。
- 開始處理 Step Functions 狀態機器。
- 將訊息資料傳送至 AWS IoT SiteWise中的資產屬性。
- 將訊息資料傳送至 Web 應用程式或服務。

您的規則可以使用 MQTT 訊息，這些訊息會通過 the section called “裝置通訊協定” 支援的發佈/訂閱協定。您也可以使用基本擷取功能，安全地將裝置資料傳送至先前 AWS 服務 列出的 ，而不會產生簡訊費用。基本擷取功能會從擷取路徑移除發佈/訂閱訊息代理程式，使資料流程最佳化。這使得它具有成本效益，同時仍保有 的安全性和資料處理功能 AWS IoT。

在 AWS IoT 可以執行這些動作之前，您必須授予它代表您存取 資源 AWS 的許可。執行動作時，您需為您使用的 AWS 服務 支付標準費用。

目錄

- 存取控制
- 傳遞角色許可
- 建立 AWS IoT 規則

- 管理 AWS IoT 規則
- AWS IoT 規則動作
- 規則疑難排解
- 使用 AWS IoT 規則存取跨帳戶資源
- 錯誤處理 (錯誤動作)
- 使用基本擷取減少簡訊費用
- AWS IoT SQL 參考

存取控制

授予 AWS IoT 規則所需的存取權

使用 IAM 角色來控制每個規則可存取 AWS 的資源。建立規則之前，您必須使用允許存取所需 AWS 資源的政策來建立 IAM 角色。在實作規則時 AWS IoT ， 會擔任此角色。

完成下列步驟以建立 IAM 角色和 AWS IoT 政策，授予 AWS IoT 規則所需的存取權 (AWS CLI)。

1. 將下列信任政策文件儲存為名為 的檔案，以授予擔任該角色的 AWS IoT 許可`iot-role-trust.json`。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:iot:us-east-1:123456789012:rule/
rulename"
            }
        }
```

```
        }
    ]
}
```

使用 create-role 命令來建立用於指定 iot-role-trust.json 檔案的 IAM 角色：

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document
 file://iot-role-trust.json
```

此令命的輸出結果如下所示：

```
{
 "Role": {
  "AssumeRolePolicyDocument": "url-encoded-json",
  "RoleId": "AKIAIOSFODNN7EXAMPLE",
  "CreateDate": "2015-09-30T18:43:32.821Z",
  "RoleName": "my-iot-role",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:role/my-iot-role"
 }
}
```

2. 將下列 JSON 儲存至名為 my-iot-policy.json 的檔案。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "dynamodb:*",
   "Resource": "*"
  }
 ]
}
```

此 JSON 是授予 AWS IoT 管理員 DynamoDB 存取權的範例政策文件。

使用 create-policy 命令，在擔任角色時授予 AWS 資源的 AWS IoT 存取權，並傳入 my-iot-policy.json 檔案：

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy.json
```

如需如何在 政策 AWS 服務 中授予 存取權的詳細資訊 AWS IoT，請參閱 建立 AWS IoT 規則。

create-policy 命令的輸出結果含有該政策的 ARN。將政策連接至角色。

```
{
 "Policy": {
  "PolicyName": "my-iot-policy",
  "CreateDate": "2015-09-30T19:31:18.620Z",
  "AttachmentCount": 0,
  "IsAttachable": true,
  "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
  "DefaultVersionId": "v1",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:policy/my-iot-policy",
  "UpdateDate": "2015-09-30T19:31:18.620Z"
 }
}
```

3. 請使用 attach-role-policy 命令將政策連接到您的角色：

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

撤銷規則引擎存取權

若要立即撤銷規則引擎存取權，請執行下列動作：

1. 從信任政策中移除 iot.amazonaws.com
2. 依照步驟撤銷 IAM 角色工作階段

限制 HTTP 動作流量至規則引擎 IPv6 範圍

依來源地址限制傳入流量的端點必須允許 區域從下表的範圍。這適用於 HTTP 動作端點，這是您可以設定為只能透過 IPv6 連線的唯一目的地。對於其他動作，規則引擎會透過 IPv4 連線至雙堆疊目的地，因此沒有允許的範圍。

這些範圍僅適用於當規則引擎透過 IPv6 連線至您的端點時，即您的端點發佈 AAAA 記錄且沒有 A 記錄時。沒有同等的 IPv4 範圍，因此依來源地址限制規則引擎流量需要只能透過 IPv6 連線的端點。

每個範圍僅用於規則引擎動作，不會變更。其他區域的範圍會在可用時新增至此資料表。如需透過 IPv6 接收資料的步驟，請參閱 網路存取。

> **Note**
>
> 此資料表中未列出的區域沒有規則引擎 IPv6 範圍。

區域	IPv6 地址範圍
us-east-1	2600:1f33:8000::/48
us-east-2	2600:1f33:6000::/48
us-west-1	2600:1f33:c000::/48
us-west-2	2600:1f33:4000::/48
ap-east-1	2406:da33:e000::/48
ap-northeast-1	2406:da33:4000::/48
ap-northeast-2	2406:da33:2000::/48
ap-south-1	2406:da33:a000::/48
ap-southeast-1	2406:da33:8000::/48
ap-southeast-2	2406:da33:c000::/48
ap-southeast-5	2406:da33:800::/48
ca-central-1	2600:1f33:1000::/48
eu-central-1	2a05:d033:4000::/48
eu-north-1	2a05:d033:6000::/48

區域	IPv6 地址範圍
eu-south-1	2a05:d033:a000::/48
eu-south-2	2a05:d033:1000::/48
eu-west-1	2a05:d033:8000::/48
eu-west-2	2a05:d033:c000::/48
eu-west-3	2a05:d033:2000::/48
il-central-1	2a05:d033:5000::/48
sa-east-1	2600:1f33:e000::/48
us-gov-east-1	2600:1f33:5000::/48
us-gov-west-1	2600:1f33:2000::/48

下列範例會將傳入流量限制在 us-east-1範圍。將其取代為您的 區域的範圍。

API Gateway 資源政策，允許規則引擎並拒絕其他項目：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": "*",
   "Action": "execute-api:Invoke",
   "Resource": "execute-api:/*"
  },
  {
   "Effect": "Deny",
   "Principal": "*",
   "Action": "execute-api:Invoke",
   "Resource": "execute-api:/*",
   "Condition": {
    "NotIpAddress": {
     "aws:SourceIp": ["2600:1f33:8000::/48"]
    }
```

```
    }
   }
  ]
}
```

對於負載平衡器後方或 Amazon EC2 上的端點，將傳入規則新增至安全群組，允許連接埠 443 上來自您區域範圍的 TCP 流量。

傳遞角色許可

IAM 角色是規則定義的一部分，其授予指定於規則動作中資源的存取許可。當叫用規則的動作時，規則引擎會擔任該角色。角色必須在 AWS 帳戶 與規則相同的 中定義。

當建立或替換某項規則時，實際上就是在將某個角色傳送至規則引擎。執行此操作需要 `iam:PassRole` 許可。若要驗證您是否具有此許可，請建立一項授予 `iam:PassRole` 許可的政策，並將其連接至您的 IAM 使用者。以下政策顯示了如何允許角色的 `iam:PassRole` 許可。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1",
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": [
    "arn:aws:iam::123456789012:role/myRole"
   ]
  }
 ]
}
```

在此政策範例中，已將 `iam:PassRole` 許可授予 `myRole` 角色。該角色是使用角色的 ARN 來指定。將此政策連接至您的 IAM 使用者或是您使用者所屬的角色。如需詳細資訊，請參閱處理受管政策的相關文章。

> **Note**
>
> Lambda 函數使用的是資源型政策，即是政策直接連接至 Lambda 函數本身。在您建立一個叫用 Lambda 函數的規則時，並不會傳遞角色，因此建立該規則的使用者不需要 `iam:PassRole` 許可。如需 Lambda 函數授權的詳細資訊，請參閱使用資源政策授予許可。

建立 AWS IoT 規則

您可以建立 AWS IoT 規則，從連線的實物路由資料，以與其他 AWS 服務互動。 AWS IoT 規則包含下列元件：

規則的元件

元件	說明	必要或選用
規則名稱	規則的名稱。請注意，我們不建議您在規則名稱中使用個人身分識別資訊。	必要.
規則說明	規則的文字描述。請注意，我們不建議您在規則描述中使用個人身分識別資訊。	選用。
SQL 陳述式	簡化的 SQL 語法，用於篩選 MQTT 主題所收到的訊息，並將資料推送到他處。如需詳細資訊，請參閱AWS IoT SQL 參考。	必要.
SQL 版本	評估規則時所用的 SQL 規則引擎版本。雖然此屬性是選用的，但強烈建議您指定 SQL 版本。 AWS IoT Core 主控台`2016-03-23` 預設會將此屬性設定為 。如果未設定此屬性，例如在 AWS CLI 命令或 CloudFormation 範本中，`2015-10-08` 則會使用 。如需詳細資訊，請參閱SQL 版本。	必要.
一或多個動作	動作會在制定規則時 AWS IoT 執行。例如，您可將資料插入 DynamoDB 表格，將資料寫入 Amazon S3 儲存貯體，發佈至 Amazon SNS 主題，或呼叫 Lambda 函式。	必要.
錯誤動作	動作 AWS IoT 會在無法執行規則動作時執行。	選用。

建立 AWS IoT 規則之前，您必須使用允許存取所需 AWS 資源的政策來建立 IAM 角色。在實作規則時 AWS IoT ， 會擔任此角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權和傳遞角色許可。

建立規則時，請注意在主題上發佈的資料量。如果您建立了包含萬用字元主題模式的規則，這些規則可能會比對大部分的訊息。若是這種情況，您可能需要增加目標動作使用的 AWS 資源容量。我們建議在重新發佈規則中避免萬用字元主題模式，以防止重複處理並降低成本。

> **Note**
>
> 規則的建立和更新是屬於管理員層級的動作。任何擁有許可而能建立或更新規則的使用者，均能夠存取規則所處理的資料。

建立規則（主控台）

建立規則 (AWS 管理主控台)

使用 AWS 管理主控台命令來建立規則：

1. 開啟 AWS IoT 主控台。
2. 在左側導覽上，從管理區段中選擇訊息路由。然後選擇規則。
3. 在規則頁面上，選擇建立規則。
4. 在指定規則屬性頁面上，輸入規則的名稱。規則描述和標籤是選用的。選擇下一步。
5. 在設定 SQL 陳述式頁面上，選擇 SQL 版本並輸入 SQL 陳述式。SQL 陳述式的範例可以是 `SELECT temperature FROM 'iot/topic' WHERE temperature > 50`。如需詳細資訊，請參閱 SQL 版本和 AWS IoT SQL 參考。
6. 在連接規則動作頁面上，新增規則動作以將資料路由至其他服務 AWS 。
    1. 在規則動作中，從下拉式清單中選取規則動作。例如，您可以選擇 Kinesis Stream。如需規則動作的詳細資訊，請參閱AWS IoT 規則動作。
    2. 根據您選擇的規則動作，輸入相關的組態詳細資訊。例如，如果您選擇 Kinesis Stream，您將需要選擇或建立資料串流資源，並選擇性地輸入組態詳細資訊，例如分割區索引鍵，用於在蒸汽中依碎片分組資料。
    3. 在 IAM 角色中，選擇或建立角色以授予對端點的 AWS IoT 存取權。請注意， AWS IoT 會自動在您的 IAM 角色`aws-iot-rule`下建立字首為 的政策。您可以選擇檢視，從 IAM 主控台檢視

IAM 角色和政策。錯誤動作是選用的。您可以在錯誤處理（錯誤動作）中找到更多資訊。如需為規則建立 IAM 角色的詳細資訊，請參閱授予規則所需的存取權。選擇下一步。

7. 在檢閱和建立頁面上，檢閱所有組態並視需要進行編輯。選擇建立。

成功建立規則後，您會在規則頁面上看到列出的規則。您可以選擇規則以開啟詳細資訊頁面，您可以在其中檢視規則、編輯規則、停用規則，以及刪除規則。

建立規則 (CLI)

建立規則 (AWS CLI)

請使用 create-topic-rule 命令來建立規則：

```
aws iot create-topic-rule --rule-name myrule --topic-rule-payload file://myrule.json
```

下列為一個範例承載檔案，其規則會將所有發送至 `iot/test` 主題的訊息插入指定的 DynamoDB 表格中。SQL 陳述式會篩選訊息，而角色 ARN 會授予寫入 DynamoDB 資料表的 AWS IoT 許可。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "dynamoDB": {
    "tableName": "my-dynamodb-table",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "hashKeyField": "topic",
    "hashKeyValue": "${topic(2)}",
    "rangeKeyField": "timestamp",
    "rangeKeyValue": "${timestamp()}"
   }
  }
 ]
}
```

以下為範例承載檔案，含有一項規則，會將所有發送至 `iot/test` 主題的訊息插入指定的 S3 儲存貯體。SQL 陳述式會篩選訊息，而角色 ARN 會授予寫入 Amazon S3 儲存貯體的 AWS IoT 許可。

```
{
 "awsIotSqlVersion": "2016-03-23",
```

```
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "actions": [
  {
   "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
    "bucketName": "amzn-s3-demo-bucket",
    "key": "myS3Key"
   }
  }
 ]
}
```

下列是承載檔案範例，該檔案具有將資料推送至 Amazon OpenSearch Service: 的規則：

```
{
 "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "OpenSearch": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es",
    "endpoint": "https://my-endpoint",
    "index": "my-index",
    "type": "my-type",
    "id": "${newuuid()}"
   }
  }
 ]
}
```

下列為範例承載檔案，具有會呼叫 Lambda 函數的規則：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function"
   }
```

```
  }
 ]
}
```

下列為範例承載檔案，具有會發佈至 Amazon SNS 主題的規則：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

以下為範例承載檔案，含有的規則會將資料重新發佈至不同的 MQTT 主題：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "topic": "my-mqtt-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

以下是承載檔案範例，其中包含將資料推送至 Amazon Data Firehose 串流的規則：

```
{
 "sql": "SELECT * FROM 'my-topic'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
```

```
 "actions": [
  {
   "firehose": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "deliveryStreamName": "my-stream-name"
   }
  }
 ]
}
```

以下是具有開始執行 Step Functions 狀態機器之規則的範例承載檔案。

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "stepFunctions": {
    "stateMachineName": "myCoolStateMachine",
    "executionNamePrefix": "coolRunning",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

管理 AWS IoT 規則

您可以使用下列動作來管理您的 AWS IoT 規則。

在本主題中：

- 標記規則
- 檢視規則
- 刪除規則

標記規則

您可以套用標記，將另一層特殊性新增至新規則或現有規則。標記會利用規則中的鍵值對，讓您更能控制規則套用至 AWS IoT 資源和服務的方式和位置。例如，您可以將規則範圍限制為僅適用於進行發

行前測試的試用版環境 (Key=environment, Value=beta)，或擷取所有僅從特定端點傳送至 iot/test 的主題，並將這兩項資料存放在 Amazon S3 儲存貯體中。

IAM 政策範例

對於示範如何授予規則標記許可的範例，請考慮使用者執行以下命令，以建立規則並將其標記為僅適用於試用版環境。

在該範例中，替換：

- *MyTopleName* 為規則的名稱。
- *myrule.json* 為政策文件的名稱。

```
aws iot create-topic-rule
    --rule-name MyTopicRuleName
    --topic-rule-payload file://myrule.json
    --tags "environment=beta"
```

這個範例必須使用下列 IAM 政策：

```
{
    "Version":"2012-10-17",
    "Statement":
    {
        "Action": [ "iot:CreateTopicRule", "iot:TagResource" ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:rule/MyTopicRuleName"
        ]
    }
}
```

以上範例說明名為 MyTopicRuleName 並僅適用於試用版環境的新建規則。政策聲明中的 iot:TagResource 與 MyTopicRuleName 特別呼叫，允許在建立或更新 MyTopicRuleName 時予以標記。建立規則時使用的參數 --tags "environment=beta" 將 MyTopicRuleName 範圍限制為只有您的測試版環境。如果您移除參數 --tags "environment=beta"，MyTopicRuleName 將適用於所有環境。

如需根據 AWS IoT 規則建立 IAM 角色和政策的詳細資料，請參閱 授予 AWS IoT 規則所需的存取權

如需標記資源的相關資訊，請參閱 標記您的 AWS IoT 資源。

檢視規則

請使用 list-topic-rules 命令來列出規則：

```
aws iot list-topic-rules
```

請使用 get-topic-rule 命令來取得規則的相關資訊：

```
aws iot get-topic-rule --rule-name myrule
```

刪除規則

規則結束使用後即可刪除。

刪除規則 (AWS CLI)

請使用 delete-topic-rule 命令來刪除規則：

```
aws iot delete-topic-rule --rule-name myrule
```

AWS IoT 規則動作

AWS IoT 規則動作指定呼叫規則時要執行的動作。您可以定義動作，將資料傳送至 Amazon DynamoDB 資料庫、將資料傳送至 Amazon Kinesis Data Streams、叫用 AWS Lambda 函數等。AWS IoT 支援下列動作， AWS 區域 其中提供動作的服務。

規則動作	說明	API 中的名稱
Apache Kafka	將訊息傳送至 Apache Kafka 叢集。	`kafka`
CloudWatch 警示	變更 Amazon CloudWatch 警示的狀態。	`cloudwatchAlarm`
CloudWatch Logs	將訊息傳送至 Amazon CloudWatch Logs。	`cloudwatchLogs`

規則動作	說明	API 中的名稱
CloudWatch 指標	將訊息傳送至 CloudWatch 指標。	`cloudwatchMetric`
DynamoDB	將訊息傳送至 DynamoDB 表格。	`dynamoDB`
DynamoDBv2	將訊息資料傳送至 DynamoDB 表格中的多個欄。	`dynamoDBv2`
Elasticsearch	將訊息傳送至 OpenSearch 端點。	`OpenSearch`
HTTP	將訊息發佈至 HTTPS 端點。	`http`
InfluxDB	將時間序列資料從 IoT 裝置寫入 InfluxDB。	`influxDB`
AWS IoT SiteWise	傳送訊息資料至 AWS IoT SiteWise 資產屬性。	`iotSiteWise`
Firehose	傳送訊息至 Firehose 交付串流。	`firehose`
Kinesis Data Streams	將訊息傳送至 Kinesis 資料串流。	`kinesis`
Lambda	以訊息資料作為輸入呼叫 Lambda 函數。	`lambda`
Location	將位置資料傳送至 Amazon Location Service。	`location`
OpenSearch	將訊息傳送至 Amazon OpenSearch Service 端點。	`OpenSearch`
重新發佈	可在另一個 MQTT 主題上重新發佈訊息。	`republish`

規則動作	說明	API 中的名稱
S3	將訊息存放於 Amazon Simple Storage Service (Amazon S3) 儲存貯體中。	`s3`
SNS	將訊息發佈為 Amazon Simple Notification Service (Amazon SNS)推送通知。	`sns`
SQS	將訊息發佈至 Amazon Simple Queue Service (Amazon SQS) 佇列。	`sqs`
步驟函數	啟動 AWS Step Functions 狀態機器。	`stepFunctions`
the section called “Timestream”	將訊息傳送至 Amazon Timestream 資料庫表格。	`timestream`

備註

- 在與其他服務資源 AWS 區域 相同的 中定義規則，以便規則動作可以與該資源互動。
- 如果發生間歇性錯誤， AWS IoT 規則引擎可能會多次嘗試執行動作。若所有嘗試都失敗了，則該訊息將遭捨棄，而且可在您的 CloudWatch Logs 中取得錯誤。您可以為失敗發生之後調用的每一個規則指定一個錯誤動作。如需詳細資訊，請參閱錯誤處理 (錯誤動作)。
- 某些規則動作會啟動服務中與 AWS Key Management Service (AWS KMS) 整合的動作，以支援靜態資料加密。如果您使用客戶受管 AWS KMS key (KMS 金鑰） 加密靜態資料，服務必須具有代表發起人使用 KMS 金鑰的許可。若要了解如何管理客戶受管 KMS 金鑰的許可，請參閱適當服務指南中的資料加密主題。如需客戶受管 KMS 金鑰的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 AWS Key Management Service 概念。

  您可以在錯誤動作的 SQL 陳述式中使用任何函數或替代範本，包括外部函數：`aws_lambda()`、`get_dynamodb()`、`get_registry_data()get_thing_shadow()`、、

get_secret()和 decode()。如果錯誤動作需要呼叫外部 函數，則叫用錯誤動作可能會導致外部函數產生額外的帳單。

Apache Kafka

Apache Kafka (Kafka) 動作會將訊息直接傳送到您的 Amazon Managed Streaming for Apache Kafka (Amazon MSK)、由 Confluent Cloud 等第三方供應商管理的 Apache Kafka 叢集，或自我管理的 Apache Kafka 叢集。使用 Kafka 規則動作，您可以將 IoT 資料路由到 Kafka 叢集。這可讓您針對各種用途建置高效能資料管道，例如串流分析、資料整合、視覺化和關鍵任務業務應用程式。

> **Note**
>
> 本主題假設您熟悉 Apache Kafka 平台及相關概念。如需有關 Apache Kafka 的詳細資訊，請參閱 Apache Kafka。不支援 MSK Serverless。MSK Serverless 叢集只能透過 Apache Kafka 規則動作目前不支援的 IAM 身分驗證來完成。如需如何使用 AWS IoT Core Confluent 設定 的詳細資訊，請參閱利用 Confluent 和 AWS 來解決 IoT 裝置和資料管理挑戰。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行 ec2:CreateNetworkInterface、ec2:DescribeNetworkInterfaces、、ec2:CreateNetworkI ec2:DescribeVpcAttribute和 ec2:DescribeSecurityGroups操作的 ec2:DescribeSubnetsIAM 角色。此角色會建立並管理您 Amazon Virtual Private Cloud 的彈性網路介面，以達到您的 Kafka 代理程式。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT Core 以允許 執行此規則動作。

  如需有關網路介面的詳細資訊，請參閱《Amazon EC2 使用者指南》中的彈性網路介面。

  連接至您指定之角色的政策應如下列範例所示：

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
```

```
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute",
            "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

- 如果您使用 AWS Secrets Manager 來存放連線至 Kafka 代理程式所需的登入資料，則必須建立 AWS IoT Core 可擔任的 IAM 角色，以執行 `secretsmanager:GetSecretValue`和 `secretsmanager:DescribeSecret`操作。

  連接至您指定之角色的政策應如下列範例所示：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_client_truststore-*",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_keytab-*"
            ]
        }
    ]
}
```

- 您可在 Amazon Virtual Private Cloud (Amazon VPC) 內部執行您的 Apache Kafka 叢集。您必須建立 Apache Kafka Virtual Private Cloud (VPC) 目的地，並使用子網路中的 NAT 閘道，將訊息從 轉送

AWS IoT 到公有 Kafka 叢集。 AWS IoT 規則引擎會在目的地中列出的每個子網路中建立網路介面，將流量直接路由到 VPC。當您到達目的地時， AWS IoT 規則引擎會自動建立 VPC 規則動作。如需有關 VPC 規則動作的詳細資訊，請參閱 Apache Kafka 虛擬私有雲端 (VPC) 目的地。

- 如果您使用客戶受管 AWS KMS key (KMS 金鑰） 來加密靜態資料，服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱《Amazon Managed Streaming for Apache Kafka 開發人員指南》中的 Amazon MSK 加密。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

destinationArn

Apache Kafka Virtual Private Cloud (VPC) 目的地的 Amazon Resource Name (ARN)。如需建立目的地的詳細資訊，請參閱 Apache Kafka 虛擬私有雲端 (VPC) 目的地。

主題

要傳送至 Kafka 代理程式之訊息的 Kafka 主題。

您可使用替代範本來替代此欄位。如需詳細資訊，請參閱the section called “替代範本”。

金鑰 (選用)

Kafka 訊息金鑰。

您可使用替代範本來替代此欄位。如需詳細資訊，請參閱the section called “替代範本”。

標頭 (選用)

您指定的 Kafka 標頭清單。每個標頭都是鍵值對，您可以在建立 Kafka 動作時指定。您可以使用這些標頭將資料從 IoT 用戶端路由到下游 Kafka 叢集，而不需修改訊息承載。

您可使用替代範本來替代此欄位。若要了解如何在 Kafka 動作標頭中傳遞內嵌規則的函數作為替換範本，請參閱範例。如需詳細資訊，請參閱the section called “替代範本”。

> **Note**
>
> 不支援二進位格式的標頭。

分割區 (選用)

Kafka 訊息分割區。

您可使用替代範本來替代此欄位。如需詳細資訊，請參閱the section called “替代範本”。

clientProperties

定義 Apache Kafka 生產者用戶端屬性的物件。

acks (選用)

生產者要求伺服器在考慮請求完成之前所收到的確認數目。

如果您指定 0 作為值，生產者將不會等待伺服器的任何確認。若伺服器並未收到訊息，生產者不會重試傳送訊息。

有效值：`-1`、`0`、`1`、`all`。預設值為 `1`。

bootstrap.servers

用來建立 Kafka 叢集初始連線的主機和連接埠配對清單 (例如 `host1:port1`、`host2:port2`)。

compression.type (選用)

生產者所產生所有資料的壓縮類型。

有效值：`none`、`gzip`、`snappy`、`lz4`、`zstd`。預設值為 none。

security.protocol

用來連接至您 Kafka 代理程式的安全通訊協定。

有效值：`SSL`、`SASL_SSL`。預設值為 `SSL`。

key.serializer

指定如何將與 `ProducerRecord` 一起提供的金鑰物件轉換為位元組。

有效值：`StringSerializer`。

value.serializer

指定如何將與 `ProducerRecord` 一起提供的值物件轉換為位元組。

有效值：`ByteBufferSerializer`。

ssl.truststore

base64 格式的信任庫檔案或在 AWS Secrets Manager 中的信任庫檔案位置。若 Amazon 憑證授權機構 (CA) 信任您的信任存放區，則不需要此值。

此欄位支援替代範本。若您使用 Secrets Manager 儲存連線至 Kafka 代理程式所需的憑證，則可使用 `get_secret` SQL 函數來擷取此欄位的值。如需替代範本的詳細資訊，請參閱 the section called “替代範本”。如需 `get_secret` SQL 函數的詳細資訊，請參閱 the section called “get_secret(secretId, secretType, key, roleArn)”。若信任庫為檔案形式，請使用 `SecretBinary` 參數。若信任庫為字串形式，請使用 `SecretString` 參數。

此值的最大大小為 65 KB。

ssl.truststore.password

信任庫的密碼。只有在您已建立信任庫的密碼時，才需要此值。

ssl.keystore

金鑰存放區檔案。當您指定 `SSL` 為 `security.protocol` 的值時，則需要此值。

此欄位支援替代範本。使用 Secrets Manager 來存放連接至 Kafka 代理程式所需的憑證。若要擷取此欄位的值，請使用 `get_secret` SQL 函數。如需替代範本的詳細資訊，請參閱 the section called “替代範本”。如需 `get_secret` SQL 函數的詳細資訊，請參閱 the section called “get_secret(secretId, secretType, key, roleArn)”。使用 `SecretBinary` 參數。

ssl.keystore.password

金鑰存放區檔案的存放區密碼。若指定 `ssl.keystore` 的值，則需要此值。

此欄位的值可以是純文字。此欄位也支援替代範本。使用 Secrets Manager 來存放連接至 Kafka 代理程式所需的憑證。若要擷取此欄位的值，請使用 `get_secret` SQL 函數。如需替代範本的詳細資訊，請參閱 the section called “替代範本”。如需 `get_secret` SQL 函數的詳細資訊，請參閱 the section called “get_secret(secretId, secretType, key, roleArn)”。使用 `SecretString` 參數。

ssl.key.password

金鑰存放區檔案中私鑰的密碼。

此欄位支援替代範本。使用 Secrets Manager 來存放連接至 Kafka 代理程式所需的憑證。若要擷取此欄位的值，請使用 `get_secret` SQL 函數。如需替代範本的詳細資訊，請參閱 the

section called “替代範本”。如需 get_secret SQL 函數的詳細資訊，請參閱 the section called “get_secret(secretId, secretType, key, roleArn)”。使用 SecretString 參數。

sasl.mechanism

用來連接至 Kafka 代理程式的安全機制。指定 security.protocol 的 SASL_SSL 時，則需要此值。

有效值：PLAIN、SCRAM-SHA-512、GSSAPI。

> **Note**
>
> SCRAM-SHA-512 是 cn-north-1、cn-northwest-1、us-gov-east-1 和 us-gov-west-1 區域中唯一支援的安全機制。

sasl.plain.username

用來從 Secrets Manager 擷取秘密字串的使用者名稱。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 PLAIN 時，則需要此值。

sasl.plain.password

用於從 Secrets Manager 擷取秘密字串的密碼。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 PLAIN 時，則需要此值。

sasl.scram.username

用來從 Secrets Manager 擷取秘密字串的使用者名稱。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 SCRAM-SHA-512 時，則需要此值。

sasl.scram.password

用於從 Secrets Manager 擷取秘密字串的密碼。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 SCRAM-SHA-512 時，則需要此值。

sasl.kerberos.keytab

Secrets Manager 中 Kerberos 驗證的 keytab 檔案。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 GSSAPI 時，則需要此值。

此欄位支援替代範本。使用 Secrets Manager 來存放連接至 Kafka 代理程式所需的憑證。若要擷取此欄位的值，請使用 get_secret SQL 函數。如需替代範本的詳細資訊，請參閱 the

section called “替代範本”。如需 get_secret SQL 函數的詳細資訊，請參閱 the section called “get_secret(secretId, secretType, key, roleArn)”。使用 SecretBinary 參數。

sasl.kerberos.service.name

在 Apache Kafka 執行之下的 Kerberos 委託人名稱。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 GSSAPI 時，則需要此值。

sasl.kerberos.krb5.kdc

您的 Apache Kafka 生產者用戶端連線至金鑰分配中心 (KDC) 的主機名稱。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 GSSAPI 時，則需要此值。

sasl.kerberos.krb5.realm

您的 Apache Kafka 生產者用戶端連線的領域。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 GSSAPI 時，則需要此值。

sasl.kerberos.principal

Kerberos 可指派票證來存取 Kerberos 感知服務的唯一 Kerberos 身分。指定 security.protocol 的 SASL_SSL 和 sasl.mechanism 的 GSSAPI 時，則需要此值。

範例

下列 JSON 範例定義 AWS IoT 規則中的 Apache Kafka 動作。以下範例會在 Kafka 動作標頭中傳遞 sourceIp() 內嵌函數作為替換範本。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "kafka": {
     "destinationArn": "arn:aws:iot:region:123456789012:ruledestination/vpc/
VPCDestinationARN",
     "topic": "TopicName",
     "clientProperties": {
      "bootstrap.servers": "kafka.com:9092",
      "security.protocol": "SASL_SSL",
      "ssl.truststore": "${get_secret('kafka_client_truststore',
 'SecretBinary','arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
```

```
      "ssl.truststore.password": "kafka password",
      "sasl.mechanism": "GSSAPI",
      "sasl.kerberos.service.name": "kafka",
      "sasl.kerberos.krb5.kdc": "kerberosdns.com",
      "sasl.kerberos.keytab": "${get_secret('kafka_keytab','SecretBinary',
 'arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "sasl.kerberos.krb5.realm": "KERBEROSREALM",
      "sasl.kerberos.principal": "kafka-keytab/kafka-keytab.com"
     },
     "headers": [
      {
       "key": "static_header_key",
       "value": "static_header_value"
      },
      {
       "key": "substitutable_header_key",
       "value": "${value_from_payload}"
      },
      {
       "key": "source_ip",
       "value": "${sourceIp()}"
      }
     ]
    }
   }
  ]
 }
}
```

關於您 Kerberos 設定的重要注意事項

- 您的金鑰分配中心 (KDC) 必須透過目標 VPC 內的私人網域名稱系統 (DNS) 進行解析。其中一種可能的方法是將 KDC DNS 項目新增至私有託管區域。如需此方法的詳細資訊，請參閱使用私有託管區域。
- 每個 VPC 皆必須啟用 DNS 解析。如需詳細資訊，請參閱以 VPC 使用 DNS。
- VPC 目的地中的網路介面安全群組和執行個體層級安全群組必須允許 VPC 內下列連接埠上的流量。
  - 引導代理程式接聽程式連接埠上的 TCP 流量 (通常是 9092，但必須在 9000-9100 範圍內)
  - KDC 連接埠 88 上的 TCP 和 UDP 流量
- SCRAM-SHA-512 是 cn-north-1、cn-northwest-1、us-gov-east-1 和 us-gov-west-1 區域中唯一支援的安全機制。

Apache Kafka 虛擬私有雲端 (VPC) 目的地

Apache Kafka 規則動作將資料路由至 Amazon Virtual Private Cloud (Amazon VPC) 中的 Apache Kafka 叢集。在為規則動作指定 VPC 目的地時，系統會自動啟用 Apache Kafka 規則動作所使用的 VPC 組態。

Apache Kafka Virtual Private Cloud (VPC) 目的地包含 VPC 內的子網路清單。規則引擎會在您於此清單中指定的每個子網路中建立彈性網路介面。如需有關網路介面的詳細資訊，請參閱《Amazon EC2 使用者指南》中的彈性網路介面。

需求和考量事項

- 如果您使用的是將由公有端點透過網際網路存取的自我管理 Apache Kafka 叢集：
  - 為子網路中的執行個體建立 NAT 閘道。NAT 閘道具有可連接到網際網路的公有 IP 地址，允許規則引擎將您的訊息轉發到公有 Kafka 叢集。
  - 使用 Apache Kafka Virtual Private Cloud (VPC) 目的地建立的彈性網路介面 (ENIs) 配置彈性 IP 地址。您使用的安全群組必須配置為封鎖傳入流量。

    > **Note**
    >
    > 如果 Apache Kafka Virtual Private Cloud (VPC) 目的地已停用，然後重新啟用，您必須重新建立彈性 IPs與新 ENIs關聯。

- 如果 Apache Kafka Virtual Private Cloud (VPC) 目的地連續 7 天未收到任何流量，則會停用。
- 如果 Apache Kafka Virtual Private Cloud (VPC) 目的地使用的任何資源變更，則會停用目的地且無法使用。
- 可以停用 Apache Kafka Virtual Private Cloud (VPC) 目的地的一些變更包括：
  - 刪除 VPC、子網路、安全群組或使用的角色。
  - 將角色修改為不再具有必要的許可。
  - 達到接近子網路容量，這使得我們無法套用 FedRAMP 修補。
  - 停用目的地。

定價

基於定價目的，除了資源位於您 VPC 中時將訊息傳送至資源之外，還會計量 VPC 規則動作。如需定價資訊，請參閱 AWS IoT Core 定價。

建立 Apache Kafka Virtual Private Cloud (VPC) 目的地

您可以使用 CreateTopicRuleDestination API 或 AWS IoT Core 主控台建立 Apache Kafka Virtual Private Cloud (VPC) 目的地。

建立目的地時，您必須指定下列資訊。

vpcId

Amazon VPC 的唯一 ID。

subnetIds

規則引擎在其中建立彈性網路介面的子網路清單。規則引擎會為清單中的每個子網路配置一個單一網路介面。

> **Note**
>
> 建立目的地時，您無法在 use1-az3 可用區域中使用子網路。

securityGroups (選用)

套用至網路介面的安全性群組清單。

roleArn

有權代您建立網路介面的角色的 Amazon 資源名稱 (ARN)

此 ARN 應連接至一個看似於下列範例的政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
```

```
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/VPCDestinationENI": "true"
                }
        }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateNetworkInterface",
                    "aws:RequestTag/VPCDestinationENI": "true"
                }
        }
        }
    ]
}
```

使用 建立 Apache Kafka Virtual Private Cloud (VPC) 目的地 AWS CLI

下列範例顯示如何使用 建立目的地 AWS CLI。

```
aws --region regions iot create-topic-rule-destination --destination-configuration
 'vpcConfiguration={subnetIds=["subnet-
123456789101230456"],securityGroups=[],vpcId="vpc-
123456789101230456",roleArn="arn:aws:iam::123456789012:role/role-name"}'
```

執行此命令後，目的地狀態將為 IN_PROGRESS。幾分鐘後，其狀態會變更為 ERROR (若命令不成功) 或 ENABLED。目的地狀態為 ENABLED 時，即可使用。

您可以使用下列命令來取得 Apache Kafka Virtual Private Cloud (VPC) 目的地的狀態。

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

使用 AWS IoT Core 主控台建立 Apache Kafka Virtual Private Cloud (VPC) 目的地

下列步驟說明如何使用 AWS IoT Core 主控台建立目的地。

1. 導覽至 AWS IoT Core 主控台。在左窗格的動作索引標籤上，選擇目的地。
2. 請輸入下列欄位的值。
   - VPC ID
   - 子網路 ID
   - 安全群組
3. 選取具有建立網路介面所需許可的角色。上述範例政策包含這些許可。

當 Apache Kafka Virtual Private Cloud (VPC) 目的地狀態為 ENABLED 時，即可開始使用。

CloudWatch 警示

CloudWatch 警示 (`cloudWatchAlarm`) 動作變更 Amazon CloudWatch 警示的狀態。您可以指定此呼叫狀態變更的原因和值。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`cloudwatch:SetAlarmState`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`alarmName`

CloudWatch 警示名稱。

僅支援替代範本：API 和 AWS CLI

`stateReason`

警示變更的原因。

支援替代範本：是

`stateValue`

警示狀態的值。有效值：`OK`、`ALARM`、`INSUFFICIENT_DATA`。

支援替代範本：是

`roleArn`

允許存取 CloudWatch 警示的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下面的 JSON 範例定義 AWS IoT 規則中的 CloudWatch 警示動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchAlarm": {
                    "alarmName": "IotAlarm",
                    "stateReason": "Temperature stabilized.",
                    "stateValue": "OK",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
```

```
}
```

另請參閱

- 《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？
- 《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 警示

CloudWatch Logs

CloudWatch Logs (`cloudwatchLogs`) 動作會將資料傳送至 Amazon CloudWatch Logs。您可以使用 `batchMode`，以一則訊息上傳多個裝置日誌記錄並加上時間戳記。您也可以指定動作傳送資料的日誌群組。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行 `logs:CreateLogStream`、 `logs:DescribeLogStreams`和 `logs:PutLogEvents`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用客戶受管 AWS KMS key (KMS 金鑰） 加密 CloudWatch Logs 中的日誌資料，則服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中使用 AWS KMS加密 CloudWatch Logs 中的日誌資料。

`batchMode` 的 MQTT 訊息格式需求

如果您在關閉 `batchMode` 的情況下使用 CloudWatch Logs 規則動作，則不需要 MQTT 訊息格式設定。(注意：`batchMode` 參數的預設值為 `false`。) 不過，如果您在 `batchMode` 開啟的情況下使用 CloudWatch Logs 規則動作 (參數值為 `true`)，則包含裝置端日誌的 MQTT 訊息必須格式化為包含時間戳記和訊息承載。注意：`timestamp` 代表事件發生的時間，以 1970 年 1 月 1 日 00:00:00 UTC 起算的毫秒數表示。

以下是發佈格式的範例：

```
[
```

```
  {"timestamp": 1673520691093, "message": "Test message 1"},
  {"timestamp": 1673520692879, "message": "Test message 2"},
  {"timestamp": 1673520693442, "message": "Test message 3"}
]
```

根據裝置端日誌的產生方式，這些日誌可能需要先篩選並重新格式化才能傳送，以符合此需求。如需詳細資訊，請參閱 MQTT 訊息承載。

與 batchMode 參數無關，message內容必須符合 AWS IoT 訊息大小限制。如需詳細資訊，請參閱 AWS IoT Core 端點和配額。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`logGroupName`

其動作傳送資料的 CloudWatch 日誌群組。

AWS CLI 僅支援替代範本：API 和

`roleArn`

允許存取 CloudWatch Logs 群組的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

（選用）`batchMode`

指出是否要擷取日誌記錄批次並上傳至 CloudWatch。值包括 `true` 或 `false` (預設值)。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例會在 AWS IoT 規則中定義 CloudWatch Logs 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
```

```
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchLogs": {
                    "logGroupName": "IotLogs",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",
                    "batchMode": false
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon CloudWatch Logs 使用者指南》中的什麼是 Amazon CloudWatch Logs？

CloudWatch 指標

CloudWatch 指標 (`cloudwatchMetric`) 動作擷取 Amazon CloudWatch 指標。您可以指定指標命名空間、名稱、值、單位、時間戳記。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`cloudwatch:PutMetricData`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`metricName`

CloudWatch 指標名稱。

支援替代範本：是

`metricNamespace`

CloudWatch 指標命名空間名稱。

支援替代範本：是

`metricUnit`

CloudWatch 支援的指標單位。

支援替代範本：是

`metricValue`

包含 CloudWatch 指標值的字串。

支援替代範本：是

`metricTimestamp`

(選用) 包含 Unix epoch 時間中時間戳記 (以秒為單位來表達) 的字串。預設為目前的 Unix epoch 時間。

支援替代範本：是

`roleArn`

允許存取 CloudWatch 指標的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下面的 JSON 範例定義 AWS IoT 規則中的 CloudWatch 指標動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "IotMetric",
                    "metricNamespace": "IotNamespace",
                    "metricUnit": "Count",
```

```
                    "metricValue": "1",
                    "metricTimestamp": "1456821314",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 CloudWatch 指標動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "${topic()}",
                    "metricNamespace": "${namespace}",
                    "metricUnit": "${unit}",
                    "metricValue": "${value}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon CloudWatch 使用者指南》中的什麼是 Amazon CloudWatch？
- 《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 指標

DynamoDB

DynamoDB (dynamoDB) 動作會將 MQTT 訊息的全部或部分寫入 Amazon DynamoDB 表格。

您可依循對您展示如何使用 DynamoDB 動作 來建立及測試規則的教學課程。如需詳細資訊，請參閱教學課程：將裝置資料儲存在 DynamoDB 表格中。

> **Note**
>
> 此規則會以二進位資料形式將非 JSON 資料寫入 DynamoDB 中。DynamoDB 主控台會以 Base64 編碼文字來顯示資料。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`dynamodb:PutItem`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用客戶受管 AWS KMS key (KMS 金鑰） 來加密 DynamoDB 中的靜態資料，則服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱《Amazon DynamoDB 入門指南》中的客戶管理 KMS 金鑰。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`tableName`

DynamoDB 資料表的名稱。

僅支援替代範本：API 和 AWS CLI

`hashKeyField`

雜湊索引鍵 (也稱為分割區索引鍵) 的名稱。

僅支援替代範本：API 和 AWS CLI

`hashKeyType`

(選用) 雜湊索引鍵 (也稱為分割區索引鍵) 的資料類型。有效值：`STRING`、`NUMBER`。

僅支援替代範本：API 和 AWS CLI

`hashKeyValue`

雜湊索引鍵的值。考慮使用替代範本，例如 `${topic()}` 或 `${timestamp()}`。

支援替代範本：是

`rangeKeyField`

(選用) 範圍索引鍵 (亦稱為排序索引鍵) 的名稱。

僅支援替代範本：API 和 AWS CLI

`rangeKeyType`

(選用) 範圍索引鍵 (亦稱為排序索引鍵) 的資料類型。有效值：`STRING`、`NUMBER`。

僅支援替代範本：API 和 AWS CLI

`rangeKeyValue`

(選用) 範圍索引鍵的值。考慮使用替代範本，例如 `${topic()}` 或 `${timestamp()}`。

支援替代範本：是

`payloadField`

(選用) 承載寫入的欄名稱。若省略此值，則會將承載寫入名為 `payload` 的欄。

支援替代範本：是

`operation`

(選用) 欲執行的作業類型。有效值：`INSERT`、`UPDATE`、`DELETE`。

支援替代範本：是

`roleARN`

允許存取 DynamoDB 資料表的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

寫入 DynamoDB 表格的資料產自於規則的 SQL 陳述式。

範例

下列 JSON 範例會在 AWS IoT 規則中定義 DynamoDB 動作。

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDB": {
                    "tableName": "my_ddb_table",
                    "hashKeyField": "key",
                    "hashKeyValue": "${topic()}",
                    "rangeKeyField": "timestamp",
                    "rangeKeyValue": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon DynamoDB 開發人員指南》中的什麼是 Amazon DynamoDB？
- 《Amazon DynamoDB 開發人員指南》中的 DynamoDB 入門
- 教學課程：將裝置資料儲存在 DynamoDB 表格中

DynamoDBv2

DynamoDBv2 (dynamoDBv2) 動作會將 MQTT 訊息的全部或部分寫入 Amazon DynamoDB 表格。承載中的每個屬性都會寫入 DynamoDB 資料庫中不同的欄。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`dynamodb:PutItem`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- MQTT 訊息承載一定要包含一個與該表格主要分割區索引鍵相符的根層級索引鍵，以及一個與該表格的主要排序索引鍵相符的根層級索引鍵 (若有定義)。

- 如果您使用客戶受管 AWS KMS key (KMS 金鑰） 來加密 DynamoDB 中的靜態資料，則服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱《Amazon DynamoDB 入門指南》中的客戶管理 KMS 金鑰。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`putItem`

指定訊息資料將寫入的 DynamoDB 表格物件。此物件必須包含下列資訊：

`tableName`

DynamoDB 資料表的名稱。

僅支援替代範本：API 和 AWS CLI

`roleARN`

允許存取 DynamoDB 資料表的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

寫入 DynamoDB 表格的資料產自於規則的 SQL 陳述式。

範例

下列 JSON 範例會在 AWS IoT 規則中定義 DynamoDBv2 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "my_ddb_table"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2",
```

```
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 DynamoDB 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2015-10-08",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "${topic()}"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon DynamoDB 開發人員指南》中的什麼是 Amazon DynamoDB？
- 《Amazon DynamoDB 開發人員指南》中的 DynamoDB 入門

Elasticsearch

Elasticsearch (`elasticsearch`) 動作將資料從 MQTT 訊息寫入 Amazon OpenSearch Service 網域。您可於之後使用如 OpenSearch 儀表板等工具來查詢及視覺化 OpenSearch 中的資料。

> **Warning**
>
> `Elasticsearch` 動作只能由現有規則動作使用。若要建立新的規則動作或更新現有的規則動作，請改用 `OpenSearch` 規則動作。如需詳細資訊，請參閱OpenSearch。

需求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`es:ESHttpPut`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用客戶受管 AWS KMS key (KMS 金鑰） 來加密 OpenSearch 中的靜態資料，則服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱 《Amazon OpenSearch Service 開發人員指南》 中 Amazon OpenSearch Service 的靜態資料加密。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`endpoint`

您的服務網域端點。

僅支援替代範本：API 和 AWS CLI

`index`

您想儲存資料的索引。

支援替代範本：是

`type`

欲存放文件的類型。

支援替代範本：是

`id`

各文件的專屬識別符。

支援替代範本：是

`roleARN`

IAM 角色，允許存取 OpenSearch Service 網域。如需詳細資訊，請參閱需求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 Elasticsearch 動作，以及如何指定`elasticsearch`動作的欄位。如需詳細資訊，請參閱 ElasticsearchAction。

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "my-type",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 Elasticsearch 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
```

```
            }
        ]
    }
}
```

另請參閱

- OpenSearch
- 什麼是 Amazon OpenSearch Service？

HTTP

HTTPS (`http`) 動作會將資料從 MQTT 訊息傳送至 HTTPS 端點，而該端點可指向 Web 應用程式或服務。

要求

此規則動作具有下列需求：

- 您必須先確認並啟用 HTTPS 端點，才可讓規則引擎使用它們。如需詳細資訊，請參閱HTTP 動作目的地。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`url`

使用 HTTP POST 方法傳送訊息的 HTTPS 端點。若您使用 IP 地址取代主機名稱，其必須為 IPv4 地址。不支援 IPv6 地址取代主機名稱。若要透過 IPv6 接收資料，請使用發佈 AAAA 記錄且無 A 記錄的主機名稱。

如需 規則引擎用來連線至端點之 IP 版本的相關資訊，請參閱 網路存取。

支援替代範本：是

`confirmationUrl`

（選用）如果指定， AWS IoT 會使用確認 URL 來建立相符的主題規則目的地。您必須先啟用 HTTP 動作目的地，才能在 HTTP 動作中使用它。如需詳細資訊，請參閱HTTP 動作目

的地。如果您使用替代範本，您必須先手動建立 HTTP 動作目的地，才能使用 http動作。confirmationUrl必須是 的字首url。

url 與 confirmationUrl 之間的關係如下所述：

- 如果 url 為硬式編碼confirmationUrl且未提供，我們會隱含地將 url 欄位視為 confirmationUrl。 會 AWS IoT 建立 的主題規則目的地url。
- 如果 url和 confirmationUrl 為硬式編碼， url必須以 開頭confirmationUrl。 會 AWS IoT 建立 的主題規則目的地confirmationUrl。
- 如果 url 包含替代範本，則您必須指定 confirmationUrl 且 url 必須以 confirmationUrl 開頭。如果 confirmationUrl包含替代範本，您必須先手動建立 HTTP 動作目的地，才能使用 http動作。如果 confirmationUrl 不包含替代範本， 會為 AWS IoT 建立主題規則目的地confirmationUrl。

支援替代範本：是

headers

(選用) 要包含於端點之 HTTP 請求中的標頭清單。每個標頭必須包含下列資訊：

key

標頭的金鑰。

支援替代範本：否

value

標頭的值。

支援替代範本：是

> **Note**
>
> 當承載採用 JSON 格式時，預設內容類型為 application/json。否則，它是 application/octet-stream。您可以在標頭中使用重要的 content-type 指定確切的內容類型 (不區分大小寫) 來覆寫它。

auth

(選用) 規則引擎用來連線至 url 引數中所指定端點 URL 的身分驗證。目前，Signature 第 4 版是唯一支援的身分驗證類型。如需詳細資訊，請參閱 HTTP 授權。

支援替代範本：否

enableBatching

（選用）是否將 HTTP 動作訊息處理為特定 URL 的單一請求。值可以是 true 或 false。如需批次處理的詳細資訊，請參閱批次處理 HTTP 動作訊息。

布林值

支援替代範本：否

batchConfig

（選用）批次處理的組態設定。啟用後，必須指定batchConfig參數。如果未指定batchConfig參數，則會使用預設值。

maxBatchOpenMs

傳出訊息等待其他訊息建立批次的時間上限（以毫秒為單位）。設定越高，批次 HTTP 動作的延遲就越長。

最小值：5 毫秒。最大值：200 毫秒。

預設值：20 ms

支援替代範本：否

maxBatchSize

在單一動作執行中批次處理的訊息數量上限。

支援替代範本：否

最小值：2 則訊息。最大值：10 則訊息

預設值：10 則訊息

maxBatchSizeBytes

訊息批次的大小上限，以位元組為單位。

最小值：100 位元組。最大值：131，072 個位元組

預設值：5，120 個位元組

支援替代範本：否

`batchAcrossTopics`

是否允許將來自不同 MQTT 主題的訊息批次處理到單一 HTTP 請求。根據預設，只有來自相同主題的訊息會批次處理在一起。針對路由來自多個裝置主題的訊息目的地為相同 HTTP 端點的使用案例，啟用此參數。

預設值：false

支援替代範本：否

> **Note**
>
> 訊息一律會在相同帳戶、規則名稱、目標 HTTP 端點 URL 和帳單群組的範圍內批次處理。無論`batchAcrossTopics`設定為何，任何這些屬性中不同的訊息都不會合併為相同的批次。

> **Note**
>
> 當承載採用 JSON 格式時，預設內容類型為 application/json。否則，它是 application/octet-stream。您可以在標頭中使用重要的 content-type 指定確切的內容類型 (不區分大小寫) 來覆寫它。

範例

下列 JSON 範例使用 HTTP 動作定義 AWS IoT 規則。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
```

```
                    "headers": [
                        {
                            "key": "static_header_key",
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                        }
                    ]
                }
            }
        ]
    }
}
```

```
"http": {
    "url": "https://www.example.com/subpath",
    "confirmationUrl": "https://www.example.com",
    "headers": [
        {
            "key": "Content-Type",
            "value": "application/json"
          }
    ],
    "enableBatching": true,
    "batchConfig": {
      "maxBatchOpenMs": 123,
      "maxBatchSize": 5,
      "maxBatchSizeBytes": 131072,
     }
 },
 "errorAction": {
        "http": {
            "url": "https://www.example.com/subpath",
            "confirmationUrl": "https://www.example.com"
            // batchConfig is not allowed here
        }
}
```

HTTP 動作重試邏輯

AWS IoT 規則引擎會根據這些規則重試 HTTP 動作：

- 規則引擎會嘗試傳送訊息至少一次。
- 規則引擎最多重試兩次。嘗試次數上限為三次。
- 在下列情況下，規則引擎不會嘗試重試：
  - 上一次嘗試提供了大於 16,384 個位元組的回應。
  - 下游 Web 服務或應用程式在嘗試之後關閉 TCP 連線。
  - 完成請求的總時間超過請求逾時限制。
  - 請求傳回了 429、500-599 以外的 HTTP 狀態碼。

> **Note**
>
> 標準資料傳輸成本適用於重試。

另請參閱

- 批次處理 HTTP 動作訊息
- HTTP 動作目的地
- 部落格上的物聯網 AWS將資料直接從 路由 AWS IoT Core 到您的 Web 服務

批次處理 HTTP 動作訊息

您可以使用批次處理在單一請求中傳送多個 HTTP 動作訊息。

概觀

批次處理可讓您批次從 AWS IoT Core 規則引擎傳送訊息至 HTTP 端點。此功能可以透過減少 HTTP 動作執行的數量，以及透過減少與建立新連線相關聯的額外負荷來提高效率，從而協助降低成本。

> **Note**
>
> 批次 HTTP 動作會計量為單一動作。根據 AWS IoT Core 規則引擎對下游服務發出的傳出批次承載大小，以 5 kiB 的增量計量。如需詳細資訊，請參閱 AWS IoT Core 定價頁面。

當您在 IoT 規則動作的定義中啟用批次處理時，下列參數將可用於組態：

`maxBatchOpenMs`

傳出訊息等待其他訊息建立批次的時間上限（以毫秒為單位）。設定越高，批次 HTTP 動作的延遲就越長。

最小值：5 毫秒。最大值：200 毫秒。

預設值：20 ms

支援替代範本：否

`maxBatchSize`

在單一 IoT 規則動作執行中批次處理的訊息數量上限。

最小值：2 則訊息。最大值：10 則訊息

預設值：10 則訊息

支援替代範本：否

`maxBatchSizeBytes`

訊息批次的大小上限，以位元組為單位。

最小值：100 位元組。最大值：131，072 個位元組

預設值：5120 位元組

支援替代範本：否

`batchAcrossTopics`

是否允許將來自不同 MQTT 主題的訊息批次處理到單一 HTTP 請求。根據預設，只有來自相同主題的訊息會批次處理在一起。針對路由來自多個裝置主題的訊息目的地為相同 HTTP 端點的使用案例，啟用此參數。

預設值：false

支援替代範本：否

> **Important**
>
> 當您指定多個批次參數時，批次會在達到第一個限制時完成。例如，如果您將 100 毫秒指定為最大批次開啟時間，將 5 kiB 指定為最大批次大小，且規則引擎只會在 100 毫秒內批次處理 2 kiB，則會建立並傳送 2 kiB 批次。

> **Note**
>
> 訊息一律會在相同帳戶、規則名稱、目標 HTTP 端點 URL 和帳單群組的範圍內批次處理。無論`batchAcrossTopics`設定為何，任何這些屬性中不同的訊息都不會合併為相同的批次。

在批次中使用 HTTP 標頭

當您在 HTTP 動作中使用標頭時，批次請求會使用新增至批次的最後一個訊息（不一定是您發佈的最後一個訊息）中的標頭值。我們建議使用下列其中一個標頭值：

- 批次中所有訊息的相同
- 適用於所有訊息（例如，身分驗證憑證）

標頭會與 HTTP 請求一起傳送，且不屬於訊息內文。

> **Note**
>
> 啟用批次處理時：
>
> - 批次請求會自動包含 `Content-Type: application/json`標頭，因為批次會以 JSON 陣列傳送。
> - 我們無法保證批次中的最後一個訊息是您發佈的最後一個訊息。這是最後一個訊息，讓它進入批次。

承載範例

下列範例顯示傳送至 HTTP 端點的批次訊息承載結構：

```
[
```

```
  {
    "user_id": "user1",
    "steps_today": 1000
  },
  {
    "user_id": "user2",
    "steps_today": 21000
  },
  {
    "user_id": "user8",
    "steps_today": 1500
  },
  ...
]
```

限制

以下是批次處理的限制：

- AWS IoT Core 不保證整體訊息排序。批次處理會在每個主機本機執行，這可能會導致批次中的訊息處理順序與收到的順序不同。
- AWS IoT Core 不提供接收者端的訊息處理支援。您有責任確保您的下游服務設定為接受並批次處理資料。
- 即使訊息目的地為相同的資源識別符 (HTTP URL 或資源 ARN)，也不支援跨帳戶批次處理。
- AWS IoT Core 不保證批次大小符合您指定的組態。根據時間和訊息流程，批次可能小於您設定的限制。
- 啟用批次處理時，不支援二進位承載（非 UTF-8 資料）。僅接受 UTF-8 文字承載（例如 JSON)。若要傳送二進位資料，請在傳送至 HTTP 動作之前先對其進行 base64 編碼，然後在接收端點對其進行解碼。例如，您可以使用 IoT 規則中的編碼函數來編碼二進位承載。或者，您可以在 IoT 裝置中編碼二進位承載並將其發佈至 AWS IoT Core。

批次處理的錯誤動作

您將無法在錯誤動作定義中定義個別的批次邏輯。不過，如果您已在主要動作中定義批次邏輯，您的錯誤動作將支援批次處理。

當批次請求失敗時， AWS IoT Core Rules 引擎將遵循 HTTP 動作重試邏輯。在最後一次重試嘗試之後，將會針對整個失敗的批次叫用錯誤動作。

以下是啟用批次處理的錯誤動作訊息範例：

```
{
    "ruleName": "FailedTopicRule",
    "topic": "topic/rulesengine",
    "payloadsWithMetadata": [
        {
            "id": 1,
            "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
        {
            "id": 2,
            "cloudwatchTraceId": "af94d3b8-0b18-1dbf-2c7d-513f5cb9e2e1",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
        {
            "id": 3,
            "cloudwatchTraceId": "ca441266-c2ce-c916-6aee-b9e5c7831675",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        }
    ],
    "failures": [
        {
            "affectedIds": [
                1,
                2,
                3
            ],
            "failedAction": "HttpAction",
            "failedResource": "https://example.foobar.com/HttpAction",
            "errorMessage": "HttpAction failed to make a request to the specified
 endpoint. StatusCode: 500. Reason: Internal Server Error."
        },
        {
            "affectedIds": [
```

```
                3
            ],
            "failedAction": "S3Action",
            "failedResource": "amzn-s3-demo-bucket",
            "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist"
        },
        {
            "affectedIds": [
                3
            ],
            "failedAction": "LambdaAction",
            "failedResource": "arn:aws:lambda:us-west-2:123456789012:function:dummy",
            "errorMessage": "Failed to invoke lambda function. Received Server error
 from Lambda. The error code is 403"
        }
    ]
}
```

啟用 batchAcrossTopics 時，錯誤動作承載格式會變更。topic 欄位會從頂層移至每個payloadsWithMetadata項目內部：

```
{
    "ruleName": "FailedTopicRule",
    "payloadsWithMetadata": [
        {
            "id": 1,
            "topic": "topic/sensor1",
            "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
        {
            "id": 2,
            "topic": "topic/sensor2",
            "cloudwatchTraceId": "af94d3b8-0b18-1dbf-2c7d-513f5cb9e2e1",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        }
```

```
    ],
    "failures": [
        {
            "affectedIds": [1, 2],
            "failedAction": "HttpAction",
            "failedResource": "https://example.foobar.com/HttpAction",
            "errorMessage": "HttpAction failed to make a request to the specified
 endpoint. StatusCode: 500. Reason: Internal Server Error."
        }
    ]
}
```

> **Note**
>
> 批次動作失敗也會產生較大的錯誤動作承載，這可能會增加因大小而造成錯誤動作失敗的機率。您可以使用 ErrorActionFailure 指標監控錯誤動作失敗。如需詳細資訊，請參閱規則動作指標。

使用 批次處理 HTTP 動作訊息 AWS CLI

使用批次處理建立或更新規則動作

1. 使用適當的 AWS CLI 命令來建立或更新規則：
   - 若要建立新的規則，請使用 create-topic-rule 命令：

     ```
     aws iot create-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

   - 若要更新現有規則，請使用 replace-topic-rule 命令：

     ```
     aws iot replace-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

2. 在主題規則承載中將 enableBatching 參數設定為 true，以啟用批次功能：

```
{
        "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
```

```
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                         }
                    ],
                    "enableBatching": true,
                    "batchConfig": {
                       "maxBatchOpenMs": 100,
                       "maxBatchSize": 5,
                       "maxBatchSizeBytes": 1024,
                       "batchAcrossTopics": true
                    }
                }
            }
      ]
}
```

3. 設定批次處理參數。您不需要指定所有批次參數。您可以選擇指定 1、2、3 或所有 4 個批次參數。如果您未指定批次參數，則 Rules Engine 會使用預設值更新該參數。如需批次處理參數及其預設值的詳細資訊，請參閱 HTTP 參數。

HTTP 動作目的地

HTTP 動作目的地是 Web 服務，規則引擎可以將來自主題規則的資料路由到該服務。 AWS IoT Core 資源說明 的 Web 服務 AWS IoT。目的地資源可以由不同的規則共用。

在 AWS IoT Core 可以將資料傳送到另一個 Web 服務之前，它必須確認它可以存取服務的端點。

概觀

HTTP 動作目的地是指支援確認 URL 和一或多個資料收集 URL 的 Web 服務。目的地資源包含 Web 服務的確認 URL。當您設定 HTTP 動作時，您可以指定應接收資料的端點實際 URL，以及 Web 服務

的確認 URL。在確認了您的目的地之後，主題規則會將 SQL 陳述式的結果傳送至 HTTPS 端點 (而非確認 URL)。

HTTP 動作目的地可以處於下列其中一種狀態：

ENABLED

目的地已確認，且可由規則動作使用。目的地必須處於 ENABLED 狀態，才能在規則中使用它。您只能啟用處於 DISABLED 狀態的目的地。

DISABLED

目的地已確認，但無法由規則動作使用。如果您想暫時防止流量傳至端點，而不必再次進行確認程序，這會很有用。您只能停用處於 ENABLED 狀態的目的地。

IN_PROGRESS

正在確認目的地。

ERROR

目的地確認逾時。

確認並啟用 HTTP 動作目的地之後，即可與帳戶中的任何規則搭配使用。

管理 HTTP 動作目的地

您可以使用下列操作來管理您的 HTTP 動作目的地。

建立 HTTP 動作目的地

您可以透過呼叫 `CreateTopicRuleDestination`操作或使用 AWS IoT 主控台來建立 HTTP 動作目的地。

建立目的地之後， 會將確認請求 AWS IoT 傳送至確認 URL。確認請求的格式如下：

```
HTTP POST {confirmationUrl}/?confirmationToken={confirmationToken}
Headers:
x-amz-rules-engine-message-type: DestinationConfirmation
x-amz-rules-engine-destination-arn:"arn:aws:iot:us-east-1:123456789012:ruledestination/
http/7a280e37-b9c6-47a2-a751-0703693f46e4"
Content-Type: application/json
Body:
{
```

```
    "arn":"arn:aws:iot:us-east-1:123456789012:ruledestination/http/7a280e37-b9c6-47a2-
a751-0703693f46e4",
    "confirmationToken": "AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "enableUrl": "https://iot.us-east-1.amazonaws.com/confirmdestination/
AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "messageType": "DestinationConfirmation"
}
```

確認請求的內容包含下列資訊：

arn

要確認之 HTTP 動作目的地的 Amazon Resource Name (ARN)。

confirmationToken

傳送的確認字符 AWS IoT Core。範例中的字符會被截斷。您的字符將更長。您需要此字符來使用 AWS IoT Core確認目的地。

enableUrl

您瀏覽以確認主題規則目的地的 URL。

messageType

訊息的類型。

確認 HTTP 動作目的地

若要完成端點確認程序，如果您使用的是 AWS CLI，您必須在確認 URL 收到確認請求後執行下列步驟。

1. 確認目的地已準備好接收訊息

   若要確認 HTTP 動作目的地已準備好接收 IoT 訊息，請在確認請求`enableUrl`中呼叫 ，或執行 `ConfirmTopicRuleDestination` API 操作並從`confirmationToken`確認請求傳遞 。
2. 將主題規則狀態設定為已啟用

   確認目的地可以接收訊息後，您必須執行 `UpdateTopicRuleDestination` API 操作，將主題規則的狀態設定為 `ENABLED`。

如果您使用的是 AWS IoT 主控台，請複製 `confirmationToken` 並將其貼到 AWS IoT 主控台的目的地確認對話方塊中。然後，您可以啟用 主題規則。

傳送新的確認請求

若要啟動目的地的新確認訊息，請呼叫 UpdateTopicRuleDestination，並將主題規則目的地的狀態設為 IN_PROGRESS。

在傳送新的確認請求之後重複確認程序。

停用和刪除 HTTP 動作目的地

若要停用目的地，請呼叫 UpdateTopicRuleDestination，並將主題規則目的地的狀態設為 DISABLED。處於 DISABLED 狀態的主題規則可以再次啟用，無需傳送新的確認請求。

若要刪除 HTTP 動作目的地，請呼叫 DeleteTopicRuleDestination。

憑證授權機構支援

> **Note**
>
> 不支援自我簽署憑證。

HTTP 動作目的地中的 HTTPS 端點支援AWS 私有憑證授權單位和 Lets Encrypt 發行的憑證。 https://letsencrypt.org/certificates/

網路存取

根據預設， AWS IoT 規則引擎會透過 IPv4 連線至 HTTPS 端點。如果您端點的主機名稱僅解析為 IPv6 地址，則規則引擎會改為透過 IPv6 與其連線。

當您的主機名稱發佈 AAAA 記錄且沒有 A 記錄時，就會發生這種情況。中沒有設定 AWS IoT Core 可控制此項目。您端點的 DNS 會決定規則引擎使用的版本。

> **Note**
>
> 目的地確認一律使用 IPv4。即使您打算透過 IPv6 接收資料，您的端點仍必須透過 IPv4 連線。
> IPv6

透過 IPv6 接收資料

1. 同時發佈端點主機名稱的 A 記錄和 AAAA 記錄。

2. 建立目的地並完成確認程序。如需詳細資訊，請參閱建立 HTTP 動作目的地及確認 HTTP 動作目的地。
3. 移除 A 記錄，只留下 AAAA 記錄。規則引擎會從此時開始透過 IPv6 連線。

如果您稍後將目的地的狀態設定為 IN_PROGRESS 以傳送新的確認請求，請先還原 A 記錄。確認請求目前無法到達僅發佈 AAAA 記錄的端點。

依來源地址限制傳入流量的端點，必須允許規則引擎的地址。規則引擎在每個區域中使用固定範圍的 IPv6 地址。如需詳細資訊，請參閱限制 HTTP 動作流量至規則引擎 IPv6 範圍。

InfluxDB

當您想要查詢裝置遙測以取得操作趨勢或即時監控時間序列資料時，請選擇此動作。您可以使用用戶端或伺服器端批次，將多個點合併在一個寫入請求中。 會將每個訊息 AWS IoT Core 轉換為 InfluxDB 線路通訊協定，並將其寫入指定的資料庫和資料表。如需 格式的詳細資訊，請參閱 InfluxData 文件中的 InfluxDB 行通訊協定。 InfluxData 如需受管叢集的資訊，請參閱 Amazon Timestream for InfluxDB。

在本主題中：

- 先決條件
- InfluxDB 動作目的地
- InfluxDB 術語映射
- Parameters
- 批次處理
- 陣列承載的每個元素範本
- InfluxDB 記錄內容
- 錯誤動作
- 範例

先決條件

此規則動作具有下列先決條件：

- InfluxDB 動作目的地 – 建立 InfluxDB 動作目的地，指定 InfluxDB 執行個體的端點、版本和登入資料。 會在傳送流量之前 AWS IoT Core 驗證端點擁有權。請參閱 InfluxDB 動作目的地。

- IAM 角色， AWS IoT 可擔任此角色來寫入 InfluxDB 資料庫，並在存放 InfluxDB 登入資料的秘密上執行 `GetSecretValue`操作。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。
- 存放在 中的 InfluxDB 登入 AWS Secrets Manager資料 – 對於 InfluxDB V3，Amazon Timestream for InfluxDB 會在您建立 InfluxDB 叢集時自動佈建 Secrets Manager 秘密。秘密包含叢集登入資料。對於 InfluxDB V2，從您的 InfluxDB 執行個體產生 All Access 或自訂 API 字符。將字符值儲存為 中的純文字秘密 AWS Secrets Manager。如需有關權杖建立的詳細資訊，請參閱 InfluxData 文件中的建立權杖。在執行時間，規則動作會呼叫 `secretsmanager:GetSecretValue` 來擷取這些登入資料，然後再驗證您的 InfluxDB 端點。
- 訊息承載中的時間戳記 – 訊息承載中的每個物件都必須包含名稱為 `timestamp`（區分大小寫） 且具有整數 Unix epoch 值的索引鍵。時間戳記單位也可以在 IoT 規則定義中設定。 AWS IoT Core 不會產生 InfluxDB 動作的時間戳記，除非 InfluxDB 用作錯誤動作。

> **Note**
>
> `time` 無法使用 `ts`或 等別名，而非 `timestamp`。

- HTTPS 連線 – 您的 InfluxDB 執行個體必須透過 HTTPS AWS IoT Core 連線。支援的傳出連接埠為：443、8443、8086 (InfluxDB V2 的預設連接埠） 和 8181 (InfluxDB V3 的預設連接埠）。
- JSON 格式承載 – InfluxDB 動作只會處理 JSON 承載。如果您的裝置發佈二進位或 Protobuf 資料，請在執行動作之前，使用規則 SQL 中的 `decode()`函數轉換為 JSON。

InfluxDB 動作目的地

您必須先建立 InfluxDB 動作目的地，才能使用 InfluxDB 規則動作。目的地會定義 InfluxDB 執行個體的連線參數。 AWS IoT Core 然後驗證端點擁有權。

建立目的地

使用 `CreateTopicRuleDestination` API 建立 InfluxDB 目的地：

```
{
  "destinationConfiguration": {
    "influxDBConfiguration": {
      "endpoint": "https://my-instance.timestream-influxdb.us-
west-2.amazonaws.com:8086",
      "influxDBVersion": "V2",
      "secretId": "arn:aws:secretsmanager:us-west-2:111122223333:secret:my-influxdb-
credentials-AbCdEf"
    }
```

```
  }
}
```

目的地參數

參數	Type	必要	描述
endpoint	String	是	InfluxDB 執行個體的 HTTPS 端點 URL。不支援 HTTP。支援的連接埠：443、8086、8181、8443。
influxDBVersion	String	是	InfluxDB 版本。有效值：V2、V3。
secretId	String	是	包含 InfluxDB 字符之 AWS Secrets Manager 秘密的名稱或 ARN。
secretType	String	否	秘密值的類型。有效值：SecretString 、SecretBinary 。
secretKey	String	否	秘密 JSON 中的金鑰，其中包含身分驗證字符。只有在秘密是具有多個金鑰的 JSON 物件時才需要。

端點擁有權驗證

當您建立 InfluxDB 動作目的地時， 會使用您提供的登入資料，針對 InfluxDB API 驗證 AWS IoT Core 端點擁有權：

- InfluxDB V2： AWS IoT Core 呼叫/api/v2/me端點。

- InfluxDB V3： AWS IoT Core 呼叫清單資料庫端點 (`GET /api/v3/configure/database`)。

成功回應 (2xx) 會將目的地狀態設定為 `ENABLED`。失敗時， 會將狀態 AWS IoT Core 設定為 `ERROR`。若要重試驗證，請呼叫 `UpdateTopicRuleDestination` 狀態設為 `IN_PROGRESS`。

InfluxDB 術語映射

下列術語在 InfluxDB V2 和 V3 之間不同。

AWS IoT Core 參數	InfluxDB V2 詞彙	InfluxDB V3 詞彙
`databaseName`	儲存貯體	資料庫
`tableName`	測量	資料表

> **Note**
>
> 如果您要從 Amazon Timestream 規則動作遷移， `dimensions` 參數會映射至 InfluxDB 動作中的標籤，而查詢結果屬性會映射至 欄位。

Parameters

當您使用 InfluxDB 動作建立 AWS IoT 規則時，您必須指定下列資訊：

`destinationArn`

InfluxDB 動作目的地的 ARN。請參閱 InfluxDB 動作目的地。支援替代範本：否

`roleArn`

授予 AWS IoT 許可以存取 Secrets Manager 秘密的 IAM 角色 ARN。請參閱 先決條件。支援替代範本：否

`databaseName`

要寫入記錄的 InfluxDB 資料庫名稱 （ 在 InfluxDB v2 中稱為儲存貯體，或在 InfluxDB v3 中稱為資料庫)。支援替代範本：否。若要將資料路由到不同的資料庫，請為每個資料庫建立個別的規則動作。

`tableName`

要寫入記錄的資料表名稱（在 InfluxDB v2 中稱為測量，或在 InfluxDB v3 中稱為資料表)。支援替代範本：是。

`organization`

InfluxDB 組織名稱。InfluxDB v2 的必要項目。如果您包含 InfluxDB v3 的此參數，則會予以忽略。支援替代範本：否。

`tags`

每個點的中繼資料，指定為映射。每個映射索引鍵都是標籤名稱，而每個映射值都是對應的標籤值。標籤會針對查詢效能編製索引。

- 在 InfluxDB V3 中，每個標籤名稱在資料表中必須是唯一的，而且不能複製欄位名稱。
- 標籤值支援訊息範圍和每個元素替換範本。

`timestampUnit`

承載中時間戳記值的精確度。有效值： `s`（秒）| `ms`（毫秒）| `us`（微秒）| `ns`（奈秒）。預設：`ms`。支援替代範本：否。

`batchConfig`

（選用）伺服器端批次處理組態。如需詳細資訊，請參閱批次處理。

- `maxBatchSize` – 每個批次中的點數上限。有效範圍：1–500。
- `maxBatchOpenMs` – 保持批次開啟的時間上限，以毫秒為單位。有效範圍：5–1，000。
- `maxBatchSizeBytes` – 排清前以位元組為單位的總大小上限。有效範圍：100–131，072。
- `batchAcrossTopics` – 布林值。當 時`true`，批次包含來自不同主題上訊息的點。預設：`false`。

批次處理

InfluxDB 動作支援兩種批次模式。

用戶端批次處理

您的 IoT 裝置會將時間序列資料批次處理為 JSON 陣列，並將其發佈為單一 MQTT 訊息。透過用戶端批次處理，每個陣列元素會在單一寫入請求中成為一個行通訊協定點。您不需要額外的組態。

伺服器端批次處理

在將個別訊息寫入 InfluxDB 之前，使用伺服器端批次處理來分組個別訊息。使用 batchConfig 參數設定伺服器端批次處理。批次會在先達到任何設定的限制 (maxBatchOpenMs、 maxBatchSize或 maxBatchSizeBytes) 時排清。

> **Note**
>
> 您可以同時設定伺服器端批次處理和用戶端批次處理 (JSON 陣列承載）。

InfluxDB 動作是根據傳出承載大小計量，增量為 5 KiB。也會測量線路通訊協定轉換失敗。

陣列承載的每個元素範本

當您的 IoT 裝置將批次時間序列資料作為 JSON 陣列傳送時，您可以使用「每個元素替代範本」來解析每個個別陣列元素的值。這會將每個資料點路由到不同的資料表，或套用元素特定的標籤。如需詳細資訊，請參閱AWS IoT 規則中陣列承載的每個元素替換。

兩個替代範本的語法

- ${expression} – 針對傳入的裝置訊息，在訊息範圍內解決 。每個訊息評估一次；相同值適用於陣列中的每個點。
- @{expression} – 針對規則 SQL SELECT 陳述式產生的承載的個別元素，在元素範圍內解決此問題。針對每個陣列元素重新評估，因此每個點都可以取得不同的值。如需詳細資訊，請參閱@{expression}參考。

使用 @{...} tableName和 標籤值，個別解析每個陣列元素的表達式。

範例

根據下列裝置承載 (JSON 陣列）：

```
[
  {"measurement_type": "temperature", "room": "kitchen", "timestamp": 1700000000000,
 "value": 23.5},
  {"measurement_type": "humidity", "room": "bedroom", "timestamp": 1700000001000,
 "value": 60.1}
]
```

以及下列動作組態：

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
abc123",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
    "tags": {
      "room": "@{room}"
    },
    "timestampUnit": "ms"
  }
}
```

產生的明細協定輸出包含兩個點：

```
temperature,room=kitchen value=23.5 1700000000000
humidity,room=bedroom value=60.1 1700000001000
```

> **Note**
>
> 參考的欄位`@{...}`會從明細行通訊協定欄位集中移除，因此也不會顯示為欄位。在此範例中，`measurement_type`會成為資料表名稱並`room`成為標籤，因此欄位集內不會出現 - `value` 是唯一的欄位。

限制

- `@{...}` 僅支援 InfluxDB 動作組態 (`tableName` 和 標籤值) 。
- 您無法`@{...}`在規則 SQL (SELECT/WHERE 子句) 、錯誤動作定義或任何其他規則動作中使用 。
- 內部僅支援欄位參考`@{...}`。不支援 函數。
- 每個值最多支援一個`@{...}`標記。單一值中的多個標記會產生 API 例外狀況。
- 您無法在相同的值`@{...}`中混合 `${...}`和 。混合會產生 API 例外狀況。
- 單一 JSON 物件視為單元素陣列。

InfluxDB 記錄內容

對於 post-SQL 查詢結果中的每個記錄，產生的 InfluxDB line-protocol 點包含下列元件：

元件	來源
資料表	`tableName` 參數值
Tags (標籤)	來自 `tags` 參數的鍵/值對
欄位	未用作標籤、資料表名稱或時間戳記的剩餘承載屬性
時間戳記	從承載擷取的`timestamp` 金鑰

資料類型轉換

AWS IoT Core 將 JSON 值轉換為 InfluxDB 行通訊協定類型，如下所示：

JSON 類型	線路通訊協定類型	範例
整數 (−263 至 263−1)	帶正負號的整數 (i 尾碼）	`42` → `42i`
整數 (263 到 264−1)	未簽章的整數 (u 尾碼）	`9223372036854775808` → `9223372036854775808u`
外部整數 (−263 至 264−1)	已拒絕 （已觸發錯誤動作）	—
浮點數/小數	IEEE-754 64 位元浮點數	`23.5` → `23.5`
Boolean	`t` 或 `f` *	`true` → `t`
String	引號字串	`"active"` → `"active"`
Null	已省略 （未寫入）	—
物件或陣列	壓縮的 JSON 字串 （空格已分割）	`{"a":1}` → `"{\"a\":1}"`

命名限制

- 欄位索引鍵和標籤索引鍵不可為空白或以底線 () 開頭_。
- 在 InfluxDB V3 中，資料表名稱和標籤索引鍵必須以字母或數字開頭。

- 欄位索引鍵、標籤索引鍵和標籤值中的逗號、等號和空格會自動逸出。
- 測量：逗號和空格會自動逸出。
- 在序列化之前，標籤會依索引鍵字母順序排序，以改善擷取效能。
- 從輸出省略空的標籤值。

預留金鑰

下列承載金鑰會在明細行通訊協定轉換期間從欄位集分割：

- `timestamp` – 用作點時間戳記。
- `tableName`（透過 `${...}`或 `@{...}`) 參考的金鑰 – 用作資料表名稱。
- 標籤值參考的索引鍵（透過 `${...}`或 `@{...}`) – 用作標籤值。

錯誤動作

如果 InfluxDB 動作失敗，則會觸發設定的錯誤動作。

使用 InfluxDB 做為錯誤動作

您可以將 InfluxDB 動作設定為任何規則的錯誤動作。整個承載會寫入單一`tableName`記錄。錯誤動作的 `tableName`和 支援訊息範圍替代範本 (`${...}`)`tags`。

錯誤動作輸出

觸發錯誤動作時，輸出承載包含：`ruleName`、`topic`、`cloudwatchTraceId`、`sourceIp`、、`clientId``base64OriginalPayload`(Base64 編碼的原始訊息），以及每個項目都有`failedAction`、 `failedResource`和 的`failures`陣列`errorMessage`。

透過伺服器端批次處理，承載會使用 `payloadsWithMetadata`，每個不同的傳入 MQTT 訊息各有一個項目。每個失敗`affectedIds`的值都是指這些訊息項目`id`的值；它們不是點索引。用戶端 JSON 陣列是一種傳入訊息，即使它會產生多個 InfluxDB 點。如需完整的承載格式，請參閱 批次處理的錯誤動作。

失敗案例	說明
目的地已停用或錯誤	InfluxDB 動作目的地未啟用。驗證端點擁有權驗證是否成功。

失敗案例	說明
無效目的地 ARN	指定的目的地不存在。
無效的角色 ARN	IAM 角色不存在或缺少許可。
秘密擷取失敗	秘密或設定的 `secretKey` 不存在，或規則動作角色無法擷取或解密秘密。
缺少時間戳記	承載不包含`timestamp` 金鑰。承載中的每個物件必須包含具有整數 Unix epoch 值`timestamp` 的欄位。
無效的時間戳記值	時間戳記值不是整數（例如字串、浮點數或 ISO-8601 日期）。值必須是 所指定單位中的整數 Unix epoch`timestampUnit` 。
無效的承載（無欄位）	承載在移除預留金鑰後，不包含行通訊協定的有效欄位。
欄位索引鍵無效	欄位金鑰為空白或以 開頭_。
用戶端批次包含無效的點	JSON 陣列中的一或多個元素行通訊協定驗證失敗。
標籤 + 欄位超過資料欄限制	合併的標籤和欄位索引鍵超過資料欄計數上限 (250)。
連線失敗	AWS IoT Core 無法連線至 InfluxDB 端點。
身分驗證失敗	InfluxDB 字符無效或已過期。更新 中的秘密 AWS Secrets Manager。
找不到資源	InfluxDB 中不存在指定的資料庫、資料表或組織。
欄位類型衝突	一或多個欄位與現有的結構描述衝突。整個批次寫入失敗。
InfluxDB 伺服器錯誤	InfluxDB 中發生內部錯誤。

失敗案例	說明
InfluxDB 服務無法使用	InfluxDB 暫時無法使用。規則引擎會以指數退避重試。

> **Important**
>
> 批次中任何點上的欄位類型衝突會導致整個批次寫入失敗。InfluxDB 不會部分遞交點 – 所有點成功或整個寫入遭到拒絕。

可重試錯誤 (503) 會以指數退避重試。對於 HTTP 401 回應， 會從 AWS IoT Core 重新載入字符 AWS Secrets Manager ，並重試一次請求。不可重試的錯誤 (404、422) 會立即觸發錯誤動作。如需重試限制，請參閱AWS IoT Core 服務配額。

範例

InfluxDB 規則動作

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'devices/+/telemetry'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "influxDB": {
          "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/
influxdb/a1b2c3d4",
          "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
          "organization": "my-org",
          "databaseName": "sensor_data",
          "tableName": "device_metrics",
          "tags": {
            "device_id": "${clientid()}",
            "location": "building-a"
          },
          "timestampUnit": "ms"
        }
      }
```

```
    ]
  }
}
```

範例承載：

```
{
  "timestamp": 1700000000000,
  "temperature": 23.5,
  "humidity": 60.1,
  "pressure": 1013.25,
  "battery_level": 87
}
```

產生的線路通訊協定：

```
device_metrics,device_id=myDevice123,location=building-a
 temperature=23.5,humidity=60.1,pressure=1013.25,battery_level=87i 1700000000000
```

輸出中的欄位順序可能會有所不同 — 欄位不會按字母順序排序。

具有每個元素範本的用戶端批次陣列承載

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
a1b2c3d4",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "organization": "my-org",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
    "tags": {
      "sensor_id": "@{sensor_id}",
      "location": "${topic(2)}"
    },
    "timestampUnit": "ns"
  }
}
```

範例承載（發佈至 **devices/floor3/telemetry**）：

```
[
```

```
  {"measurement_type": "temperature", "sensor_id": "sensor-42", "timestamp":
 1700000000000000000, "value": 23.5},
  {"measurement_type": "humidity", "sensor_id": "sensor-42", "timestamp":
 1700000001000000000, "value": 60.1},
  {"measurement_type": "pressure", "sensor_id": "sensor-43", "timestamp":
 1700000002000000000, "value": 1013.25}
]
```

產生的線路通訊協定：

```
temperature,location=floor3,sensor_id=sensor-42 value=23.5 1700000000000000000
humidity,location=floor3,sensor_id=sensor-42 value=60.1 1700000001000000000
pressure,location=floor3,sensor_id=sensor-43 value=1013.25 1700000002000000000
```

使用 InfluxDB 進行伺服器端批次處理

若要將伺服器端批次新增至任何 InfluxDB 動作，請在動作組態`batchConfig`中包含 ：

```
"batchConfig": {
  "maxBatchSize": 50,
  "maxBatchOpenMs": 1000,
  "maxBatchSizeBytes": 65536,
  "batchAcrossTopics": false
}
```

規則動作角色的 IAM 政策

信任政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {"Service": "iot.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

許可政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-west-2:111122223333:secret:my-influxdb-
secret-a1b2c3"
    }
  ]
}
```

合併用戶端和伺服器端批次處理 （點重新排序）

當您同時啟用用戶端批次處理 (JSON 陣列承載） 和伺服器端批次處理 (batchConfig) 時，請注意伺服器端批次可能會從用戶端批次處理承載重新排序點。規則引擎會將來自多個傳入訊息的點累積到單一伺服器端批次。由於訊息以非同步方式從不同的裝置或主題到達，因此在原始用戶端承載中排序的點可能會與最終寫入中其他訊息的點交錯。

動作組態：

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
abc123",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
    "tags": {
      "device_id": "${topic(2)}",
      "floor": "@{floor}"
    },
    "timestampUnit": "ms",
    "batchConfig": {
      "maxBatchSize": 100,
      "maxBatchOpenMs": 500,
      "maxBatchSizeBytes": 65536,
      "batchAcrossTopics": true
    }
  }
}
```

裝置 A devices/deviceA/telemetry會在時間 T 發佈至 ：

```
[
  {"measurement_type": "temperature", "floor": "1", "timestamp": 1700000000000,
 "value": 22.1},
  {"measurement_type": "temperature", "floor": "2", "timestamp": 1700000000100,
 "value": 23.4},
  {"measurement_type": "humidity", "floor": "1", "timestamp": 1700000000200, "value":
 55.0}
]
```

裝置 B 在 T+10ms devices/deviceB/telemetry 時發佈至 ：

```
[
  {"measurement_type": "temperature", "floor": "3", "timestamp": 1700000000050,
 "value": 21.8},
  {"measurement_type": "humidity", "floor": "3", "timestamp": 1700000000150, "value":
 62.3}
]
```

這兩個訊息都會在 500ms 批次時段 (maxBatchOpenMs) 內送達，因此規則引擎會將所有五個點合併為單一伺服器端批次。

產生的線路通訊協定 （伺服器端批次寫入）：

```
temperature,device_id=deviceB,floor=3 value=21.8 1700000000050
temperature,device_id=deviceA,floor=1 value=22.1 1700000000000
temperature,device_id=deviceA,floor=2 value=23.4 1700000000100
humidity,device_id=deviceB,floor=3 value=62.3 1700000000150
humidity,device_id=deviceA,floor=1 value=55.0 1700000000200
```

請注意，這些點不再依照每個用戶端承載中顯示的順序顯示。裝置 A 的三個點 （時間戳記 1700000000000、1700000000100、1700000000200) 與裝置 B 的兩個點 （時間戳記 1700000000050、1700000000150) 交錯。伺服器端批次不保證每個訊息中的原始排序。InfluxDB 使用時間戳記欄位將每個點放置在時間軸上，因此重新排序不會影響查詢正確性。不過，如果您的應用程式依賴寫入順序語意 （例如，在相同毫秒內處理欄位類型衝突或last-write-wins重複資料刪除），請注意，有效的寫入順序可能與發佈順序不同。

AWS IoT SiteWise

AWS IoT SiteWise (`iotSiteWise`) 動作會將資料從 MQTT 訊息傳送至其中的資產屬性 AWS IoT SiteWise。

您可以遵循教學課程，示範如何從 AWS IoT 物件擷取資料。如需詳細資訊，請參閱AWS IoT SiteWise《 使用者指南》中的從 AWS IoT 實物擷取資料 AWS IoT SiteWise教學課程或使用AWS IoT 核心規則擷取資料一節。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`iotsitewise:BatchPutAssetPropertyValue`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  您可連接下列範例信任政策至該角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        }
    ]
}
```

  若要改善安全性，您可以在 `Condition` 屬性中指定 AWS IoT SiteWise 資產階層路徑。下列範例是指定資產階層路徑的信任政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
```

```
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
            }
    ]
}
```

- 當您 AWS IoT SiteWise 使用此動作將資料傳送至 時，您的資料必須符合 `BatchPutAssetPropertyValue`操作的要求。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 BatchPutAssetPropertyValue。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`putAssetPropertyValueEntries`

資產屬性值項目的清單，其中每個項目都包含下列資訊：

`propertyAlias`

(選用) 與資產屬性相關聯的屬性別名。指定 `propertyAlias` 或同時指定 `assetId` 和 `propertyId`。如需詳細資訊，請參閱《AWS IoT SiteWise 使用者指南》中的將工業資料串流映射至資產屬性。

支援替代範本：是

`assetId`

（選用）AWS IoT SiteWise 資產的 ID。指定 `propertyAlias` 或同時指定 `assetId` 和 `propertyId`。

支援替代範本：是

`propertyId`

(選用) 該資產屬性的 ID。指定 `propertyAlias` 或同時指定 `assetId` 和 `propertyId`。

支援替代範本：是

`entryId`

(選用) 此項目的唯一識別符。定義 `entryId`，以更好地追蹤在發生失敗時哪則訊息造成錯誤。預設為新的 UUID。

支援替代範本：是

`propertyValues`

要插入的屬性值清單，其中每個屬性值包含以下格式的時間戳記、品質和值 (TQV)：

`timestamp`

包含下列資訊的時間戳記結構：

`timeInSeconds`

包含 Unix epoch 時間中時間 (以秒為單位) 的字串。如果您的消息承載沒有時間戳記，可以使用 timestamp()，返回目前時間 (以毫秒為單位)。若要將該時間轉換為以秒為單位，您可以使用下列替代範本：**`${floor(timestamp() / 1E3)}`**。

支援替代範本：是

`offsetInNanos`

(選用) 包含從該時間 (以秒為單位) 開始的奈秒時間位移的字串。如果您的消息承載沒有時間戳記，可以使用 timestamp()，返回目前時間 (以毫秒為單位)。若要計算從該時間開始的奈秒位移，可以使用下列替代範本：**`${(timestamp() % 1E3) * 1E6}`**。

支援替代範本：是

關於 Unix epoch 時間， 僅 AWS IoT SiteWise 接受過去最多 7 天內最多 5 分鐘的時間戳記項目。

`quality`

(選用) 說明數值品質的字串。有效值：`GOOD`、`BAD`、`UNCERTAIN`。

支援替代範本：是

`value`

包含下列其中一個值欄位的值結構，視資產屬性的資料類型而定：

`booleanValue`

(選用) 包含值項目之布林值的字串。

支援替代範本：是

`doubleValue`

(選用) 包含值項目之雙值的字串。

支援替代範本：是

`integerValue`

(選用) 包含值項目之整數值的字串。

支援替代範本：是

`stringValue`

(選用) 值項目的字串值。

支援替代範本：是

`roleArn`

授予 AWS IoT 許可的 IAM 角色 ARN，可將資產屬性值傳送至該角色 AWS IoT SiteWise。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下面的 JSON 範例定義 AWS IoT 規則中的基本 IoT SiteWise 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "/some/property/alias",
                            "propertyValues": [
                                {
                                    "timestamp": {
```

```
                                    "timeInSeconds": "${my.payload.timeInSeconds}"
                                },
                                "value": {
                                    "integerValue": "${my.payload.value}"
                                }
                            }
                        ]
                    }
                ],
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
            }
        }
    ]
  }
}
```

下列 JSON 範例定義 AWS IoT 規則中的 IoT SiteWise 動作。此範例使用此主題作為屬性別名和 `timestamp()` 函數。例如，如果您將資料發佈至 `/company/windfarm/3/turbine/7/rpm`，此動作會將資料傳送至屬性別名與您指定主題相同的資產屬性。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "${topic()}",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${floor(timestamp() / 1E3)}",
                                        "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                                    },
                                    "value": {
                                        "doubleValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
```

```
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

另請參閱

- 《AWS IoT SiteWise 使用者指南》中的什麼是 AWS IoT SiteWise？
- AWS IoT SiteWise 《 使用者指南》中的使用 AWS IoT Core 規則擷取資料
- AWS IoT SiteWise 《 使用者指南》中的AWS IoT SiteWise 從 AWS IoT 實物擷取資料至
- AWS IoT SiteWise 《 使用者指南》中的AWS IoT SiteWise 規則動作疑難排解

Firehose

Firehose(`firehose`) 動作會將資料從 MQTT 訊息傳送至 Amazon Data Firehose 串流。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`firehose:PutRecord`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用 Firehose 將資料傳送至 Amazon S3 儲存貯體，並使用受管客戶 AWS KMS key 來加密 AWS KMS Amazon S3 中的靜態資料，Firehose 必須能夠存取您的儲存貯體，並有權代表發起人使用 AWS KMS key 。如需詳細資訊，請參閱《Amazon Data Firehose 開發人員指南》中的授予 Firehose 對 Amazon S3 目的地的存取權。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`batchMode`

（選用）是否使用 以批次形式交付 Firehose `PutRecordBatch` 串流。預設值為 `false`。

若 batchMode 為 true 且規則的 SQL 陳述式評估為 Array，則每個 Array 元會在 PutRecordBatch 請求中形成一筆記錄。產生的陣列不能含有超過 500 條記錄。

支援替代範本：否

deliveryStreamName

要寫入訊息資料的 Firehose 串流。

僅支援替代範本：API 和 AWS CLI

separator

（選用）字元分隔符號，用於分隔寫入 Firehose 串流的記錄。若您略過此參數，則串流不會使用分隔符號。有效值：,(逗號)、\t (索引標籤)、\n (換行符號)，\r\n (Windows 換行)。

支援替代範本：否

roleArn

允許存取 Firehose 串流的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 Firehose 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "my_firehose_stream",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 Firehose 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

另請參閱

- 什麼是 Amazon Data Firehose？ Amazon Data Firehose 開發人員指南中的

Kinesis Data Streams

Kinesis Data Streams (`kinesis`) 動作 會將資料從 MQTT 訊息寫入 Amazon Kinesis Data Streams。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`kinesis:PutRecord`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用 AWS KMS 客戶受管 AWS KMS key (KMS 金鑰） 來加密 Kinesis Data Streams 中的靜態資料，則服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中使用使用者產生之 AWS KMS keys的許可。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`stream`

要寫入資料的 Kinesis 資料串流。

僅支援替代範本：API 和 AWS CLI

`partitionKey`

分割區索引鍵用來決定資料要寫入哪個碎片。分割區索引鍵通常是由表達式 (如 `${topic()}` 或 `${timestamp()}`) 組成。

支援替代範本：是

`roleArn`

授予存取 Kinesis 資料串流 AWS IoT 許可的 IAM 角色 ARN。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 Kinesis Data Streams 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "my_kinesis_stream",
                    "partitionKey": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 Kinesis 動作。

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "${topic()}",
                    "partitionKey": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon Kinesis Data Streams 開發人員指南》中的什麼是 Amazon Kinesis Data Streams？

Lambda

Lambda (`lambda`) 動作會叫用 AWS Lambda 函數，以非同步方式傳入 MQTT message. AWS IoT invokes Lambda 函數。

您可依循對您展示如何使用 Lambda 動作 來建立及測試規則的教學課程。如需詳細資訊，請參閱教學課程：使用 AWS Lambda 函數格式化通知。

需求

此規則動作具有下列需求：

- 若要 AWS IoT 讓 叫用 Lambda 函數，您必須設定授予 `lambda:InvokeFunction`許可的政策 AWS IoT。您只能叫用 Lambda 政策存在 AWS 區域 的相同 中定義的 Lambda 函數。Lambda 函數使用的是資源型政策，因此您必須將政策連接至 Lambda 函數本身。

  使用下列 AWS CLI 命令來連接授予 `lambda:InvokeFunction`許可的政策。在此命令中，取代：

  - *`function_name`* 與 Lambda 函數的名稱。您可以新增許可來更新函數的資源政策。
  - 具有 函數 AWS 區域 之 *`###`*。
  - *`account-id`*，其中包含定義規則的 AWS 帳戶 數字。

- *rule-name*，其中包含您定義 Lambda 動作之 AWS IoT 規則的名稱。
- *unique_id* 具有唯一的陳述式識別符。

> **Important**
>
> 如果您為 AWS IoT 委託人新增許可，但未提供 `source-arn`或 `source-account`，則任何使用 Lambda 動作建立規則 AWS 帳戶 的 都可以啟用規則來叫用您的 Lambda 函數 AWS IoT。

如需詳細資訊，請參閱 AWS Lambda 許可。

```
aws lambda add-permission \
    --function-name function_name \
    --region region \
    --principal iot.amazonaws.com \
    --source-arn arn:aws:iot:region:account-id:rule/rule_name \
    --source-account account-id
    --statement-id unique_id
    --action "lambda:InvokeFunction"
```

- 如果您使用 AWS IoT 主控台為 Lambda 規則動作建立規則，則會自動觸發 Lambda 函數。如果您 AWS CloudFormation 改搭配 使用 `AWS::IoT::TopicRule LambdaAction`，則必須新增 `AWS::lambda::Permission` 資源。然後，資源會授予您觸發 Lambda 函數的許可。

  下列程式碼顯示如何新增此資源的範例。在此範例中，取代：

  - *function_name* 與 Lambda 函數的名稱。
  - *##*與 函數 AWS 區域 的 。
  - *account-id*，其中包含定義規則的 AWS 帳戶 數字。
  - *rule-name*，其中包含您定義 Lambda 動作之 AWS IoT 規則的名稱。

```
Type: AWS::Lambda::Permission
Properties:
  Action: lambda:InvokeFunction
  FunctionName: !Ref function_name
  Principal: "iot.amazonaws.com"
  SourceAccount: account-id
  SourceArn: arn:aws:iot:region:account-id:rule/rule_name
```

- 如果您使用受管 AWS KMS 客戶 AWS KMS key 來加密 Lambda 中的靜態資料，服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的靜態加密。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`functionArn`

要叫用之 Lambda 函數的 ARN。 AWS IoT 必須具有叫用函數的許可。如需詳細資訊，請參閱需求。

如果您未針對您的 Lambda 函數指定版本或別名，則最新版的函數會關閉。如果想要關閉特定版本的 Lambda 函數，您可以指定版本或別名。如要指定版本或別名，請將版本或別名附加至 Lambda 函數的 ARN。

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

如需版本控制與別名的詳細資訊，請參閱 AWS Lambda 函數版本控制與別名。

僅支援替代範本：API 和 AWS CLI

範例

下列 JSON 範例會在 AWS IoT 規則中定義 Lambda 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
                }
            }
```

```
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本定義 Lambda 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

另請參閱

- 什麼是 AWS Lambda？《AWS Lambda 開發人員指南》中的
- 教學課程：使用 AWS Lambda 函數格式化通知

Location

Location (`location`) (位置) 動作會將您的地理位置資料傳送至 Amazon Location Service。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`geo:BatchUpdateDevicePosition`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色，以允許 AWS IoT 執行此規則動作。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`deviceId`

提供位置資料的裝置唯一 ID。如需詳細資訊，請參閱《Amazon Location Service API 參考》中的 `DeviceId`。

支援替代範本：是

`latitude`

此字串評估為雙精確度值，該值代表裝置位置的緯度。

支援替代範本：是

`longitude`

此字串評估為雙精確度值，該值代表裝置位置的經度。

支援替代範本：是

`roleArn`

IAM 角色，允許存取 Amazon Location Service 網域。如需詳細資訊，請參閱要求。

`timestamp`

對位置資料進行取樣的時間。預設值是處理 MQTT 訊息的時間。

`timestamp` 值包含以下兩個值：

- `value`：傳回一個很長的 Epoch 時間值的表達式。您可使用 the section called "time_to_epoch(String, String)" 函數，從訊息承載中傳遞的日期或時間建立有效的時間戳記。支援替代範本：是。
- `unit`：(選擇性) `value` 中所述運算式產生的 timestamp 值的精確度。有效值：`SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`。預設值為 `MILLISECONDS`。支援替代範本： AWS CLI 僅限 API 和 。

`trackerName`

在已更新位置的 Amazon Location 中的追蹤器資源名稱。如需詳細資訊，請參閱《Amazon Location Service 開發人員指南》中的追蹤器。

僅支援替代範本：API 和 AWS CLI

範例

下列 JSON 範例定義 AWS IoT 規則中的位置動作。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123454962127:role/service-role/ExampleRole",
     "trackerName": "MyTracker",
     "deviceId": "001",
     "sampleTime": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "-12.3456",
     "longitude": "65.4321"
    }
   }
  ]
 }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本定義位置動作。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123456789012:role/service-role/ExampleRole",
     "trackerName": "${TrackerName}",
     "deviceId": "${DeviceID}",
     "timestamp": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
```

```
     },
     "latitude": "${get(position, 0)}",
     "longitude": "${get(position, 1)}"
    }
   }
  ]
 }
}
```

下列 MQTT 承載範例顯示上述範例中的替代範本如何存取資料。您可以使用 get-device-position-history CLI 命令來驗證 MQTT 承載資料是否已在您的位置追蹤器中進行傳送。

```
{
 "TrackerName": "mytracker",
 "DeviceID": "001",
 "position": [
  "-12.3456",
  "65.4321"
 ]
}
```

```
aws location get-device-position-history --device-id 001 --tracker-name mytracker
```

```
{
 "DevicePositions": [
  {
   "DeviceId": "001",
   "Position": [
    -12.3456,
    65.4321
   ],
   "ReceivedTime": "2022-11-11T01:31:54.464000+00:00",
   "SampleTime": "2022-11-11T01:31:54.308000+00:00"
  }
 ]
}
```

另請參閱

- 在《Amazon Location Service 開發人員指南》中的什麼是 Amazon Location Service？。

OpenSearch

OpenSearch (`openSearch`) 動作將資料從 MQTT 訊息寫入 Amazon OpenSearch Service 網域。您可於之後使用如 OpenSearch 儀表板等工具來查詢及視覺化 OpenSearch Service 中的資料。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`es:ESHttpPut`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用受管客戶 AWS KMS key 來加密 OpenSearch Service 中的靜態資料，則該服務必須具有代表發起人使用 KMS 金鑰的許可。如需詳細資訊，請參閱 《Amazon OpenSearch Service 開發人員指南》 中 Amazon OpenSearch Service 的靜態資料加密。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`endpoint`

您 Amazon OpenSearch Service 網域的端點。

AWS CLI 僅支援替代範本：API 和

`index`

您想存放資料的 OpenSearch 索引。

支援替代範本：是

`type`

欲存放文件的類型。

> **Note**
>
> 對於 1.0 之後的 OpenSearch 版本， `type` 參數的值必須是 `_doc`。如需詳細資訊，請參閱 OpenSearch 文件。

支援替代範本：是

`id`

各文件的專屬識別符。

支援替代範本：是

`roleARN`

IAM 角色，允許存取 OpenSearch Service 網域。如需詳細資訊，請參閱要求。

支援替代範本：否

限制

OpenSearch (`openSearch`) 動作不能用於將資料傳送到 VPC Elasticsearch 叢集。

範例

下列 JSON 範例定義 AWS IoT 規則中的 OpenSearch 動作，以及如何指定`OpenSearch`動作的欄位。如需詳細資訊，請參閱 OpenSearchAction。

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "_doc",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 OpenSearch 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

> **Note**
>
> 取代`type`的欄位適用於 OpenSearch 1.0 版。對於 1.0 之後的任何版本， 的值`type`必須是`_doc`。

另請參閱

什麼是 ？Amazon OpenSearch Service 開發人員指南 中的Amazon OpenSearch Service

重新發佈

重新發佈 (`republish`) 動作將 MQTT 訊息重新發佈至另一個 MQTT 主題。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`iot:Publish`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`headers`

MQTT 5.0 版標頭資訊。

如需詳細資訊，請參閱《AWS API 參考》 中的 RepublishAction 和 MqttHeaders。

`topic`

要將訊息重新發佈至該 MQTT 主題。

如要重新發佈至以 $ 開頭的預留主題，請改用 $$。例如，如要重新發佈至裝置影子主題 `$aws/things/MyThing/shadow/update`，請將主題指定為 `$$aws/things/MyThing/shadow/update`。

> **Note**
>
> 重新發佈至預留任務主題不受支援。
> AWS IoT Device Defender 預留主題不支援 HTTP 發佈。

支援替代範本：是

`qos`

(選用) 重新發佈訊息時要使用的服務品質 (QoS) 層級。有效值：`0`、`1`。預設值為 `0`。如需 MQTT QoS 的詳細資訊，請參閱 MQTT。

支援替代範本：否

`roleArn`

允許 AWS IoT 發佈至 MQTT 主題的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的重新發佈動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "another/topic",
                    "qos": 1,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本定義重新發佈動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

下列 JSON 範例定義在 AWS IoT 規則headers中使用 重新發佈動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
```

```
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
                    "headers": {
                        "payloadFormatIndicator": "UTF8_DATA",
                        "contentType": "rule/contentType",
                        "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
                        "userProperties": [
                            {
                                "key": "ruleKey1",
                                "value": "ruleValue1"
                            },
                            {
                                "key": "ruleKey2",
                                "value": "ruleValue2"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

> **Note**
>
> 原始來源 IP 不會透過重新發布動作傳遞。

S3

S3 (s3) 動作會從 MQTT 訊息將資料寫入 Amazon Simple Storage Service (Amazon S3) 儲存貯體。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`s3:PutObject`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。

- 如果您使用 AWS KMS 受客戶管理的 AWS KMS key 來加密 Amazon S3 中的靜態資料，則服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱《Amazon Simple Storage Service 開發人員指南》中的AWS 受管 AWS KMS keys 和客戶受 AWS KMS keys管。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`bucket`

要寫入資料的 Amazon S3 儲存貯體。

僅支援替代範本：API 和 AWS CLI

`cannedacl`

(選用) Amazon S3 固定的 ACL，負責控制物件金鑰辨識出之物件的存取權限。如需詳細資訊 (包含允許的值)，請參閱固定 ACL。

支援替代範本：否

`key`

寫入資料的檔案路徑。

考慮一個範例，其中此參數為 `${topic()}/${timestamp()}`，且規則會收到主題為 `some/topic` 訊息。若目前的時間戳記為 `1460685389`，則此動作會將資料寫入在 S3 儲存貯體 `some/topic` 資料夾中名為 `1460685389` 的檔案。

> **Note**
>
> 如果您使用靜態金鑰， 會在每次呼叫規則時 AWS IoT 覆寫單一檔案。我們建議您使用訊息時間戳記或另一個唯一的訊息識別符，以便每一個收到的訊息都會將新檔案儲存於 Amazon S3 中。

支援替代範本：是

`roleArn`

允許存取 Amazon S3 儲存貯體的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 S3 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "bucketName": "amzn-s3-demo-bucket",
                    "cannedacl": "public-read",
                    "key": "${topic()}/${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon Simple Storage Service 使用者指南》中的什麼是 Amazon S3？

SNS

使用 SNS (sns) 動作，將來自 MQTT 訊息的資料作為 Amazon Simple Notification Service (Amazon SNS) 推送通知進行傳送。

您可依循對您展示如何使用 SNS 動作 來建立及測試規則的教學課程。如需詳細資訊，請參閱教學課程：傳送 Amazon SNS 通知。

> **Note**
>
> SNS 動作不支援 Amazon SNS FIFO (先進先出) 主題。因為規則引擎是一項完全分配式的服務，因此在呼叫 SNS 動作時無法保證訊息順序。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`sns:Publish`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用 AWS KMS 客戶受管 AWS KMS key 來加密 Amazon SNS 中的靜態資料，服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的金鑰管理。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`targetArn`

推送通知傳送到的 SNS 主題或個別裝置。

僅支援替代範本：API 和 AWS CLI

`messageFormat`

(選用) 訊息格式。Amazon SNS 會使用此設定來判斷是否應剖析承載，以及是否應擷取承載與特定平台相關的部分。有效值：`JSON`、`RAW`。預設為 `RAW`。

支援替代範本：否

`roleArn`

允許存取 SNS 的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 SNS 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
```

```
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 SNS 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-1:123456789012:${topic()}",
                    "messageFormat": "JSON",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon Simple Notification Service 開發人員指南》中的什麼是 Amazon Simple Notification Service？
- 教學課程：傳送 Amazon SNS 通知

SQS

使用 SQS (sqs) 動作，將資料從 MQTT 訊息傳送至 Amazon Simple Queue Service (Amazon SQS)佇列。

> **Note**
>
> SQS 動作不支援 Amazon SQS FIFO (先進先出) 佇列。因為規則引擎是一項完全分配式的服務，因此在觸發 SQS 動作時無法保證訊息順序。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`sqs:SendMessage`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用受管 AWS KMS 客戶 AWS KMS key 來加密 Amazon SQS 中的靜態資料，則服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的金鑰管理。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`queueUrl`

要寫入資料的 Amazon SQS 佇列 URL。此 URL 中的區域不需要 AWS 區域 與您的AWS IoT 規則相同。

> **Note**
>
> AWS 區域 使用 SQS 規則動作進行跨資料傳輸可能會產生額外費用。如需詳細資訊，請參閱 Amazon SQS 定價。

僅支援替代範本：API 和 AWS CLI

`useBase64`

設定此參數為 `true` 以配置規則動作，在資料寫入 Amazon SQS 佇列之前，對訊息資料進行 base64 編碼。預設為 `false`。

支援替代範本：否

`roleArn`

允許存取 Amazon SQS 佇列的 IAM 角色。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 SQS 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
my_sqs_queue",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

下列 JSON 範例使用 AWS IoT 規則中的替代範本來定義 SQS 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
${topic()}",
                    "useBase64": true,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
```

```
                }
            }
        ]
    }
}
```

另請參閱

- 《Amazon Simple Queue Service 開發人員指南》中的什麼是 Amazon Simple Queue Service？

步驟函數

Step Functions (`stepFunctions`) 動作會啟動 AWS Step Functions 狀態機器。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行`states:StartExecution`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇或建立角色， AWS IoT 以允許 執行此規則動作。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`stateMachineName`

要啟動的 Step Functions 狀態機器名稱。

僅支援替代範本：API 和 AWS CLI

`executionNamePrefix`

(選用) 狀態機器執行的名稱，包含此字首後接 UUID。若未提供名稱，Step Functions 會為每個狀態機器建立一個唯一的名稱。

支援替代範本：是

`roleArn`

授予啟動狀態機器 AWS IoT 許可之角色的 ARN。如需詳細資訊，請參閱要求。

支援替代範本：否

範例

下列 JSON 範例定義 AWS IoT 規則中的 Step Functions 動作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "stepFunctions": {
                    "stateMachineName": "myStateMachine",
                    "executionNamePrefix": "myExecution",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_step_functions"
                }
            }
        ]
    }
}
```

另請參閱

- 什麼是 AWS Step Functions？《AWS Step Functions 開發人員指南》中的

Timestream

Timestream 規則動作將屬性 (測量) 從 MQTT 訊息寫入 Amazon Timestream 表格中。如需 Amazon Timestream 的詳細資訊，請參閱什麼是 Amazon Timestream？。

> **Note**
>
> 並非所有 AWS 區域都提供 Amazon Timestream。若 Amazon Timestream 無法在您的區域中使用，則不會顯示於規則動作清單中。

此規則儲存於 Timestream 資料庫中的屬性是規則查詢陳述式所產生的屬性。剖析查詢陳述式結果中每個屬性值，以推斷其資料類型 (如 the section called “DynamoDBv2” 動作中所示)。每個屬性值都會寫

入其在 Timestream 表格中的記錄。如要指定或變更屬性的資料類型，請使用查詢陳述式中的 `cast()` 函數。如需每個 Timestream 記錄內容的詳細資訊，請參閱 the section called “Timestream 記錄內容”。

> **Note**
>
> 在 SQL V2 (2016-03-23) 中，作為完整數的數值，例如 `10.0`，將轉換為其整數表示形式 (`10`)。將其明確轉換為 `Decimal` 值，例如使用 cast() 函數不能阻止此種行為，結果仍是一個 `Integer` 值。這可能會造成類型不相符的錯誤，導致資料無法記錄於 Timestream 資料庫中。若要將整數數值當成 `Decimal` 值處理，請針對規則查詢陳述式使用 SQL V1 (2015-10-08)。

> **Note**
>
> 可以寫入 Amazon Timestream 表格的最大 Timestream 規則動作值為 100。如需詳細資訊，請參閱 Amazon Timestream 配額參考。

要求

此規則動作具有下列需求：

- AWS IoT 可以擔任以執行 `timestream:DescribeEndpoints`和 `timestream:WriteRecords`操作的 IAM 角色。如需詳細資訊，請參閱授予 AWS IoT 規則所需的存取權。

  在 AWS IoT 主控台中，您可以選擇、更新或建立角色， AWS IoT 以允許 執行此規則動作。
- 如果您使用客戶 AWS KMS 加密 Timestream 中的靜態資料，服務必須具有代表發起人使用 AWS KMS key 的許可。如需詳細資訊，請參閱 AWS 服務如何使用 AWS KMS。

Parameters

當您使用此動作建立 AWS IoT 規則時，您必須指定下列資訊：

`databaseName`

Amazon Timestream 資料庫的名稱，其中包含用於接收此動作所建立之記錄的資料表。另請參閱**`tableName`**。

AWS CLI 僅支援替代範本：API 和

dimensions

寫入每個量測記錄的時間序列的中繼資料屬性。例如，EC2 執行個體的名稱和可用區域，或風力發電機製造商的名稱即為維度。

name

中繼資料維度名稱。這是資料庫資料表記錄中的欄的名稱。

無法命名的維度：measure_name、measure_value, 或 time。這些為預留名稱。維度名稱不可以 ts_ 或 measure_value 開頭，且其不可包含冒號 (:) 字元。

支援替代範本：否

value

要寫入資料庫記錄的該資料欄的值。

支援替代範本：是

roleArn

授予 AWS IoT 許可之角色的 Amazon 資源名稱 (ARN)，可寫入 Timestream 資料庫表格。如需詳細資訊，請參閱要求。

支援替代範本：否

tableName

要寫入測量記錄的資料庫表格名稱。另請參閱**databaseName**。

僅支援替代範本：API 和 AWS CLI

timestamp

要用於項目的 timestamp 的值。若為空白，則會使用處理項目的時間。

unit

value 中所述運算式產生的 timestamp 值的精確度。

有效值：SECONDS | MILLISECONDS | MICROSECONDS | NANOSECONDS。預設值為 MILLISECONDS。

value

傳回一個很長的 Epoch 時間值的表達式。

您可使用 the section called “time_to_epoch(String, String)” 函數，從訊息承載中傳遞的日期或時間建立有效的時間戳記。

Timestream 記錄內容

此動作寫入 Amazon Timestream 表格的資料包括時間戳記、來自 Timestream 規則動作的中繼資料，及規則查詢陳述式的結果。

對於查詢陳述式結果中的每個屬性 (測量)，此規則動作會將記錄寫入具有這些欄的指定 Timestream 表格。

欄名稱	屬性類型	Value	說明
dimension-name	DIMENSION	指定於 Timestream 規則動作項目中的值。	指定於規則動作項目中的每個維度都會於 Timestream 資料庫中建立一個具維度名稱的欄。
measure_name	MEASURE_NAME	屬性名稱	查詢陳述式結果中的屬性名稱，其值指定於 `measure_value::` *data-type* 欄中。
measure_value::*data-type*	MEASURE_VALUE	查詢陳述式結果中的屬性值。屬性名稱位於 `measure_name` 欄中。	此值會進行解譯 * 並轉換為最適合下列的比對項目：`bigint`、`boolean`、`double` 或 `varchar`。Amazon Timestream 為每種資料類型建立一個個別的欄。訊息中的值可使用規則查詢陳述式中的 `cast()` 函數轉換為另一個資料。

欄名稱	屬性類型	Value	說明
time	TIMESTAMP	資料庫中記錄的日期和時間。	此值是由規則引擎或 timestamp 屬性來指派 (若已定義)。

* 從訊息承載讀取的屬性值解譯如下。請參閱 the section called “範例”，以取得每個案例的說明。

- 將 true 或 false 的未加引號值解釋為 boolean 類型。
- 將十進位數字解釋為 double 類型。
- 將未帶小數點的十進位數值解釋為 bigint 類型。
- 將帶引號的字串解釋為 varchar 類型。
- 將物件和陣列數值轉換為 JSON 字串並儲存為 varchar 類型。

範例

下列 JSON 範例使用 規則中的替代範本來定義 Timestream AWS IoT 規則動作。

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "timestream": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_timestream",
          "tableName": "devices_metrics",
          "dimensions": [
            {
              "name": "device_id",
              "value": "${clientId()}"
            },
            {
              "name": "device_firmware_sku",
              "value": "My Static Metadata"
            }
          ],
          "databaseName": "record_devices"
```

```
        }
      }
    ]
  }
}
```

上一個範例中定義的 Timestream 主題規則動作與下列訊息承載搭配使用會導致 Amazon Timestream 記錄寫入後續表格中。

```
{
  "boolean_value": true,
  "integer_value": 123456789012,
  "double_value": 123.456789012,
  "string_value": "String value",
  "boolean_value_as_string": "true",
  "integer_value_as_string": "123456789012",
  "double_value_as_string": "123.456789012",
  "array_of_integers": [23,36,56,72],
  "array of strings": ["red", "green","blue"],
  "complex_value": {
    "simple_element": 42,
    "array_of_integers": [23,36,56,72],
    "array of strings": ["red", "green","blue"]
  }
}
```

下表會顯示使用指定主題規則動作處理先前訊息承載建立的資料庫欄和記錄。`device_firmware_sku` 和 `device_id` 欄為定義於主題規則動作中的 DIMENSIONS (維度)。Timestream 主題規則動作會建立 `time` 欄及 `measure_name` 和 `measure_value::*` 欄，其中其會填入主題規則動作查詢陳述式結果的值。

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	complex_value	-	{"simple_element":42,"array_of_integers":[23,	-	-	2020-08-26 22:42:16.423000000

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
				36,56,72],"字串陣列":["紅","綠","藍"]}			
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	integer_value_as_string	-	123456789012	-	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	boolean_value	-	-	-	TRUE	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	integer_value	123456789012	-	-	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	string_value	-	字串值	-	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	array_of_integers	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	字串陣列	-	["紅","綠","藍"]	-	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	boolean_value_as_string	-	TRUE	-	-	2020-08-26 22:42:16.423000000

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	double_value	-	-	123.456789012	-	2020-08-26 22:42:16.423000000
我的靜態中繼資料	iotconsole-159EXAMPLE738-0	double_value_as_string	-	123.45679	-	-	2020-08-26 22:42:16.423000000

規則疑難排解

如果您遇到規則方面的問題，建議您啟動 CloudWatch Logs。您可分析您的記錄，即可判斷該問題是否在於授權，或者是否為 WHERE 子句條件不相符之類的問題。如需詳細資訊，請參閱設定 CloudWatch Logs。

使用 AWS IoT 規則存取跨帳戶資源

您可以設定跨帳戶存取的 AWS IoT 規則，以便將某個帳戶的 MQTT 主題擷取的資料路由到另一個帳戶的 AWS 服務，例如 Amazon SQS 和 Lambda。以下說明如何設定跨帳戶資料擷取的 AWS IoT 規則，從一個帳戶中的 MQTT 主題，到另一個帳戶中的目的地。

跨帳戶規則可使用資源型許可，在目的地資源上進行配至。因此，只有支援資源型許可的目的地才能使用 AWS IoT 規則啟用跨帳戶存取。支援的目的地包括 Amazon SQS、Amazon SNS、Amazon S3 和 AWS Lambda。

> **Note**
>
> 對於支援的目的地，除了 Amazon SQS 之外，您必須在與其他服務資源 AWS 區域 相同的中定義規則，以便規則動作可以與該資源互動。如需 AWS IoT 規則動作的詳細資訊，請參閱AWS IoT 規則動作。如需規則 SQS 動作的詳細資訊，請參閱 ???。

先決條件

- 熟悉 AWS IoT 規則
- 了解 IAM 使用者、角色 和資源型許可
- 安裝了 AWS CLI

Amazon SQS 的跨帳戶設定

案例：帳戶 A 將資料從 MQTT 訊息傳送至帳戶 B 的 Amazon SQS 佇列。

AWS 帳戶	帳戶稱為	說明
1111-1111-1111	帳戶 A	規則動作：`sqs:SendMessage`
2222-2222-2222	帳戶 B	Amazon SQS 佇列 • ARN：*arn:aws:sqs:region:2222-2222-2222:ExampleQueue* • URL：*https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue*

> **Note**
>
> 您的目的地 Amazon SQS 佇列不需要與您的AWS IoT 規則 AWS 區域 位於相同的 中。如需規則 SQS 動作的詳細資訊，請參閱 ???。

執行帳戶 A 任務

> **注意**
>
> 如要執行下列命令，您的 IAM 使用者應具有以規則的 Amazon 資源名稱 (ARN) 作為資源的 `iot:CreateTopicRule` 許可，及以資源作為角色之 ARN 的 `iam:PassRole` 動作許可。

1. 使用帳戶 A 的 IAM 使用者配置 AWS CLI。
2. 建立信任 AWS IoT 規則引擎的 IAM 角色，並連接允許存取帳戶 B Amazon SQS 佇列的政策。請參閱授予 AWS IoT 必要存取權中的命令和政策文件範例。
3. 如要建立連接至主題的規則，請執行 create-topic-rule 命令。

```
aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
```

下列為範例承載檔案，具有一個會將傳送至 `iot/test` 主題之所有訊息插入指定 Amazon SQS 佇列的規則。SQL 陳述式會篩選訊息，而角色 ARN 授予 AWS IoT 新增 Amazon SQS 佇列之訊息的許可。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sqs": {
    "queueUrl": "https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role",
    "useBase64": false
   }
  }
 ]
}
```

如需如何在 AWS IoT 規則中定義 Amazon SQS 動作的詳細資訊，請參閱AWS IoT 規則動作 - Amazon SQS。

執行帳戶 B 任務

1. 使用帳戶 B 的 IAM 使用者配置 AWS CLI。
2. 如要將 Amazon SQS 佇列資源的許可授予帳戶 A，請執行 add-permission 命令。

```
aws sqs add-permission --queue-url https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue --label SendMessagesToMyQueue --aws-account-ids 1111-1111-1111 --actions SendMessage
```

Amazon SNS 的跨帳戶設定

案例：帳戶 A 將資料從 MQTT 訊息傳送至帳戶 B 的 Amazon SNS 主題。

AWS 帳戶	帳戶稱為	說明
1111-1111-1111	帳戶 A	規則動作：`sns:Publish`
2222-2222-2222	帳戶 B	Amazon SNS 主題 ARN：*arn:aws:sns:region:2222-2222-2222:ExampleTopic*

執行帳戶 A 任務

> **備註**
>
> 如要執行下列命令，您的 IAM 使用者應具有以規則 ARN 作為資源的 `iot:CreateTopicRule` 許可，以及資源作為角色 ARN 的 `iam:PassRole` 動作許可。

1. 使用帳戶 A 的 IAM 使用者配置 AWS CLI。
2. 建立信任 AWS IoT 規則引擎的 IAM 角色，並連接允許存取帳戶 B Amazon SNS 主題的政策。如需命令和政策文件的範例，請參閱授予 AWS IoT 必要的存取權。
3. 如要建立連接至主題的規則，請執行 create-topic-rule 命令。

   ```
   aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
   ```

   下列為範例承載檔案，具有一個會將傳送至 `iot/test` 主題之所有訊息插入指定 Amazon SNS 主題的規則。SQL 陳述式會篩選訊息，而角色 ARN 授予 AWS IoT 將訊息傳送至 Amazon SNS 主題的許可。

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
     {
   ```

```
    "sns": {
     "targetArn": "arn:aws:sns:region:2222-2222-2222:ExampleTopic",
     "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
    }
   }
  ]
}
```

如需如何在 AWS IoT 規則中定義 Amazon SNS 動作的詳細資訊，請參閱AWS IoT 規則動作 - Amazon SNS。

執行帳戶 B 任務

1. 使用帳戶 B 的 IAM 使用者配置 AWS CLI。
2. 如要將 Amazon SNS 主題資源的許可授予帳戶 A，請執行 add-permission 命令。

```
aws sns add-permission --topic-arn arn:aws:sns:region:2222-2222-2222:ExampleTopic
 --label Publish-Permission --aws-account-id 1111-1111-1111 --action-name Publish
```

Amazon S3 的跨帳戶設定

案例：帳戶 A 將資料從 MQTT 訊息傳送至帳戶 B 的 Amazon S3 儲存貯體。

AWS 帳戶	帳戶稱為	說明
1111-1111-1111	帳戶 A	規則動作：`s3:PutObject`
2222-2222-2222	帳戶 B	Amazon S3 儲存貯體 ARN：*arn:aws:s3:::amzn-s3-demo-bucket*

執行帳戶 A 任務

> **注意**
>
> 如要執行下列命令，您的 IAM 使用者應具有以規則 ARN 作為資源的 `iot:CreateTopicRule` 許可，以及資源作為角色 ARN 的 `iam:PassRole` 動作許可。

1. 使用帳戶 A 的 IAM 使用者配置 AWS CLI。
2. 建立信任 AWS IoT 規則引擎並連接允許存取帳戶 B Amazon S3 儲存貯體的政策的 IAM 角色。如需命令和政策文件的範例，請參閱授予 AWS IoT 必要的存取權。
3. 如要建立連接至您目標 S3 儲存貯體的規則，請執行 create-topic-rule 命令。

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
```

下列為範例承載檔案，具有一個會將傳送至 `iot/test` 主題之所有訊息插入指定 Amazon S3 儲存貯體的規則。SQL 陳述式會篩選訊息，而角色 ARN 授予 AWS IoT 將訊息新增至 Amazon S3 儲存貯體的許可。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "s3": {
    "bucketName": "amzn-s3-demo-bucket",
    "key": "${topic()}/${timestamp()}",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
   }
  }
 ]
}
```

如需如何在 AWS IoT 規則中定義 Amazon S3 動作的詳細資訊，請參閱AWS IoT 規則動作 - Amazon S3。

執行帳戶 B 任務

1. 使用帳戶 B 的 IAM 使用者配置 AWS CLI。
2. 建立一個信任帳戶 A 之委託人的儲存貯體政策。

下列為範例承載檔案，其會定義信任另一個帳戶之委託人的儲存貯體政策。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AddCannedAcl",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:root"
                ]
        },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

如需詳細資訊，請參閱儲存貯體政策範例。

3. 如要將儲存貯體政策連接至指定的儲存貯體，請執行 put-bucket-policy 命令。

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://./amzn-s3-demo-bucket-policy.json
```

4. 如要讓跨帳戶存取運作，請確定您具有正確的封鎖所有公有存取設定。如需詳細資訊，請參閱 Amazon S3 安全最佳實務。

的跨帳戶設定 AWS Lambda

案例：帳戶 A 叫用帳戶 B 的 AWS Lambda 函數，傳入 MQTT 訊息。

AWS 帳戶	帳戶稱為	說明
1111-1111-1111	帳戶 A	規則動作：`lambda:InvokeFunction`
2222-2222-2222	帳戶 B	Lambda 函數 ARN： *arn:aws:lambda:region:2222-2222-2222:function:example-function*

執行帳戶 A 任務

> **備註**
>
> 如要執行下列命令，您的 IAM 使用者應具有以規則 ARN 作為資源的 `iot:CreateTopicRule` 許可，以及資源作為角色 ARN 的 `iam:PassRole` 動作許可。

1. 使用帳戶 A 的 IAM 使用者配置 AWS CLI。
2. 執行 create-topic-rule 命令，建立定義對帳戶 B 之 Lambda 函數的跨帳戶存取權規則。

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
```

下列為範例承載檔案，具有一個會將傳送至 `iot/test` 主題之所有訊息插入指定 Lambda 函數的規則。SQL 陳述式會篩選訊息，而角色 ARN 授予 AWS IoT 將資料傳入 Lambda 函數的許可。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:region:2222-2222-2222:function:example-function"
   }
  }
 ]
}
```

如需如何在 規則中 AWS IoT 定義 AWS Lambda 動作的詳細資訊，請參閱AWS IoT 規則動作 - Lambda。

執行帳戶 B 任務

1. 使用帳戶 B 的 IAM 使用者配置 AWS CLI。
2. 執行 Lambda 的 add-permission 命令，給予 AWS IoT 規則啟用 Lambda 函數的許可。如要執行下列命令，您的 IAM 使用者應具有 `lambda:AddPermission` 動作的權限。

```
aws lambda add-permission --function-name example-function --region us-east-1 --
principal iot.amazonaws.com --source-arn arn:aws:iot:region:1111-1111-1111:rule/
example-rule --source-account 1111-1111-1111 --statement-id "unique_id" --action
 "lambda:InvokeFunction"
```

選項：

--principal

此欄位提供 AWS IoT （ 由 表示`iot.amazonaws.com`) 呼叫 Lambda 函數的許可。

--source-arn

此欄位確認僅 AWS IoT 中的 `arn:aws:iot:region:1111-1111-1111:rule/example-rule` 會觸發此 Lambda 函數，且相同或不同帳戶中的任何其他規則皆無法啟動此 Lambda 函數。

--source-account

此欄位會確認 僅代表`1111-1111-1111`帳戶 AWS IoT 啟用此 Lambda 函數。

> **備註**
>
> 若您看到來自 AWS Lambda 函數主控台 Configuration (組態) 之下的錯誤訊息「找不到規則」，請略過錯誤訊息並繼續測試連線。

錯誤處理 (錯誤動作)

當 從裝置 AWS IoT 收到訊息時，規則引擎會檢查訊息是否符合規則。若是如此，則會對規則的查詢陳述式進行評估，並啟動規則的動作，傳送查詢陳述式的結果。

如果在啟動動作時發生問題，則若針對規則指定了錯誤動作，規則引擎便會啟動該動作。這可能在以下情況時發生：

- 規則並未擁有存取 Amazon S3 儲存貯體的許可。
- 使用者錯誤會造成 DynamoDB 佈建的傳輸量超量。

> **Note**
>
> 本主題所述的錯誤處理適用於規則動作。若要偵錯 SQL 問題，包括外部函數，您可以設定 AWS IoT 記錄。如需詳細資訊，請參閱???。

錯誤動作訊息格式

每一項規則和訊息會產生單條訊息。例如，如果相同規則中的兩個規則動作失敗，則錯誤動作會收到包含那兩個錯誤的訊息。

錯誤動作訊息看似下列範例。

```
{
  "ruleName": "TestAction",
  "topic": "testme/action",
  "cloudwatchTraceId": "7e146a2c-95b5-6caf-98b9-50e3969734c7",
  "clientId": "iotconsole-1511213971966-0",
  "base64OriginalPayload":
 "ewogICJtZXNzYWdlIjogIkhlbGxvIHZyb20gQVdTIElvVCBjb25zb2xlIgp9",
  "failures": [
    {
      "failedAction": "S3Action",
      "failedResource": "us-east-1-s3-verify-user",
      "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist (Service: Amazon S3; Status Code: 404; Error
 Code: NoSuchBucket; Request ID: 9DF5416B9B47B9AF; S3 Extended Request ID:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y=).
 Message arrived on: error/action, Action: s3, Bucket: us-
east-1-s3-verify-user, Key: \"aaa\". Value of x-amz-id-2:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y="
    }
  ]
}
```

ruleName

觸發錯誤動作的規則名稱。

主題

收到原始訊息的主題。

cloudwatchTraceId

指向 CloudWatch 中錯誤記錄的專屬識別資料。

clientId

訊息發佈者的用戶端 ID。

base64OriginalPayload

原始訊息承載 Base64 編碼。

failures

failedAction

無法完成的動作名稱 (例如，「S3Action」)。

failedResource

資源名稱 (例如，S3 儲存貯體名稱)。

errorMessage

對該項錯誤的敘述和說明。

錯誤動作範例

以下範例為有加入錯誤動作的規則。下列規則中有個動作會將訊息資料寫入 DynamoDB 表格，還有一個錯誤動作，會將資料寫入 Amazon S3 儲存貯體：

```
{
    "sql" : "SELECT * FROM ..."
    "actions" : [{
        "dynamoDB" : {
            "table" : "PoorlyConfiguredTable",
            "hashKeyField" : "AConstantString",
            "hashKeyValue" : "AHashKey"}}
    ],
    "errorAction" : {
        "s3" : {
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
            "bucketName" : "message-processing-errors",
            "key" : "${replace(topic(), '/', '-') + '-' + timestamp() + '-' +
 newuuid()}"
```

```
        }
    }
}
```

您可以在錯誤動作的 SQL 陳述式中使用任何函數或替代範本，包括外部函數：aws_lambda()、get_dynamodb()、get_registry_data()get_thing_shadow()、、get_secret()和 decode()。如果錯誤動作需要呼叫外部 函數，則叫用錯誤動作可能會導致外部函數產生額外的帳單。

如需規則以及如何指定錯誤動作的詳細資訊，請參閱建立 AWS IoT 規則。

若需更多有關使用 CloudWatch 監控規則成功或失敗的資訊，請參閱 AWS IoT 指標和維度。

使用基本擷取減少簡訊費用

您可以使用基本擷取，安全地將裝置資料傳送至 AWS 服務 支援的 AWS IoT 規則動作，而不會產生簡訊費用。基本擷取會從擷取路徑移除發佈/訂閱訊息代理程式，讓資料流程最佳化。

基本擷取可從您的裝置或應用程式傳送訊息。訊息的前三個層級具有以 `$aws/rules/rule_name` 開頭的主題名稱，其中 *rule_name* 是您要叫用的 AWS IoT 規則的名稱。

您可將基本擷取字首 (`$aws/rules/rule_name`) 新增至您用來叫用規則的訊息主題，即可搭配基本擷取使用現有的規則。例如，如果您有一個名為 `BuildingManager` 規則會透過 `Buildings/Building5/Floor2/Room201/Lights` (`"sql": "SELECT * FROM 'Buildings/#'"`) 之類的訊息主題叫用，則您可以透過傳送主題為 `$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights` 的訊息，使用基本擷取叫用相同的規則。

> **Note**
>
> - 您的裝置和規則無法訂閱基本擷取預留主題。例如，指標 AWS IoT Device Defender num-messages-received不會發出，因為它不支援訂閱主題。如需詳細資訊，請參閱預留主題。
> - 如果您需要發佈/訂閱代理程式將訊息分發給多個訂閱者 （例如，將訊息傳遞到其他裝置和規則引擎），您應該繼續使用 AWS IoT 訊息代理程式來處理訊息分發。不過，請確認使用基本擷取主題以外的主題來發佈您的訊息。

使用基本擷取

在使用基本擷取之前，請驗證您的裝置或應用程式是否正在使用對 `$aws/rules/*` 具有發佈許可的政策。或者，您可以在 政策`$aws/rules/rule_name/*`中使用 指定個別規則的許可。除此之外，您的裝置和應用程式可以繼續使用他們與 AWS IoT Core的現有連線。

當訊息抵達規則引擎時，從基本擷取叫用的規則和透過訊息代理程式訂閱叫用的規則，在實作或錯誤處理上就沒有差別。

您可以建立用於基本擷取的規則。請謹記以下幾點：

- 基本擷取主題的初始字首 (`$aws/rules/rule_name`) 無法用於 topic(Decimal) 函數。
- 如果您定義了只能由基本擷取叫用的規則，`rule` 定義中 `sql` 欄位的 `FROM` 子句則為選用。如果規則也需要由必須透過訊息代理程式傳送的其他訊息呼叫 (例如，因為那些其他訊息必須分配到多個訂閱者)，則仍然會需要此項。如需詳細資訊，請參閱AWS IoT SQL 參考。
- 首三個層級的基本擷取主題 (`$aws/rules/rule_name`) 不會算在主題的 8 區段長度或 256 個總字元限制中。除此之外，適用其他相同限制，如 AWS IoT 限制所述。
- 如果接收的訊息包含指定非作用中或不存在規則的基本擷取主題，則會在 Amazon CloudWatch 記錄中建立錯誤記錄，以協助您偵錯。如需詳細資訊，請參閱Rules engine 日誌項目。系統會指出 `RuleNotFound` 指標，讓您可以為此指標建立警示。如需詳細資訊，請參閱 規則指標 中的規則指標。
- 您仍可使用 QoS 1 發佈基本擷取主題。訊息成功交付至規則引擎後，您就會收到 PUBACK。收到 PUBACK 不表示您的規則已經成功完成動作。您可以設定錯誤動作，以在動作執行時處理錯誤。如需詳細資訊，請參閱錯誤處理 (錯誤動作)。

AWS IoT SQL 參考

在 中 AWS IoT，使用類似 SQL 的語法定義規則。SQL 陳述式是由三種類型的子句所組成：

SET

（選用）定義您可以在 SQL 陳述式和替代範本中重複使用的變數。使用表達式將值指派給變數。在 SELECT 和 WHERE 子句以及動作替代範本中參考這些變數。

SET 子句支援 資料類型、運算子、函數、文字、案例陳述式、JSON Extensions、變數和 巢狀物件查詢。

SELECT

(必要) 從傳入的訊息承載擷取資訊並對資訊執行轉換。要使用的訊息是由 FROM 子句中指定的主題篩選條件所識別。

SELECT 子句支援 資料類型、運算子、函數、文字、案例陳述式JSON Extensions、替代範本、、變數、 巢狀物件查詢和 二進位承載。

FROM

MQTT 訊息主題篩選條件，可識別要從中擷取資料的訊息。系統會為每則傳送到符合此處指定的主題篩選條件之 MQTT 主題的訊息啟動該規則。對於透過訊息啟動的規則是必要的，而這些訊息是透過訊息代理程式傳遞的。若為僅使用基本擷取功能啟動的規則，則為選用。

WHERE

(選用) 新增條件邏輯，其會判斷是否執行某規則指定的動作。

WHERE 子句支援 資料類型、運算子、函數、文字、案例陳述式、JSON Extensions、變數和 巢狀物件查詢。

SQL 陳述式範例如下所示：

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

MQTT 訊息 (也稱作傳入承載) 的範例如下所示：

```
{
    "color":"red",
    "temperature":100
}
```

如果此訊息是發佈在 'topic/subtopic' 主題上，便會觸發規則，SQL 陳述式惠受到評估。如果 color 屬性大於 50，SQL 陳述式會擷取 "temperature" 屬性的值。WHERE 子句會指定條件 temperature > 50。AS 關鍵字將 "color" 屬性重新命名為 "rgb"。結果 (也稱作傳出承載) 會如以下所示：

```
{
    "rgb":"red"
}
```

該資料接下來會轉送給該規則的動作，並發送資料以進行更多處理作業。如需規則動作的詳細資訊，請參閱 AWS IoT 規則動作。

> **Note**
>
> AWS IoT SQL 語法目前不支援註解。
> 包含空格的屬性名稱不能用作 SQL 陳述式中的欄位名稱。雖然傳入的承載可以包含含有空格的屬性名稱，但這些名稱不能在 SQL 陳述式中使用。但是，如果您使用萬用字元 (*) 欄位名稱規範，這些名稱會被傳遞到傳出承載。

SELECT 子句

AWS IoT SELECT 子句基本上與 ANSI SQL SELECT 子句相同，但有一些細微差異。

SELECT 子句支援 資料類型、運算子、函數、文字、案例陳述式JSON Extensions、、變數、 巢狀物件查詢和 二進位承載。

以使用 SELECT 子句來擷取傳入的 MQTT 訊息內的資訊。您也可以使用 SELECT * 來擷取整個傳入訊息的承載。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

如果承載為 JSON 物件，您可以參考物件中的鍵。您的傳出承載會包含鍵值組。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

您可以使用 AS 關鍵字重新命名索引鍵。例如：

```
Incoming payload published on topic 'topic/subtopic':{"color":"red", "temperature":50}
SQL:SELECT color AS my_color FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red"}
```

您可以選擇多個項目，以逗號分隔即可。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
```

```
SQL: SELECT color as my_color, temperature as fahrenheit FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red","fahrenheit":50}
```

您可以選擇多個項目，包括以「*」新增項目至來年的承載。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT *, 15 as speed FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50, "speed":15}
```

您可以使用 "VALUE" 關鍵字產生傳出的承載不 JSON 物件。使用 SQL 版本 2015-10-08，您只能選取一個項目。使用 SQL 版本 2016-03-23 或更新版本，您也可以選取要作為頂層物件輸出的陣列。

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

您可以使用 '.' 語法來深入巢狀 JSON 物件傳入的承載。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":
{"red":255,"green":0,"blue":0}, "temperature":50}
SQL: SELECT color.red as red_value FROM 'topic/subtopic'
Outgoing payload: {"red_value":255}
```

如需如何使用 JSON 物件和包含保留字元 (例如數字或連字號 (減號) 字元) 之屬性名稱的相關資訊，請參閱 JSON Extensions

您可以使用函數 (請參閱 函數)，將傳入的承載。您可以使用括號進行分組。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM
 'topic/subtopic'
Outgoing payload: {"celsius":10,"my_color":"RED"}
```

您可以使用 SELECT 子句中的物件常值，以包含標準識別符不支援之字元的索引鍵來建構 JSON 物件，例如連字號。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT {"key-with-hyphen": color, "123-numeric-start": temperature} FROM 'topic/
subtopic'
```

```
Outgoing payload: {"key-with-hyphen":"red","123-numeric-start":50}
```

FROM 子句

FROM 子句可讓主題或主題篩選條件訂閱您的規則。以單引號 (') 括住主題或主題篩選條件。系統會為每則傳送到符合此處指定的主題篩選條件之 MQTT 主題的訊息觸發該規則。您可以使用主題篩選條件訂閱類似主題的群組。

範例：

發佈在主題 'topic/subtopic' 的傳入承載：{temperature: 50}

發佈在主題 'topic/subtopic-2' 的傳入承載：{temperature: 50}

SQL："SELECT temperature AS t FROM 'topic/subtopic'"。

規則已訂閱至 'topic/subtopic'，因此傳入的承載會傳遞至該規則。傳送至規則動作的外寄承載為：{t: 50}。規則並未給 'topic/subtopic-2' 訂閱，因此發佈在 'topic/subtopic-2' 的訊息不會觸發該規則。

萬用字元範例：

您可以使用 '#'(多層級) 萬用字元，以比對一或多個特定路徑元素：

發佈在主題 'topic/subtopic' 的傳入承載：{temperature: 50}。

發佈在主題 'topic/subtopic-2' 的傳入承載：{temperature: 60}。

發佈在主題 'topic/subtopic-3/details' 的傳入承載：{temperature: 70}。

發佈在主題 'topic-2/subtopic-x' 的傳入承載：{temperature: 80}。

SQL："SELECT temperature AS t FROM 'topic/#'"。

已將該規則訂閱至開頭為 'topic' 的任何主題，因此該規則會執行三次，將傳出的 {t: 50} (表示主題/子主題)、{t: 60} (表示主題/子主題 2) 和 {t: 70} (表示主題/子主題 3/詳細資訊) 的承載傳送給其動作。它沒有訂閱 'topic-2/subtopic-x'，因此不會觸發 {temperature: 80} 訊息的規則。

+ 萬用字元範例：

您可以使用 '+'(單層級) 萬用字元，以比對任何一個特定路徑元素：

發佈在主題 'topic/subtopic' 的傳入承載：{temperature: 50}。

發佈在主題 'topic/subtopic-2' 的傳入承載：{temperature: 60}。

發佈在主題 'topic/subtopic-3/details' 的傳入承載：{temperature: 70}。

發佈在主題 'topic-2/subtopic-x' 的傳入承載：{temperature: 80}。

SQL："SELECT temperature AS t FROM 'topic/+'"。

所有主題訂閱的規則有兩個路徑元素，第一個為：'topic'。規則是針對傳送至 'topic/subtopic' 和 'topic/subtopic-2' 的訊息執行，但不是針對 'topic/subtopic-3/details' (它具有比主題過濾條件更多的層級) 或 'topic-2/subtopic-x' (它不是以 topic 開頭) 執行。

SET 子句

使用 SET 子句來定義存放運算式結果的變數。您可以在 SELECT 和 WHERE 子句以及替代範本中重複使用這些變數。這可協助您避免重複複雜的表達式，並減少 SQL 陳述式中的函數呼叫次數。

SET 子句支援 資料類型、運算子、函數、文字案例陳述式、JSON Extensions、、變數和 巢狀物件查詢。

SET 子句語法

SET 子句必須出現在 SQL 陳述式中的 SELECT 子句前面。使用下列語法：

```
SET @variable_name = expression [, @variable_name2 = expression2]
```

語法規則：

- 使用 啟動變數名稱 @
- 變數名稱可以包含字母、數字和底線
- 變數名稱長度最多可達 64 個字元
- 您可以在單一 SET 子句中設定多個變數，並以逗號分隔
- 每個變數只能指派一次（變數不可變）
- 每個 SQL 陳述式只能使用一次 SET 關鍵字

使用變數

定義變數之後，您可以在 中使用它們：

- SELECT 子句

- WHERE 子句
- 其他 SET 變數指派
- 動作替換範本
- 錯誤動作替換範本
- 巢狀 SELECT 查詢
- 函數參數（某些參數，例如 roleArn 參數和切換類似 函數模式的參數`transform("enrichArray", attributes, values)`，不支援變數）

變數會使用與 SET `@variable_name` 子句相同的語法來參考。您也可以使用 JSON 延伸語法來存取包含 物件之變數的屬性，例如 `@variable_name.property`。

SET 子句範例

基本變數用量

下列範例顯示發佈在主題 上的承載`device/data`：`{"temp_fahrenheit": 75, "humidity": 60}`

SQL 陳述式：

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

傳出承載：`{"celsius": 23.89, "humidity": 60}`

存取內嵌 JSON 物件中的成員

下列範例顯示發佈在主題 上的承載`device/data`：`{"device1": {"deviceId":"weather_sensor", "deviceData": {"sensors": {"temp_fahrenheit": 75, "humidity": 60}, "location": [47.606,-122.332]}}}`

SQL 陳述式：

```
SET @device_sensor_data = device1.deviceData.sensors
SELECT @device_sensor_data.temp_fahrenheit AS temp_fahrenheit,
 @device_sensor_data.humidity as humidity, device1.deviceId as deviceId FROM 'device/
data'
```

傳出承載：`{"temp_fahrenheit":75,"humidity":60,"deviceId":"weather_sensor"}`

如需如何使用 JSON 延伸模組的詳細資訊，請參閱 JSON Extensions

避免重複的函數呼叫

SET 變數有助於避免重複複雜的解碼操作：

```
SET @decoded_data = decode(encode(*, 'base64'), 'proto', 'schema', 'schema.desc',
 'message.proto', 'Message')
SELECT @decoded_data.sensor_id, @decoded_data.reading FROM 'device/protobuf'
WHERE @decoded_data.reading > 100
```

如果沒有 SET 變數，您需要重複三次解碼函數，這超過函數呼叫限制。

多個變數

您可以在單一 SET 子句中定義多個變數，方法是使用逗號分隔它們：

```
SET @user_data = get_user_properties(device_id), @threshold = 50
SELECT @user_data.name, temp_fahrenheit FROM 'sensors/+'
WHERE temp_fahrenheit > @threshold AND @user_data.active = true
```

在替代範本中使用變數

變數也可以用於動作替換範本，讓您可以在 SQL 陳述式和規則動作之間重複使用運算值。

SQL 陳述式：

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

動作組態：

```
{
  "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/testRuleRole",
    "bucketName": "bucket",
    "key": "temperature-data/${device_id}/temp-${@temp_celsius}C.json"
  }
}
```

在此範例中，SET 變數`@temp_celsius`用於替代範本，以建構 S3 動作的金鑰欄位。

非 JSON 承載用量

SET 變數不支援直接支援非 JSON 承載，因此必須先對承載進行編碼或解碼：

```
SET @encoded_payload = encode(*, 'base64')
SELECT @encoded_payload AS raw_data FROM 'device/binary'
```

如需如何使用非 JSON 承載的詳細資訊，請參閱 使用二進位承載

SET 子句限制

下列限制適用於 SET 變數：

- 每個 SQL 陳述式最多 10 個唯一變數
- 最大變數值大小為 128 KiB （最小 UTF-8 JSON 字串）
- 所有變數的總值大小上限為 128 KiB
- 變數名稱限制為 64 個字元
- 變數可以直接接受 JSON 承載 （必須先編碼/解碼非 JSON 承載）

WHERE 子句

WHERE 子句會判斷是否執行某規則指定的動作。如果 WHERE 子句判斷值為 true，則規則動作已執行。否則，規則動作就不會執行。

WHERE 子句支援 資料類型、運算子、函數、文字、案例陳述式、JSON Extensions、變數和 巢狀物件查詢。

範例：

發佈在 topic/subtopic 的傳入承載：{"color":"red", "temperature":40}。

SQL：SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'。

在此情況下，該規則會觸發，但不會執行由規則指定的動作。將不會出現傳出承載。

您可以在 WHERE 子句中使用函數和運算子。不過您無法參考任何在 SELECT 中以 AS 關鍵字建立的別名。(系統會優先評估 WHERE 子句，以判定 SELECT 是否受到評估。)

非 JSON 承載的範例：

在「主題/副主題」上發佈的傳入非 JSON 承載：`80`

SQL:`SELECT decode(encode(*, 'base64'), 'base64') AS value FROM 'topic/subtopic' WHERE decode(encode(*, 'base64'), 'base64') > 50

在此情況下，該規則會觸發，並且執行由規則指定的動作。傳出承載將由 SELECT 子句轉換為 JSON 承載 {"value":80}。

資料類型

AWS IoT 規則引擎支援所有 JSON 資料類型。

支援的資料類型

Type	意義
Int	離散的 Int。最多 34 位數。
Decimal	精度為 34 位的 Decimal，最小非零量級為 1E-999，最大量級為 9.999...E999。 **Note** 有些函數傳回的 Decimal 值為雙精度，而非 34 位數的精度。 使用 SQL V2 (2016-03-23) 時，數值若為整數 (例如 10.0) 會被當成 Int 值 (10) 處理，而不是預期的 Decimal 值 (10.0)。若要可靠地將整數數值當成 Decimal 值處理，請針對規則查詢陳述式使用 SQL V1 (2015-10-08)。
Boolean	True 或 False
String	UTF-8 字串。
Array	一系列類型無需相同的值。
Object	由鍵和值組成的 JSON 值。鍵必須為字串。值可以為任何類型。

Type	意義
`Null`	`Null` 是由 JSON 定義。此為用來代表缺少某個值的實際值。您可以在 SQL 陳述式中使用 `Null` 的關鍵字來明確建立 `Null` 的值。例如：`"SELECT NULL AS n FROM 'topic/subtopic'"`

`Undefined`	並非值。無法以 JSON 明確顯示，除非是省略該值。舉例而言，在物件 `{"foo": null}` 中，「foo」鍵會傳回 NULL，但「bar」鍵傳回 `Undefined` 。SQL 語言在內部將 `Undefined` 視為一個值，但無法以 JSON 顯示，因此序列化為 JSON 時，結果會是 `Undefined` 。
	`{"foo":null, "bar":undefined}`
	序列化為 JSON 如下：
	`{"foo":null}`
	同樣地，`Undefined` 由自身序列化時，會轉換為空白字串。以無效引數 (例如，錯誤類型、錯誤的引數數量等等) 呼叫的函數會傳回 `Undefined` 。

轉換

下表列出的是當某個類型的值轉換為另一種類型 (給函數指定錯誤類型的值) 所得的結果。舉例而言，如果絕對值的函數「abs」(預期為 `Int`或 `Decimal`) 得到的是 `String`，其會嘗試按照下列規則將 `String` 轉換為 `Decimal`。此例中，'abs("-5.123")' 會視為 'abs(-5.123)'。

> **Note**
>
> 不會企圖轉換為 `Array`、`Object`、`Null`、`Undefined`。

To Decimal

引數類型	結果
Int	無小數點的 Decimal。
Decimal	來源值。
Boolean	Undefined 。(您可以明確使用 Cast 函數轉換 true = 1.0、false = 0.0。)
String	SQL 引擎會嘗試將字串剖析為 Decimal. AWS IoT attempts，以剖析符合規則表達式的字串：^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。「0」、「-1.2」、「5E-12」均為自動轉換為 Decimal 的範例字串。
Array	Undefined .
物件	Undefined .
Null	Null.
未定義	Undefined .

To Int

引數類型	結果
Int	來源值。
Decimal	四捨五入到最接近 Int 的來源值。
Boolean	Undefined 。(您可以明確使用 Cast 函數轉換 true = 1.0、false = 0.0。)
String	SQL 引擎會嘗試將字串剖析為 Decimal. AWS IoT attempts，以剖析符合規則表達式的字串：^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。"0"、"-1.2"、"5E-12" 是字串的所有範例，

引數類型	結果
	這些字串會自動轉換為 Decimals. AWS IoT attempts，以String將 轉換為 Decimal，然後截斷該字串的小數位Decimal數以製作 Int。
Array	Undefined .
物件	Undefined .
Null	Null.
未定義	Undefined .

To Boolean

引數類型	結果
Int	Undefined 。(您可以明確使用 cast 函數轉換 0 = False、any_nonzero_value = True。)
Decimal	Undefined 。(您可以明確使用 Cast 函數轉換 0 = False、any_nonzero_value = True。)
Boolean	原始的值。
String	「true」= True 而「false」= False (不區分大小寫)。其他字串值為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

To String

引數類型	結果
`Int`	以標準表示法表示的 `Int` 的字串顯示方式。
`Decimal`	代表 `Decimal` 值的字串，可能是採取科學表示法。
`Boolean`	「true」或「false」。所有小寫。
`String`	原始的值。
陣列	序列化為 JSON 的 `Array`。該結果字串為以逗號分隔的清單，並包含在方括弧內。`String` 在括號中。`Decimal`、`Int`、`Boolean` 和 `Null` 則不是。
物件	序列化為 JSON 的物件。該結果字串為以逗號分隔的鍵值組清單，並以大括號為開頭和結尾。`String` 在括號中。`Decimal`、`Int`、`Boolean` 和 `Null` 則不是。
Null	`Undefined` .
未定義	未定義。

運算子

下列運算子可用於 SELECT 和 WHERE 子句。

AND 運算子

傳回 `Boolean` 結果。執行邏輯 AND 運算。如果左右運算元為 true，即傳回 true。否則會傳回 false。需要 `Boolean` 運算元或不區分大小寫的「true」或「false」字串運算元。

語法： *`expression`* `AND` *`expression`*。

AND 運算子

左運算元	右運算元	Output
Boolean	Boolean	Boolean。如果兩個運算元皆為 true 即為 true。否則為 false。
String/Boolean	String/Boolean	如果所有字串均為「true」或「false」(不區分大小寫)，他們會轉換為 Boolean，並以 *boolean* AND *boolean* 的方式正常處理。
其他值	其他值	Undefined .

OR 運算子

傳回 Boolean 結果。執行邏輯 OR 運算。如果右運算元或左運算元有一個為 true 即傳回 true。否則會傳回 false。需要 Boolean 運算元或不區分大小寫的「true」或「false」字串運算元。

語法： *expression* OR *expression*。

OR 運算子

左運算元	右運算元	Output
Boolean	Boolean	Boolean。如果有一個運算元為 true 即為 true。否則為 false。
String/Boolean	String/Boolean	如果所有字串均為「true」或「false」(不區分大小寫)，它們會轉換為布林值，並以 *boolean* OR *boolean* 的方式正常處理。
其他值	其他值	Undefined .

NOT 運算子

傳回 Boolean 結果。執行邏輯 NOT 運算。如果運算元為 false 即傳回 true。否則即傳回 true。需要 Boolean 運算元或不區分大寫的「true」或「false」字串運算元。

語法：NOT *expression*。

NOT 運算子

運算元	Output
`Boolean`	`Boolean`。如果運算元為 false 即為 true。否則為 true。
`String`	如果字串為「true」或「false」(不區分大小寫)，則會轉換為對應的布林值，並傳回相反的值。
其他值	`Undefined` .

IN 運算子

傳回 Boolean 結果。您可以使用 WHERE 子句中的 IN 運算子來檢查值是否符合陣列中的任何值。如果找到相符項目，則傳回 true，否則傳回 false。

語法： *`expression`* `IN` *`expression`*。

IN 運算子

左運算元	右運算元	Output
`Int/Decimal/String/A`	`Array`	如果在陣列中找到 `Integer/Decimal/String/Array/Object`元素，則為 True。否則為 false。

範例：

```
SQL: "select * from 'a/b' where 3 in arr"

JSON: {"arr":[1, 2, 3, "three", 5.7, null]}
```

在此範例中，條件子句`where 3 in arr`將評估為 true，因為 3 存在於名為 的陣列中`arr`。因此，在 SQL 陳述式中， `select * from 'a/b'`將執行。此範例也會顯示陣列可以是異質的。

EXISTS 運算子

傳回 Boolean 結果。您可以在條件式子句中使用 EXISTS 運算子來測試子查詢中是否存在元素。如果子查詢傳回一或多個元素，則傳回 true；如果子查詢未傳回元素，則傳回 false。

語法： *expression*。

範例：

```
SQL: "select * from 'a/b' where exists (select * from arr as a where a = 3)"

JSON: {"arr":[1, 2, 3]}
```

在此範例中，條件子句`where exists (select * from arr as a where a = 3)`將評估為 true，因為 3 存在於名為 的陣列中`arr`。因此，在 SQL 陳述式中， `select * from 'a/b'`將執行。

範例：

```
SQL: select * from 'a/b' where exists (select * from e as e where foo = 2)

JSON: {"foo":4,"bar":5,"e":[{"foo":1},{"foo":2}]}
```

在此範例中，條件子句`where exists (select * from e as e where foo = 2)`將評估為 true，因為 JSON 物件`e`內的陣列包含物件 `{"foo":2}`。因此，在 SQL 陳述式中， `select * from 'a/b'`將執行。

> 運算子

傳回 Boolean 結果。如果左運算元大於右運算元即傳回 true。兩個運算元均轉換為 Decimal，再做比較。

語法：*expression* > *expression*。

> 運算子

左運算元	右運算元	Output
Int/Decimal	Int/Decimal	Boolean。如果左運算元大於右運算元即為 true。否則為 false。
String/Int/Deci	String/Int/Deci	如果所有字串均可轉換為 Decimal，即 Boolean。如果左運算元大於右運算元即傳回 true。否則為 false。
其他值	Undefined .	Undefined .

>= 運算子

傳回 Boolean 結果。如果左運算元大於或等於右運算元，即傳回 true。兩個運算元均轉換為 Decimal，再做比較。

語法：*expression* >= *expression*。

>= 運算子

左運算元	右運算元	Output
Int/Decimal	Int/Decimal	Boolean。如果左運算元大於或等於右運算元，即為 true。否則為 false。
String/Int/Deci	String/Int/Deci	如果所有字串均可轉換為 Decimal，即 Boolean。如果左運算元大於或等於右運算元，即傳回 true。否則為 false。
其他值	Undefined .	Undefined .

< 運算子

傳回 Boolean 結果。如果左側運算元少於右運算元。兩個運算元均轉換為 Decimal，再做比較。

語法：*expression* < *expression*。

< 運算子

左運算元	右運算元	Output
Int/Decimal	Int/Decimal	Boolean。如果左運算元小於右運算元即為 true。否則為 false。
String/Int/Deci	String/Int/Deci	如果所有字串均可轉換為 Decimal，即 Boolean。如果左側運算元少於右運算元。否則為 false。
其他值	Undefined	Undefined

<= 運算子

傳回 Boolean 結果。如果左運算元小於或等於右運算元，即傳回 true。兩個運算元均轉換為 Decimal，再做比較。

語法：*expression* <= *expression*。

<= 運算子

左運算元	右運算元	Output
Int/Decimal	Int/Decimal	Boolean。如果左運算元小於或等於右運算元，即為 true。否則為 false。
String/Int/Deci	String/Int/Deci	如果所有字串均可轉換為 Decimal，即 Boolean。如果左運算元小於或等於右運算元，即傳回 true。否則為 false。
其他值	Undefined	Undefined

<> 運算子

傳回 Boolean 結果。如果左右運算元不相等，即傳回 true。否則即傳回 false。

語法： *expression* <> *expression*。

<> 運算子

左運算元	右運算元	Output
Int	Int	如果左運算元不等於右運算元即為 true。否則為 false。
Decimal	Decimal	如果左運算元不等於右運算元即為 true。否則為 false。在做比較前，先將 Int 轉換為 Decimal。
String	String	如果左運算元不等於右運算元即為 true。否則為 false。
陣列	陣列	如果各運算元的項目不相等且順序不同，即為 true。否則為 false

左運算元	右運算元	Output
物件	物件	如果各運算元的鍵和值不相同，即為 true。否則為 false。鍵/值的順序不重要。
Null	Null	False。
任何值	`Undefined`	未定義。
`Undefined`	任何值	未定義。
類型不符合	類型不符合	True。

= 運算子

傳回 `Boolean` 結果。如果兩個左右運算元相同，即傳回 true。否則即傳回 false。

語法： *`expression`* `=` *`expression`*。

= 運算子

左運算元	右運算元	Output
`Int`	`Int`	如果左運算元等於右運算元即為 true。否則為 false。
`Decimal`	`Decimal`	如果左運算元等於右運算元即為 true。否則為 false。在做比較前，先將 `Int` 轉換為 `Decimal`。
`String`	`String`	如果左運算元等於右運算元即為 true。否則為 false。
陣列	陣列	如果各運算元的項目相等且順序相同，即為 true。否則為 false。
物件	物件	如果各運算元的鍵和值相同，即為 true。否則為 false。鍵/值的順序不重要。
任何值	`Undefined`	`Undefined` .
`Undefined`	任何值	`Undefined` .
類型不符合	類型不符合	False。

+ 運算子

「+」是過載的運算子。可以用來連接或新增字串。

語法： *expression* + *expression*。

+ 運算子

左運算元	右運算元	Output
String	任何值	將右運算元轉換為字串，並連接至左運算元的尾端。
任何值	String	將左運算元轉換為字串，並將右運算元連接至轉換後的左運算元的尾端。
Int	Int	Int 值。將運算元相加。
Int/Decimal	Int/Decimal	Decimal 值。將運算元相加。
其他值	其他值	Undefined .

- 運算子

從左運算元中減去右運算元。

語法： *expression* - *expression*。

- 運算子

左運算元	右運算元	Output
Int	Int	Int 值。從左運算元中減去右運算元。
Int/Decimal	Int/Decimal	Decimal 值。從左運算元中減去右運算元。
String/Int/Deci	String/Int/Deci	如果所有字串都正確轉換為小數，則會傳回 Decimal 值。從左運算元中減去右運算元。如果不是，則傳回 Undefined 。
其他值	其他值	Undefined .
其他值	其他值	Undefined .

* 運算子

將左運算元乘以右運算元。

語法： *expression* * *expression*。

* 運算子

左運算元	右運算元	Output
Int	Int	Int 值。將左運算元乘以右運算元。
Int/Decimal	Int/Decimal	Decimal 值。將左運算元乘以右運算元。
String/Int/Deci	String/Int/Deci	如果所有字串都正確轉換為小數，則會傳回 Decimal 值。將左運算元乘以右運算元。如果不是，則傳回 Undefined 。
其他值	其他值	Undefined .

/ 運算子

將左運算元除以右運算元。

語法： *expression* / *expression*。

/ 運算子

左運算元	右運算元	Output
Int	Int	Int 值。將左運算元除以右運算元。
Int/Decimal	Int/Decimal	Decimal 值。將左運算元除以右運算元。
String/Int/Deci	String/Int/Deci	如果所有字串都正確轉換為小數，則會傳回 Decimal 值。將左運算元除以右運算元。如果不是，則傳回 Undefined 。
其他值	其他值	Undefined .

% 運算子

傳回左運算元除以右運算元的餘數。

語法： *expression* % *expression*。

% 運算子

左運算元	右運算元	Output
Int	Int	Int 值。傳回左運算元除以右運算元的餘數。
String/Int/Deci	String/Int/Deci	如果所有字串都正確轉換為小數，則會傳回 Decimal 值。傳回左運算元除以右運算元的餘數。否則為 Undefined 。
其他值	其他值	Undefined .

函數

您可以在 SQL 表達式的 SELECT 或 WHERE 子句中使用下列內建函數。

下列外部函數的計費方式等同於規則動作的計費方式：aws_lambda、get_registry_data()、get_dynamodb()和 get_thing_shadow()。只有在您將 Protobuf 訊息解碼至 JSON 時，您才會收到decode()函數的費用。如需詳細資訊，請參閱 AWS IoT Core 定價頁面。

abs(Decimal)

傳回某個數字的絕對值。受 SQL 版本 2015-10-08 和更新版本支援。

範例：abs(-5) 傳回 5。

引數類型	結果
Int	Int，引數的絕對值。
Decimal	Decimal，引數的絕對值。
Boolean	Undefined .

引數類型	結果
String	Decimal。結果為引數的絕對值。如果字串無法轉換，則結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

accountid()

將擁有此規則的帳戶 ID 傳回為 String。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

accountid() = "123456789012"

acos(Decimal)

以弧度傳回數字的反餘弦值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：acos(0) = 1.5707963267948966

引數類型	結果
Int	Decimal (使用雙精度)，引數的反向餘弦值。傳回的虛數結果為 Undefined 。
Decimal	Decimal (使用雙精度)，引數的反向餘弦值。傳回的虛數結果為 Undefined 。
Boolean	Undefined .

引數類型	結果
String	Decimal，引數的反向餘弦值。如果字串無法轉換，則結果為 Undefined 。傳回的虛數結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

asin(Decimal)

以弧度傳回數字的反正弦值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：asin(0) = 0.0

引數類型	結果
Int	Decimal (使用雙精度)，引數的反向正弦值。傳回的虛數結果為 Undefined 。
Decimal	Decimal (使用雙精度)，引數的反向正弦值。傳回的虛數結果為 Undefined 。
Boolean	Undefined .
String	Decimal (使用雙精度)，引數的反向正弦值。如果字串無法轉換，則結果為 Undefined 。傳回的虛數結果為 Undefined 。
Array	Undefined .
物件	Undefined .

引數類型	結果
Null	Undefined 。
未定義	Undefined 。

atan(Decimal)

以弧度傳回數字的反正切值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`atan(0)` = 0.0

引數類型	結果
Int	Decimal (使用雙精度)，引數的反向正切值。傳回的虛數結果為 Undefined 。
Decimal	Decimal (使用雙精度)，引數的反向正切值。傳回的虛數結果為 Undefined 。
Boolean	Undefined 。
String	Decimal，引數的反向正切值。如果字串無法轉換，則結果為 Undefined 。傳回的虛數結果為 Undefined 。
Array	Undefined 。
物件	Undefined 。
Null	Undefined 。
未定義	Undefined 。

atan2(Decimal, Decimal)

以弧度傳回 x 軸正軸和兩個引數所定義的 (x, y) 點之間的角度。 逆時針角度為正值 (上半象限，y > 0)，順時鐘的角度為負值 (下半象限 y < 0)。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`atan2(1, 0)` = 1.5707963267948966

引數類型	引數類型	結果
Int/Decimal	Int/Decimal	Decimal (使用雙精度)，x 軸與 點之間的角度。
Int/Decimal/String	Int/Decimal/String	Decimal，所述之點的反向正切 串無法轉換，則結果為 Undefi
其他值	其他值	Undefined .

aws_lambda (functionArn、inputJson)

呼叫指定的 Lambda 函數，其會將 inputJson 傳送至 Lambda 函數，並傳回 Lambda 函數產生的 JSON。

引數

引數	說明
functionArn	Lambda 函數呼叫的 ARN。Lambda 函數必須傳回 JSON 資料。
inputJson	傳遞到 Lambda 函數的 JSON 輸入。若要傳遞巢狀物件查詢和文字，您必須使用 SQL 版本 2016-03-23。

您必須授予 AWS IoT lambda:InvokeFunction叫用指定 Lambda 函數的許可。下列範例顯示了如何使用 AWS CLI授與 lambda:InvokeFunction 的許可：

```
aws lambda add-permission --function-name "function_name"
--region "region"
--principal iot.amazonaws.com
--source-arn arn:aws:iot:us-east-1:account_id:rule/rule_name
```

```
--source-account "account_id"
--statement-id "unique_id"
--action "lambda:InvokeFunction"
```

以下為 add-permission 命令的引數：

--function-name

Lambda 函數的名稱。您可以新增許可來更新函數的資源政策。

--region

您帳戶的 AWS 區域 。

--principal

取得許可的委託人。這應該`iot.amazonaws.com`允許呼叫 Lambda 函數的 AWS IoT 許可。

--source-arn

該項規則的 ARN。您可以使用 get-topic-rule AWS CLI 命令來取得規則的 ARN。

--source-account

定義規則 AWS 帳戶 的 。

--statement-id

專屬的陳述式識別符。

--action

您想要在此陳述式中允許的 Lambda 動作。若要允許 AWS IoT 叫用 Lambda 函數，請指定 `lambda:InvokeFunction`。

> **Important**
>
> 如果您在未提供 `source-arn`或 的情況下新增 AWS IoT 委託人的許可`source-account`，任何 AWS 帳戶 使用 Lambda 動作建立規則的 都可以觸發規則來叫用您的 Lambda 函數 AWS IoT。如需詳細資訊，請參閱 Lambda 許可模型。

指定的 JSON 訊息承載，如以下所示：

```
{
```

```
    "attribute1": 21,
    "attribute2": "value"
}
```

aws_lambda 函數可用於呼叫 Lambda 函數，如下所示：

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

如果您想要傳遞完整的 MQTT 訊息承載，您可以使用「*」來指定 JSON 承載，如下列範例所示。

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

payload.inner.element 會從發佈在「主題/子主題」主題上的訊息選取資料。

some.value 會從 Lambda 函數產生的輸出結果中選取資料。

> **Note**
>
> 該規則引擎會限制 Lambda 函數的執行期間。規則的 Lambda 函數呼叫應該會在 2000 毫秒內完成。

bitand (Int、Int)

在兩個 Int (已轉換) 引數的位元表現上執行按位元的 AND 運算。受 SQL 版本 2015-10-08 和更新版本支援。

範例：bitand(13, 5) = 5

引數類型	引數類型	結果
Int	Int	Int，兩個引數按位元的 AND 運
Int/Decimal	Int/Decimal	Int，兩個引數按位元的 AND 運 Int 的數字會無條件捨去至最接

引數類型	引數類型	結果
		如果任何引數無法轉換至 Int，Undefined 。
Int/Decimal/String	Int/Decimal/String	Int，兩個引數按位元的 AND 運串都會轉換為小數，並會無條件近的 Int (整數)。如果轉換失敗 Undefined 。
其他值	其他值	Undefined .

bitor(Int, Int)

在兩個引數的位元表現上執行按位元的 OR 運算。受 SQL 版本 2015-10-08 和更新版本支援。

範例：bitor(8, 5) = 13

引數類型	引數類型	結果
Int	Int	Int，兩個引數按位元的 OR 運
Int/Decimal	Int/Decimal	Int，兩個引數按位元的 OR 運 Int 的數字會無條件捨去至最接 如果轉換失敗，則結果為 Undef
Int/Decimal/String	Int/Decimal/String	Int，兩個引數按位元的 OR 運 串都會轉換為小數，並會無條件 近的 Int (整數)。如果轉換失敗 Undefined 。
其他值	其他值	Undefined .

bitxor(Int, Int)

在兩個 Int (已轉換) 引數的位元表現上執行按位元的 XOR 運算。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`bitor(13, 5) = 8`

引數類型	引數類型	結果
Int	Int	Int，兩個引數按位元的 XOR 運
Int/Decimal	Int/Decimal	Int，兩個引數按位元的 XOR 運 Int 的數字會無條件捨去至最接
Int/Decimal/String	Int/Decimal/String	Int 是在兩個引數上的位元 XOR 換為小數，並無條件捨去至最接 如果任何轉換失敗，則結果為 Un 。
其他值	其他值	Undefined .

bitnot(Int)

在 Int (已轉換) 引數的位元表現上執行按位元的 NOT 運算。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`bitnot(13) = 2`

引數類型	結果
Int	Int，引數按位元的 NOT 運算。
Decimal	Int，引數按位元的 NOT 運 算。Decimal值會無條件捨去至最接近 Int 的值。
String	Int，引數按位元的 NOT 運算。字串會 轉換為小數，並會無條件捨去至最接近的 Int (整數)。如果任何轉換失敗，則結果為 Undefined 。
其他值	其他值。

cast()

將一個值從某個資料類型轉換至另一個類型。除了增加了將數字和布林值相互轉換的能力外，轉換行為大致如同標準轉換。如果 AWS IoT 無法判斷如何將一種類型轉換為另一種類型，則結果為 `Undefined`。受 SQL 版本 2015-10-08 和更新版本支援。格式：cast (*`value`* as *`type`*)。

範例：

```
cast(true as Int) = 1
```

在呼叫 `cast` 時，以下關鍵字可能出現在「as」之後：

對於 SQL 版本 2015-10-08 與 2016-03-23

關鍵字	結果
`String`	將值轉換為 `String`。
Nvarchar	將值轉換為 `String`。
文字	將值轉換為 `String`。
Ntext	將值轉換為 `String`。
varchar	將值轉換為 `String`。
`Int`	將值轉換為 `Int`。
Integer	將值轉換為 `Int`。
Double	將值轉換為 `Decimal` (使用雙精度)。

此外，對於 SQL 版本 2016-03-23

關鍵字	結果
`Decimal`	將值轉換為 `Decimal`。
Bool	將值轉換為 `Boolean`。
`Boolean`	將值轉換為 `Boolean`。

轉換規則：

轉換為小數

引數類型	結果
`Int`	無小數點的 `Decimal`。
`Decimal`	來源值。 **Note** 使用 SQL V2 (2016-03-23) 時，數值若為整數 (例如 `10.0`) 會傳回 `Int` 值 (`10`)，而不是預期的 `Decimal` 值 (`10.0`)。若要可靠地將整數數值當成 `Decimal` 值處理，請針對規則查詢陳述式使用 SQL V1 (2015-10-08)。
`Boolean`	true = 1.0、false = 0.0。
`String`	嘗試將字串剖析為 `Decimal`。 AWS IoT 會嘗試剖析字串以符合正規表示式：^-?\d+(\.\d+)?((?i)E-?\d+)?$。「0」、「-1.2」、「5E-12」均為自動轉換為 Decimal 的範例字串。
Array	`Undefined` .
物件	`Undefined` .
Null	`Undefined` .
未定義	`Undefined` .

轉換為 Int

引數類型	結果
`Int`	來源值。
`Decimal`	無條件捨去至最接近 `Int` 的來源值。
`Boolean`	true = 1.0、false = 0.0。
`String`	嘗試將字串剖析為 `Decimal`。 AWS IoT 會嘗試剖析字串以符合正規表示式：^-?\d+(\.\d+)?((?i)E-?\d+)?$。「0」、「-1.2」、「5E-12」均為自動轉換為 Decimal 的範例字串。 AWS IoT 會嘗試將字串轉換為 `Decimal`，並無條件捨去至最接近的 `Int`。
Array	`Undefined` .
物件	`Undefined` .
Null	`Undefined` .
未定義	`Undefined` .

轉換到 **`Boolean`**

引數類型	結果
`Int`	0 = False，any_nonzero_value = True。
`Decimal`	0 = False，any_nonzero_value = True。
`Boolean`	來源值。
`String`	「true」= True 而「false」= False (不區分大小寫)。其他字串值 = `Undefined` 。
Array	`Undefined` .

引數類型	結果
物件	Undefined .
Null	Undefined .
未定義	Undefined .

轉換為字串

引數類型	結果
Int	以標準表示法表示的 Int 的字串顯示方式。
Decimal	代表 Decimal 值的字串，可能是採取科學表示法。
Boolean	「true」或「false」，均為小寫。
String	來源值。
陣列	序列化為 JSON 的陣列。結果字串是以方括號括住，並以逗號分隔的清單。String 在括號中。Decimal、Int 和 Boolean 則不是。
物件	序列化為 JSON 的物件。JSON 字串是首尾以大括號括住，並以逗號分隔的鍵值組清單。String 在括號中。Decimal、Int、Boolean 和 Null 則不是。
Null	Undefined .
未定義	Undefined .

ceil(Decimal)

將指定的 Decimal 無條件進位至最近的 Int。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

ceil(1.2) = 2

ceil(-1.2) = -1

引數類型	結果
Int	Int，引數值。
Decimal	Int，無條件進位至最接近的 Decimal 的 Int 值。
String	Int。此字串會轉換成 Decimal 並四捨五入至最接近 Int。如果字串無法轉換為 Decimal，則結果為 Undefined 。
其他值	Undefined .

chr(String)

傳回指定的 Int 引數對應到的 ASCII 字元。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

chr(65) = "A"。

chr(49) = "1"。

引數類型	結果
Int	對應到指定的 ASCII 值的字元。如果引數並非有效的 ASCII 值，結果會為 Undefined 。
Decimal	對應到指定的 ASCII 值的字元。Decimal 的引數值會無條件捨去至最接近 Int 的值。如果引數並非有效的 ASCII 值，結果會為 Undefined 。

引數類型	結果
Boolean	Undefined .
String	如果 String 可以轉換為 Decimal，要無條件捨去到最接近的 Int。如果引數並非有效的 ASCII 值，結果會為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
其他值	Undefined .

clientid()

傳回傳送訊息的 MQTT 用戶端的 ID，如果訊息不是透過 MQTT 傳送，則為 n/a。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`clientid()` = "123456789012"

concat()

連接陣列或字串。此函數會接受任意數量的引數，並傳回 String 或 Array。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`concat()` = Undefined.

`concat(1)` = "1"。

`concat([1, 2, 3], 4)` = [1, 2, 3, 4]。

`concat([1, 2, 3], "hello")` = [1, 2, 3, "hello"]

`concat("con", "cat")` = "concat"

`concat(1, "hello")` = "1hello"

`concat("he","is","man")` = "heisman"

`concat([1, 2, 3], "hello", [4, 5, 6])` = [1, 2, 3, "hello", 4, 5, 6]

引數數量	結果
0	`Undefined` .
1	傳回的引數未經修改。
2+	如果任一引數為 `Array`，則結果會是包含所有引數的單一陣列。若沒有列出引數，且至少有一個引數是 `String`，則結果為所有引數 `String` 表示法的串聯。使用之前列出的標準轉換將引數轉換為字串。

cos(Decimal)

以弧度傳回數字的餘弦值。`Decimal` 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`cos(0)` = 1。

引數類型	結果
`Int`	`Decimal` (使用雙精度)，引數的餘弦值。傳回的虛數結果為 `Undefined` 。
`Decimal`	`Decimal` (使用雙精度)，引數的餘弦值。傳回的虛數結果為 `Undefined` 。
`Boolean`	`Undefined` .
`String`	`Decimal` (使用雙精度)，引數的餘弦值。如果字串無法轉換為 `Decimal`，則結

引數類型	結果
	果為 Undefined 。傳回的虛數結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

cosh(Decimal)

以弧度傳回數字的雙曲餘弦值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：cosh(2.3) = 5.037220649268761。

引數類型	結果
Int	Decimal (使用雙精度)，引數的雙曲餘弦值。傳回的虛數結果為 Undefined 。
Decimal	Decimal (使用雙精度)，引數的雙曲餘弦值。傳回的虛數結果為 Undefined 。
Boolean	Undefined .
String	Decimal (使用雙精度)，引數的雙曲餘弦值。如果字串無法轉換為 Decimal，則結果為 Undefined 。傳回的虛數結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .

引數類型	結果
未定義	Undefined .

decode(value, decodingScheme)

使用 decode 函數來解碼已編碼值。如果解碼字串為 JSON 文件，則會傳回可定址物件。否則，解碼字串會當成字串傳回。如果字串無法解碼，函數會傳回 NULL。此功能支援解碼 base64 編碼字串及協定緩衝區 (protobuf) 訊息格式。

受 SQL 版本 2016-03-23 和更新版本支援。

value

字串值或任何有效的表達式 (如 AWS IoT SQL 參考 所定義)，其會傳回字串。

decodingScheme

代表用來解碼值之結構描述的文字字串。目前僅支援 'base64' 和 'proto'。

對 base64 編碼字串進行解碼

在此範例中，訊息承載包含編碼值。

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
}
```

SQL 陳述式中的 decode 函數會解碼訊息承載中的值。

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

解碼 encoded_temp 值會產生下列有效的 JSON 文件，允許 SELECT 陳述式讀取溫度值。

```
{ "temperature": 33 }
```

這裡顯示此範例中 SELECT 陳述式的結果。

```
{ "temp": 33 }
```

如果解碼值不是有效的 JSON 文件，解碼值將以字串形式傳回。

對 protobuf 訊息承載進行解碼

您可以使用解碼 SQL 函數來設定可對 protobuf 訊息承載進行解碼的規則。如需詳細資訊，請參閱對 protobuf 訊息承載進行解碼。

> **Important**
>
> 如果您在設定 AWS IoT 委託人的許可`source-account`時省略 `source-arn`或 ，任何 AWS 帳戶 都可以透過其他 AWS IoT 規則叫用您的解碼函數。若要保護您的函數，請參閱《Amazon Simple Storage Service 使用者指南》中的儲存貯體政策。

功能簽章外觀如下：

```
decode(<ENCODED DATA>, 'proto', '<S3 BUCKET NAME>', '<S3 OBJECT KEY>', '<PROTO NAME>',
 '<MESSAGE TYPE>')
```

`ENCODED DATA`

指定要解碼的 protobuf 編碼資料。如果傳送到規則的整個訊息是 protobuf 編碼資料，則可以使用 * 參考原始二進位傳入承載。否則，此欄位必須是 base-64 編碼的 JSON 字串，而且可以直接傳入字串的參考。

1) 要解碼原始二進位 protobuf 傳入承載：

```
decode(*, 'proto', ...)
```

2) 要解碼由 base64 編碼字串 'a.b'表示的 protobuf 編碼訊息：

```
decode(a.b, 'proto', ...)
```

`proto`

指定要以 protobuf 訊息格式解碼的資料。如果您指定 `base64` 而不是 `proto`，此函數會將 base64 編碼字串解碼為 JSON。

`S3 BUCKET NAME`

您用來上傳 `FileDescriptorSet` 檔案的 Amazon S3 儲存貯體名稱。

S3 OBJECT KEY

用來在 Amazon S3 儲存貯體中指定 FileDescriptorSet 檔案的物件索引鍵。

PROTO NAME

從中產生 FileDescriptorSet 檔案的 .proto 檔案名稱 (不含副檔名)。

MESSAGE TYPE

待解碼資料在 FileDescriptorSet 檔案中所應符合的 Protobuf 訊息結構名稱。

使用解碼 SQL 函數的 SQL 運算式可能顯示如下：

```
SELECT VALUE decode(*, 'proto', 's3-bucket', 'messageformat.desc', 'myproto',
 'messagetype') FROM 'some/topic'
```

- *

  表示二進位傳入承載，符合名為 mymessagetype 的 Protobuf 訊息類型。

- messageformat.desc

  存放在名為 s3-bucket 的 Amazon S3 儲存貯體中的 FileDescriptorSet 檔案。

- myproto

  此原始 .proto 檔案的用途是產生名為 myproto.proto 的 FileDescriptorSet 檔案。

- messagetype

  如 myproto.proto 中所定義名為 messagetype 的訊息類型(以及任何匯入的相依性)。

encode(value, encodingScheme)

根據編碼機制，使用 encode 函數將承載 (可能並非 JSON 資料) 編碼為字串表現形式。受 SQL 版本 2016-03-23 和更新版本支援。

value

任何有效的表達式，如 AWS IoT SQL 參考 的定義。無論承載是否為 JSON 格式，您都可以指定 * 以編碼整個承載。若您提供了運算式，則在編碼之前，評估結果將轉換為字串。

encodingScheme

代表您想要使用的編碼機制的文字字串。目前僅支援 'base64'。

endswith(String, String)

傳回 Boolean，指出第一個 String 引數是否以第二個 String 引數結尾。如果引數為 Null 或 Undefined，則結果為 Undefined。受 SQL 版本 2015-10-08 和更新版本支援。

範例：endswith("cat","at") = true。

引數類型 1	引數類型 2	結果
String	String	如果第一個引數以第二個引數結 true。否則為 false。
其他值	其他值	所有引數都將使用標準轉換規則 串。如果第一個引數以第二個引 為 true。否則為 false。如果引數 Undefined ，則結果為 Undef

exp(Decimal)

傳回 e 次方的 Decimal 引數。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：exp(1) = e。

引數類型	結果
Int	Decimal (使用雙精度)，e ^ 引數。
Decimal	Decimal (使用雙精度)，e ^ 引數。
String	Decimal (使用雙精度)，e ^ 引數。如果 String 無法轉換為 Decimal，則結果為 Undefined 。

引數類型	結果
其他值	Undefined .

floor(Decimal)

將指定的 Decimal 無條件進位至最接近的 Int。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

floor(1.2) = 1

floor(-1.2) = -2

引數類型	結果
Int	Int，引數值。
Decimal	Int，Decimal 值會無條件捨去至最接近的 Int。
String	Int。此字串會轉換成 Decimal，並無條件捨去至最接近的 Int。如果字串無法轉換為 Decimal，則結果為 Undefined 。
其他值	Undefined .

get

從集合類型 (陣列、字串、物件) 擷取值。第一個引數不會套用任何轉換。轉換會按表格中的記錄套用至第二個引數。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

get(["a", "b", "c"], 1) = "b"

get({"a":"b"}, "a") = "b"

get("abc", 0) = "a"

引數類型 1	引數類型 2	結果
陣列	任何類型 (轉換為 Int)	第二個引數 (轉換為 Array) 所提 從零開始的索引的項目。如果轉 ，則結果為 Undefined 。如果 Array的邊界之外 (負值或 >= ar gth)，則結果為 Undefined 。
String	任何類型 (轉換為 Int)	第二個引數 (轉換為 Int) 所提供 零開始的索引的字元。如果轉換 結果為 Undefined 。如果索引 界之外 (負值或 >= string.length) Undefined 。
物件	String (未套用轉換)	對應至第二個引數所提供的字串 引數物件所儲存的值。
其他值	任何值	Undefined .

get_dynamodb(tableName, partitionKeyName, partitionKeyValue, sortKeyName, sortKeyValue, roleArn)

從 DynamoDB 資料表擷取資料。get_dynamodb() 允許您在評估規則時查詢 DynamoDB 資料表。您可以使用從 DynamoDB 中擷取的資料來篩選或增強訊息承載。受 SQL 版本 2016-03-23 和更新版本支援。

get_dynamodb() 接受下列參數：

tableName

所要查詢 DynamoDB 資料表的名稱。

partitionKeyName

分割區索引鍵的名稱。如需詳細資訊，請參閱 DynamoDB 索引鍵。

partitionKeyValue

用來識別記錄的分割區索引鍵值。如需詳細資訊，請參閱 DynamoDB 索引鍵。

sortKeyName

(選用) 排序索引鍵的名稱。只有在查詢的 DynamoDB 資料表使用複合索引鍵時，才需要此參數。如需詳細資訊，請參閱 DynamoDB 索引鍵。

sortKeyValue

(選用) 排序索引鍵的值。只有在查詢的 DynamoDB 資料表使用複合索引鍵時，才需要此參數。如需詳細資訊，請參閱 DynamoDB 索引鍵。

roleArn

授予 DynamoDB 資料表存取權限的 IAM 角色 ARN。規則引擎會假設此角色代表您存取 DynamoDB 資料表。請避免使用過多許可的角色。僅授與角色規則所需的許可。以下是授與一個 DynamoDB 資料表存取權的範例政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/table-
name"
        }
    ]
}
```

舉例說明如何使用 get_dynamodb()，假設您有一個 DynamoDB 資料表，其中包含所有連接至 AWS IoT之裝置的裝置 ID 和位置資訊。下列 SELECT 陳述式使用 get_dynamodb() 函數來擷取指定裝置 ID 的位置：

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

> **Note**
>
> - 每個 SQL 陳述式最多可以呼叫一次 get_dynamodb()。在單一 SQL 陳述式中多次呼叫 get_dynamodb() 會導致規則終止，而不會呼叫任何動作。

get_mqtt_property(名稱)

參考下列任一 MQTT5 標頭：`contentType`、`payLoadFormatIndicator`、`responseTopic`、和 `correlationData`。此函數會接受下列任何常值字串做為引數：`content_type`、`format_indicator`、`response_topic` 和 `correlation_data`。如需詳細資訊，請參閱下列函數引數表。

ContentType

字串：描述發佈訊息內容的 UTF-8 編碼字串。

payLoadFormatIndicator

字串：列舉字串值，用於表示承載是否已格式化為 UTF-8。有效值為 `UNSPECIFIED_BYTES` 和 `UTF8_DATA`。

responseTopic

字串：UTF-8 編碼字串，用來當作回應訊息的主題名稱。回應主題是用來描述要在請求-回應流程中作為接收者發佈目標的主題。主題不得包含萬用字元。

correlationData

字串：請求訊息的傳送者使用 base64 編碼的二進位資料，用以在收到回應訊息時識別其所對應的請求。

下表列出可接受的函數引數及其相關聯的 `get_mqtt_property` 函數傳回類型：

函數引數

SQL	傳回資料類型 (如果存在)	傳回資料類型 (如果不存在)
`get_mqtt_property( "format_indicator")`	字串 (UNSPECIF IED_BYTES 或 UTF8_DATA)	字串 (NSPECIFIED_BYTES)
`get_mqtt_property( "content_type")`	String	未定義
`get_mqtt_property( "response_topic")`	String	未定義

SQL	傳回資料類型 (如果存在)	傳回資料類型 (如果不存在)
`get_mqtt_property( "correlation_data")`	base64 編碼字串	未定義
`get_mqtt_property( "some_invalid_name ")`	未定義	未定義

下列範例規則 SQL 會參考下列任一 MQTT5 標頭：`contentType`、`payLoadFormatIndicator`、`responseTopic`、和`correlationData`。

```
SELECT *, get_mqtt_property('content_type') as contentType,
          get_mqtt_property('format_indicator') as payloadFormatIndicator,
          get_mqtt_property('response_topic') as responseTopic,
          get_mqtt_property('correlation_data') as correlationData
FROM 'some/topic'
```

get_or_default(expression， defaultValue)

如果指定，則傳回第二個參數中的預設值，否則當第一個參數中的運算式傳回 null、未定義或失敗時，傳回未定義的預設值。受 SQL 版本 2016-03-23 和更新版本支援。

> **Important**
>
> `get_or_default` 不如預期直接支援非 JSON 承載。如果您使用的是非 JSON 承載，請使用 `encode`或 `decode`函數。

`get_or_default()` 接受下列參數：

表達式

任何包含 資料類型、函數、文字、變數巢狀物件查詢、 或 的有效表達式JSON Extensions。

defaultValue

（選用）包含 資料類型、函數、文字、變數巢狀物件查詢、 或 的任何有效表達式JSON Extensions。這是當第一個引數傳回 null、未定義或失敗時要傳回的值。

> **Note**
>
> defaultValue 參數不允許從客戶擁有的資源擷取資料的函數，例如 get_secret、get_dynamodb、aws_lambda、get_thing_shadow 和 decode-protobuf。

下表顯示每個引數及其相關聯輸出可接受的函數引數：

第一個引數	第二個引數	Output
成功的評估	任何值或未指定	第一個引數值。
未定義、Null 或失敗	任何值，包括未定義或 Null	第二個引數值。
未定義、Null 或失敗	未指定	`Undefined`

範例：

範例 1：

如果 DynamoDB 資料表或查詢失敗，下列範例會提供 defaultValue 值：

```
SELECT
    device_id,
    get_or_default(
        get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME"),
        {"mode": "standard", "timeout": 30, "enabled": true }
    ) as config
FROM 'device/telemetry'
```

範例 2：

如果狀態未定義，以下範例會提供安全的預設值 "UNKNOWN"：

```
SELECT
  get_or_default( CASE status
    WHEN 'active' THEN 'GOOD'
    WHEN 'inactive' THEN 'BAD'/
```

```
    ELSE 'UNKNOWN'
  END, 'UNKNOWN') as status_category
FROM 'topic/subtopic'
```

範例 3：

下列範例示範如何搭配單一參數使用 get_or_default。這在您可能沒有明確的預設值，但不希望規則執行失敗的情況下非常有用。

```
SELECT
  get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME") as config
FROM 'device/telemetry'
```

如果 DynamoDB 查詢失敗，規則執行將會失敗，而且不會執行任何動作。如果改用下列 SQL：

```
SELECT
  get_or_default(get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME")) as config
FROM 'device/telemetry'
```

get_or_default 陳述式將評估為 Undefined，因此在此範例中，整體 SELECT 陳述式將評估為 {}，並嘗試任何規則動作。

> **Important**
>
> 建議您遵循這些最佳實務，以維護使用此函數時的安全性：
>
> - 避免在規則定義中使用硬式編碼秘密，包括預設值
> - 使用 AWS Secrets Manager 管理敏感資訊

get_registry_data(registryAPI， thingName， roleArn)

擷取 AWS IoT 規則中的 AWS IoT 物件登錄檔資料。您可以讀取登錄檔資料（例如屬性、物件類型和裝置所屬的物件群組），並使用此資訊來篩選、豐富或動態路由訊息。受 SQL 版本 2016-03-23 和更新版本支援。

`get_registry_data()` 接受下列參數：

registryAPI

正在呼叫的登錄 API。有效值為 `DescribeThing` 和 `ListThingGroupsForThing`。這些值必須是常數字串。

thingName

字串：您要擷取其登錄檔資料之物件的名稱。

roleArn

字串：根據所呼叫 API 具有`iot:DescribeThing`許可和/或`iot:ListThingGroupsForThing`許可的角色 ARN。

`get_registry_data` 函數的回應格式與呼叫的登錄 API 相同。如需詳細資訊，請參閱 DescribeThing 和 ListThingGroupsForThing APIs。

範例：

您可以擷取物件類型資訊，以允許篩選物件類型為 的物件 AWS IoT Core 生命週期事件訊息 （ 物件名稱符合 MQTT 用戶端 ID)`testenv`。

```
SELECT *
FROM '$aws/events/lifecycle/+'
WHERE
    get_registry_data("DescribeThing",clientId,[roleArn]).thingTypeName='testenv'
```

範例：

您可以為閘道裝置 傳送`sensor1`的所有訊息擷取物件名稱的裝置物件屬性`gateway1`。

```
SELECT *, get_registry_data("DescribeThing","sensor1",
[roleArn]).attributes.temperature_threhold AS device1_tempthreshold
FROM home1/gateway1/sensor1/#
```

> **Note**
>
> - 每個 SQL 陳述式和動作和錯誤動作的替代範本`get_registry_data()`最多可以呼叫一次。

get_secret(secretId, secretType, key, roleArn)

擷取所加密 `SecretString` 或 `SecretBinary` 欄位的值，此欄位是 AWS Secrets Manager 中目前秘密版本的欄位。如需建立和維護秘密的詳細資訊，請參閱 CreateSecret、UpdateSecret 和 PutSecretValue。

`get_secret()` 接受下列參數：

secretId

字串：要擷取之秘密的 Amazon 資源名稱 (ARN) 或易記名稱。

secretType

字串：秘密類型。有效值：`SecretString` | `SecretBinary`。

SecretString

- 對於您使用 APIs AWS CLI、 或 AWS Secrets Manager 主控台建立為 JSON 物件的秘密：
  - 如果您指定 `key` 參數的值，此函數會傳回所指定金鑰的值。
  - 如果未指定 `key` 參數的值，此函數會傳回整個 JSON 物件。
- 對於您使用 API 或 AWS CLI建立為非 JSON 物件的秘密：
  - 如果您指定 `key` 參數的值，此函數會失敗並顯示例外狀況。
  - 如果未指定 `key` 參數的值，此函數會傳回秘密的內容。

SecretBinary

- 如果您指定 `key` 參數的值，此函數會失敗並顯示例外狀況。
- 如果未指定 `key` 參數的值，此函數會以 Base64 編碼的 UTF-8 字串形式傳回秘密值。

key

(選用) 字串：JSON 物件內的金鑰名稱，此物件存放在秘密的 `SecretString` 欄位中。當您只想要擷取存放在秘密中的秘密值，而不是整個 JSON 物件時，請使用此值。

如果您指定此參數的值，而且秘密並未在其 `SecretString` 欄位內包含 JSON 物件，此函數會失敗並顯示例外狀況。

roleArn

字串：擁有 `secretsmanager:GetSecretValue` 和 `secretsmanager:DescribeSecret` 許可的角色 ARN。

> **Note**
>
> 此函數一律傳回目前版本的秘密 (標籤為 AWSCURRENT 的版本)。 AWS IoT 規則引擎會快取每個秘密最多 15 分鐘。因此，規則引擎最多可能需要 15 分鐘來更新秘密。這表示如果您在使用更新後最多 15 分鐘內擷取秘密 AWS Secrets Manager，此函數可能會傳回先前的版本。
> 此函數不會計量，但會 AWS Secrets Manager 收取費用。由於秘密快取機制，規則引擎偶爾會呼叫 AWS Secrets Manager。因為規則引擎是完全分散式服務，您可能會在 15 分鐘快取時間範圍期間，看到多個來自規則引擎的 Secrets Manager API 呼叫。

範例：

您可以在 HTTPS 規則動作的身分驗證標題中使用 `get_secret` 函式，如下列 API 金鑰身分驗證範例所示。

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

如需 HTTPS 規則動作的詳細資訊，請參閱 the section called “HTTP”。

get_thing_shadow(thingName， shadowName， roleArn)

傳回指定物件的影子。受 SQL 版本 2016-03-23 和更新版本支援。

thingName

字串：想要擷取影子的物件名稱。

shadowName

(選用) 字串：影子的名稱。只有在參考已命名的影子時，才需要此參數。

roleArn

字串：擁有 `iot:GetThingShadow` 許可的角色 ARN。

範例：

與已命名的影子搭配使用時，請提供 `shadowName` 參數。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

與未命名影子搭配使用時，請省略 shadowName 參數。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

get_user_properties(userPropertyKey)

參考使用者屬性，這是 MQTT5 支援的一種屬性標題類型。

userProperty

字串：使用者屬性是索引鍵/值對。此函數將索引鍵當作參數，並傳回所有符合關聯索引鍵的值陣列。

函數引數

對於下列位於訊息標頭中的使用者屬性：

金鑰	值
部分索引鍵	部分值
不同的索引鍵	不同的值
部分索引鍵	具有重複索引鍵的值

下表顯示預期的 SQL 行為：

SQL	傳回資料類型。	傳回資料類型。
get_user_properties(「部分索引鍵」)	字串陣列	`['some value', 'value with duplicate key']`
get_user_properties(「其他索引鍵」)	字串陣列	`['a different value']`
get_user_properties( )	索引鍵/值對物件的陣列	`[{'"some key": "some value"'}, {"other key": "a different value"}, {"some key": "value with duplicate key"}]`
get_user_properties(「不存在的索引鍵」)	未定義	

下列範例規則 SQL 會將使用者屬性 (MQTT5 屬性標頭類型) 參考到承載中：

```
SELECT *, get_user_properties('user defined property key') as userProperty
FROM 'some/topic'
```

雜湊函數

AWS IoT 提供下列雜湊函數：

- md2
- md5
- sha1
- sha224
- sha256
- sha384
- sha512

所有雜湊函數都預期有一個字串引數。結果為該字串的雜湊值。標準字串轉換會套用至非字串的引數。SQL 版本 2015-10-08 和更新版本可支援所有雜湊函數。

範例：

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(String, String)

傳回第二個引數的第一個索引 (從 0 開始)，作為第一個引數的子字串。預期兩個引數均為字串。非字串的引數需遵守標準字串轉換規則。此函數不能套用在陣列上，僅可套用到字串上。受 SQL 版本 2016-03-23 和更新版本支援。

範例：

`indexof("abcd", "bc")` = 1

isNull()

如果引數是 `Null` 值，則傳回 true 受 SQL 版本 2016-03-23 和更新版本支援。

範例：

`isNull(5)` = false。

`isNull(Null)` = true。

引數類型	結果
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	true

引數類型	結果
Undefined	false

isUndefined()

如果引數是 Undefined，則傳回 true。受 SQL 版本 2016-03-23 和更新版本支援。

範例：

isUndefined(5) = false。

isUndefined(floor([1,2,3])) = true。

引數類型	結果
Int	false
Decimal	false
Boolean	false
String	false
Array	false
Object	false
Null	false
Undefined	true

length(String)

傳回所提供的字串內的字元數。標準轉換規則會套用至非 String 的引數。受 SQL 版本 2016-03-23 和更新版本支援。

範例：

length("hi") = 2

`length(false)` = 5

ln(Decimal)

傳回引數的自然對數。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`ln(e)` = 1。

引數類型	結果
Int	Decimal (使用雙精度)，引數的自然對數。
Decimal	Decimal (使用雙精度)，引數的自然對數。
Boolean	Undefined 。
String	Decimal (使用雙精度)，引數的自然對數。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined 。
物件	Undefined 。
Null	Undefined 。
未定義	Undefined 。

log(Decimal)

傳回引數以 10 為底的對數。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`log(100)` = 2.0。

引數類型	結果
Int	Decimal (使用雙精度)，引數以 10 為底的對數。

引數類型	結果
Decimal	Decimal (使用雙精度)，引數以 10 為底的對數。
Boolean	Undefined 。
String	Decimal (使用雙精度)，引數以 10 為底的對數。如果 String 無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined 。
物件	Undefined 。
Null	Undefined 。
未定義	Undefined 。

lower(String)

傳回小寫版本的特定 String。使用標準轉換規則將非字串引數轉換為字串。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

lower("HELLO") =「hello」。

lower(["HELLO"]) = "[\"hello\"]".

lpad(String, Int)

傳回 String 引數，左側填入第二個引數所指定的空格數。Int 引數必須介於 0 到 1000 之間。如果提供的值超出此有效範圍，則引數將設為最接近的有效值 (0 或 1000)。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

lpad("hello", 2) = " hello".

lpad(1, 3) = " 1"

引數類型 1	引數類型 2	結果
String	Int	String，所給的 String 左側填 給的 Int 的數量的空格。
String	Decimal	Decimal 引數將無條件捨去至最 Int，而 String 會在左側填充 數。
String	String	第二個引數會轉換為 Decimal 並 去至最接近的 Int，而 String 入指定的空格數。如果第二個引 為 Int，則結果為 Undefined
其他值	Int/Decimal/String	第一個值會使用標準轉換轉為 St 後 LPAD 函數會套用至該 Strin 無法轉換，則結果為 Undefine
任何值	其他值	Undefined .

ltrim(String)

移除所給的 String 前方所有空格 (tab 與空格)。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

Ltrim(" h i ") = "hi "。

引數類型	結果
Int	移除 Int 前方所有空格的 String 顯示方式。
Decimal	移除 Decimal 前方所有空格的 String 顯示方式。
Boolean	移除布林值 (「true」或「false」) 前方所有空格的 String 顯示方式。

引數類型	結果
String	移除前方所有空格的引數。
陣列	String (使用標準轉換規則) 的 Array 顯示方式，且移除所有前置空格。
物件	Object (使用標準轉換規則) 的 String 顯示方式，且移除所有前置空格。
Null	Undefined 。
未定義	Undefined 。

mod(Decimal, Decimal)

傳回第一個引數除以第二的引數的餘數。等同於 remainder(Decimal, Decimal)。也可以使用「%」當做同樣的模除功能的 infix 運算子。受 SQL 版本 2015-10-08 和更新版本支援。

範例：mod(8, 3) = 2。

左運算元	右運算元	Output
Int	Int	Int，第一個引數模除第二個引
Int/Decimal	Int/Decimal	Decimal，第一個引數模除第二
String/Int/Decimal	String/Int/Decimal	如果所有的字串都轉換為小數，第一個引數以第二個引數為模。Undefined 。
其他值	其他值	Undefined 。

nanvl(AnyValue, AnyValue)

若第一個引數為有效的 Decimal，即傳回。否則便傳回第二個引數。受 SQL 版本 2015-10-08 和更新版本支援。

範例：Nanvl(8, 3) = 8。

引數類型 1	引數類型 2	Output
未定義	任何值	第二個參數。
Null	任何值	第二個參數。
Decimal (NaN)	任何值	第二個參數。
Decimal (非 NaN)	任何值	第一個參數。
其他值	任何值	第一個參數。

newuuid()

傳回隨機的 16 位元組 UUID。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`newuuid()` = `123a4567-b89c-12d3-e456-789012345000`

numbytes(String)

傳回所給字串 UTF-8 編碼中的位元組數。標準轉換規則會套用至非 `String` 的引數。受 SQL 版本 2016-03-23 和更新版本支援。

範例：

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long[, String])

使用 `parse_time` 函數來設定時間戳記格式，變成人類易懂的日期/時間格式。受 SQL 版本 2016-03-23 和更新版本支援。若要將時間戳記字串轉換為毫秒，請參閱 time_to_epoch(String, String)。

`parse_time` 函數預期下列引數：

pattern

(字串) 遵循 Joda-Time 格式的日期/時間模式。

timestamp

(Long) 自 Unix epoch 起以毫秒單位設定格式的時間。請參閱函數 timestamp()。

timezone

(字串) 設定好格式的日期/時間的時區。預設值為「UTC」。該函數支援 Joda-Time 時區。此為選用引數。

範例：

在此訊息發佈至主題「A/B」時，承載 {"ts": "1970.01.01 AD at 21:46:40 CST"} 會傳送至 S3 儲存貯體：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", 100000000,
 'America/Belize' ) as ts FROM 'A/B'",

        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

在此訊息發佈至主題「A/B」時，與 {"ts": "2017.06.09 AD at 17:19:46 UTC"} 相似，但包含目前日期/時間的承載會傳送至 S3 儲存貯體：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", timestamp() ) as ts
 FROM 'A/B'",
```

```
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

parse_time() 也可以用作替代範本。例如，當此訊息發佈至主題「A/B」時，該承載會傳送至金鑰 =「2017」的 S3 儲存貯體：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT * FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [{
            "s3": {
                "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                "bucketName": "BUCKET_NAME",
                "key": "${parse_time('yyyy', timestamp(), 'UTC')}"
            }
        }],
        "ruleName": "RULE_NAME"
    }
}
```

power(Decimal, Decimal)

傳回第一個引數次方的第二個引數值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。受 SQL 版本 2015-10-08 和更新版本支援。

範例：power(2, 5) = 32.0。

引數類型 1	引數類型 2	Output
`Int/Decimal`	`Int/Decimal`	`Decimal` (使用雙精度)，第一個 第二個引數值。
`Int/Decimal/String`	`Int/Decimal/String`	`Decimal` (使用雙精度)，第一個 第二個引數值。任何轉換為小數 果任何 `String` 無法轉換為 `Dec` 結果為 `Undefined` 。
其他值	其他值	`Undefined` .

principal()

根據發佈觸發訊息的方式，傳回裝置用於身分驗證的委託人。下表說明為各發佈方法和通訊協定傳回的委託人。

訊息發佈方式	通訊協定	憑證類型
MQTT 用戶端	MQTT	X.509 裝置憑證
AWS IoT 主控台 MQTT 用戶端	MQTT	IAM 使用者或角色
AWS CLI	HTTP	IAM 使用者或角色
AWS IoT 裝置 SDK	MQTT	X.509 裝置憑證
AWS IoT 裝置 SDK	MQTT over WebSocket	IAM 使用者或角色

以下範例顯示 `principal()` 可以傳回哪些不同類型的值：

- X.509 憑證指紋：`ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373`
- IAM 角色 ID 和工作階段名稱：`ABCD1EFG3HIJK2LMNOP5:my-session-name`
- 傳回使用者 ID：`ABCD1EFG3HIJK2LMNOP5`

rand()

傳回虛擬亂數，平均分散在 0.0 和 1.0。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`rand()` = 0.8231909191640703

regexp_matches(String, String)

如果字串 (第一個引數) 包含規則表達式的相符項目 (第二個引數)，則傳回 true。如果您在規則表達式|中使用 ，請搭配 使用()。

範例：

`regexp_matches("aaaa", "a{2,}")` = true。

`regexp_matches("aaaa", "b")` = false。

`regexp_matches("aaa", "(aaa|bbb)")` = true。

`regexp_matches("bbb", "(aaa|bbb)")` = true。

`regexp_matches("ccc", "(aaa|bbb)")` = false。

第一個引數：

引數類型	結果
`Int`	`Int` 的 `String` 顯示方式。
`Decimal`	`Decimal` 的 `String` 顯示方式。
`Boolean`	布林值 (「true」或「false」) 的 `String` 顯示方式。
`String`	`String`。
陣列	`Array` (使用標準轉換規則) 的 `String` 顯示方式。
物件	Object (使用標準轉換規則) 的 `String` 顯示方式。

引數類型	結果
Null	Undefined .
未定義	Undefined .

第二個引數：

必須為有效的 regex 表達式。非字串類型都會使用標準轉換規則轉換為 String。根據類型，結果字串可能不是有效的正規運算式。如果 (轉換的) 引數並非有效的 regex 值，結果會為 Undefined。

regexp_replace(String, String, String)

以第三個引數取代所有在第一個引數中出現的第二個參數 (一般表達式)。請以「$」參考擷取群組。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`regexp_replace("abcd", "bc", "x")` = "axd"。

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac"。

第一個引數：

引數類型	結果
Int	Int 的 String 顯示方式。
Decimal	Decimal 的 String 顯示方式。
Boolean	布林值 (「true」或「false」) 的 String 顯示方式。
String	來源值。
陣列	Array (使用標準轉換規則) 的 String 顯示方式。
物件	Object (使用標準轉換規則) 的 String 顯示方式。

引數類型	結果
Null	Undefined .
未定義	Undefined .

第二個引數：

必須為有效的 regex 表達式。非字串類型都會使用標準轉換規則轉換為 String。根據類型，結果字串可能不是有效的正規運算式。如果 (轉換的) 引數並非有效的 regex 表達式，則結果為 Undefined。

第三個引數：

必須為有效的 regex 替換字串。(可以參考擷取群組)。非字串類型都會使用標準轉換規則轉換為 String。如果 (轉換的) 引數並非有效的 regex 替換字串，則結果為 Undefined。

regexp_substr(String, String)

在第一個參數中尋找第一個符合第二個參數 (regex) 的值。請以「$」參考擷取群組。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`regexp_substr("hihihello", "hi")` = "hi"

`regexp_substr("hihihello", "(hi)*")` = "hihi"

第一個引數：

引數類型	結果
Int	Int 的 String 顯示方式。
Decimal	Decimal 的 String 顯示方式。
Boolean	布林值 (「true」或「false」) 的 String 顯示方式。
String	String 引數。
陣列	Array (使用標準轉換規則) 的 String 顯示方式。

引數類型	結果
物件	Object (使用標準轉換規則) 的 String 顯示方式。
Null	Undefined .
未定義	Undefined .

第二個引數：

必須為有效的 regex 表達式。非字串類型都會使用標準轉換規則轉換為 String。根據類型，結果字串可能不是有效的正規運算式。如果 (轉換的) 引數並非有效的 regex 表達式，則結果為 Undefined。

remainder(Decimal, Decimal)

傳回第一個引數除以第二的引數的餘數。等同於 mod(Decimal, Decimal)。也可以使用「%」當做同樣的模除功能的 infix 運算子。受 SQL 版本 2015-10-08 和更新版本支援。

範例：remainder(8, 3) = 2。

左運算元	右運算元	Output
Int	Int	Int，第一個引數模除第二個引
Int/Decimal	Int/Decimal	Decimal，第一個引數模除第二
String/Int/Decimal	String/Int/Decimal	如果所有的字串都轉換為小數，第一個引數以第二個引數為模。Undefined 。
其他值	其他值	Undefined .

replace(String, String, String)

以第三個引數取代所有在第一個引數中出現的第二個引數。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`replace("abcd", "bc", "x")` = `"axd"`.

`replace("abcdabcd", "b", "x")` = `"axcdaxcd"`.

所有引數

引數類型	結果
`Int`	`Int` 的 `String` 顯示方式。
`Decimal`	`Decimal` 的 `String` 顯示方式。
`Boolean`	布林值 (「true」或「false」) 的 `String` 顯示方式。
`String`	來源值。
陣列	`Array` (使用標準轉換規則) 的 `String` 顯示方式。
物件	Object (使用標準轉換規則) 的 `String` 顯示方式。
Null	`Undefined` .
未定義	`Undefined` .

rpad(String, Int)

傳回字串引數，右側填入第二個引數所指定的空格數。`Int` 引數必須介於 0 到 1000 之間。如果提供的值超出此有效範圍，則引數將設為最接近的有效值 (0 或 1000)。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`rpad("hello", 2)` = `"hello  "`.

`rpad(1, 3)` = `"1  "`.

引數類型 1	引數類型 2	結果
String	Int	String 在右側填入，且空格數量等同所給的 Int。
String	Decimal	Decimal 引數會無條件捨去至最接近的 Int，並且該字串將以提供的 Int 相同的空格數填入右側。
String	String	第二個引數將轉換為 Decimal，並無條件捨去至最接近的 Int。String 在右側填入，且空格數量等同 Int 的值。
其他值	Int/Decimal/String	第一個值會使用標準轉換來轉換為 String，而 rpad 函數將套用到該 String 上。如果其無法轉換，則結果為 Undefined 。
任何值	其他值	Undefined .

round(Decimal)

將指定的 Decimal 無條件進位至最接近的 Int。如果 Decimal 與兩個 Int 值 (例如，0.5) 等距，則 Decimal 會無條件進位。受 SQL 版本 2015-10-08 和更新版本支援。

範例：Round(1.2) = 1。

Round(1.5) = 2。

Round(1.7) = 2。

Round(-1.1) = -1。

Round(-1.5) = -2。

引數類型	結果
Int	引數。
Decimal	Decimal 是要向下捨入到最接近的 Int。
String	Decimal 是要向下捨入到最接近的 Int。如果字串無法轉換為 Decimal，則結果為 Undefined 。
其他值	Undefined .

rtrim(String)

移除所給的 String 後方所有空格 (tab 與空格)。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

rtrim(" h i ") = " h i"

引數類型	結果
Int	Int 的 String 顯示方式。
Decimal	Decimal 的 String 顯示方式。
Boolean	布林值 (「true」或「false」) 的 String 顯示方式。
陣列	Array (使用標準轉換規則) 的 String 顯示方式。
物件	Object (使用標準轉換規則) 的 String 顯示方式。
Null	Undefined .

引數類型	結果
未定義	Undefined

sign(Decimal)

傳回所給數字的符號。當引數的符號為正值時，傳回 1。當引數的符號為負值時，傳回 -1。如果引數為 0，傳回 0。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`sign(-7)` = -1。

`sign(0)` = 0。

`sign(13)` = 1。

引數類型	結果
Int	Int，Int 值的符號。
Decimal	Int，Decimal 值的符號。
String	Int，Decimal 值的符號。此字串會轉換為 Decimal 值，並傳回 Decimal 值的符號。如果 String 無法轉換為 Decimal，則結果為 Undefined 。受 SQL 版本 2015-10-08 和更新版本支援。
其他值	Undefined .

sin(Decimal)

以弧度傳回數字的正弦值。Decimal 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`sin(0)` = 0.0

引數類型	結果
Int	Decimal (使用雙精度)，引數的正弦值。
Decimal	Decimal (使用雙精度)，引數的正弦值。
Boolean	Undefined 。
String	Decimal (使用雙精度)，引數的正弦值。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined 。
物件	Undefined 。
Null	Undefined 。
Undefined	Undefined 。

sinh(Decimal)

以弧度傳回數字的雙曲正弦值。Decimal 值在套用函數前會四捨五入至雙精度。結果為雙精度的 Decimal 值。受 SQL 版本 2015-10-08 和更新版本支援。

範例：sinh(2.3) = 4.936961805545957

引數類型	結果
Int	Decimal (使用雙精度)，引數的雙曲正弦值。
Decimal	Decimal (使用雙精度)，引數的雙曲正弦值。
Boolean	Undefined 。

引數類型	結果
String	Decimal (使用雙精度)，引數的雙曲正弦值。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

sourceip()

擷取裝置或與其連線之路由器的 IP 地址。如果您的裝置直接連線至網際網路，則此函數會傳回裝置的來源 IP 地址。如果您的裝置是連線到連線網際網路的路由器，則此函數會傳回路由器的來源 IP 地址。SQL 2016-03-23 版可支援。sourceip( ) 不會採用任何參數。

> **Important**
>
> 裝置的公有來源 IP 地址通常是最後一個網路位址轉譯 (NAT) 閘道的 IP 地址，例如，您的網際網路服務供應商的路由器或有線數據機。

範例：

```
sourceip()="192.158.1.38"
```

```
sourceip()="1.102.103.104"
```

```
sourceip()="2001:db8:ff00::12ab:34cd"
```

SQL 範例：

```
SELECT *, sourceip() as deviceIp FROM 'some/topic'
```

如何在 AWS IoT Core 規則動作中使用 sourceip() 函數的範例：

範例 1

下列範例顯示如何在 DynamoDB 動作中呼叫 () 函數作為替換範本。

```
{
 "topicRulePayload": {
  "sql": "SELECT * AS message FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "dynamoDB": {
     "tableName": "my_ddb_table",
     "hashKeyField": "key",
     "hashKeyValue": "${sourceip()}",
     "rangeKeyField": "timestamp",
     "rangeKeyValue": "${timestamp()}",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
    }
   }
  ]
 }
}
```

範例 2

下列範例顯示如何使用替換範本新增 sourceip() 函數作為 MQTT 使用者屬性。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "republish": {
     "topic": "${topic()}/republish",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
     "headers": {
      "payloadFormatIndicator": "UTF8_DATA",
      "contentType": "rule/contentType",
      "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
      "userProperties": [
```

```
        {
         "key": "ruleKey1",
         "value": "ruleValue1"
        },
        {
         "key": "sourceip",
         "value": "${sourceip()}"
        }
       ]
      }
     }
    }
   ]
  }
}
```

您可以從訊息中介裝置和基本擷取路徑傳遞至 AWS IoT Core 規則的訊息擷取來源 IP 地址。您也可以擷取 IPv4 和 IPv6 訊息的來源 IP。來源 IP 顯示如下：

IPv6：`yyyy:yyyy:yyyy::yyyy:yyyy`

IPv4：`xxx.xxx.xxx.xxx`

> **Note**
>
> 原始來源 IP 不會透過重新發布動作傳遞。

substring(String, Int[, Int])

預期 `String` 後有一或兩個 `Int` 值。若為 `String` 和單個 `Int` 引數，此函數會從所給的 `String` 索引 (從 0 開始，包含 0) 到 `Int` 的結尾傳回所給的 `String` 的子字串。若為 `String` 和兩個 `Int` 引數，該函數會傳回從第一個 `String` 索引引數 (從 0 開始，包含 0) 到第二個 `Int` 索引引數 (從 0 開始，不含 0) 所提供的所給 `Int` 的子字串。少於零的索引將設為零。比 `String` 長度還長的索引會設為 `String` 長度。至於第三個引數版本，如果第一個索引大於 (或等於) 第二個索引，則結果為空的 `String`。

如果提供的引數不是 (*##*、*##*) 或 (*##*、*##*、*##*)，則標準轉換會套用至該引數，以嘗試將那些引數轉換為正確類型。如果類型無法轉換，函數的結果即為 `Undefined`。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`substring("012345", 0)` = "012345"。

`substring("012345", 2)` = "2345"。

`substring("012345", 2.745)` = "2345"。

`substring(123, 2)` = "3"。

`substring("012345", -1)` = "012345"。

`substring(true, 1.2)` = "true"。

`substring(false, -2.411E247)` = "false"。

`substring("012345", 1, 3)` = "12"。

`substring("012345", -50, 50)` = "012345"。

`substring("012345", 3, 1)` = "".

sql_version()

傳回此規則中指定的 SQL 版本。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`sql_version()` = "2016-03-23"

sqrt(Decimal)

傳回數字的平方根。`Decimal` 引數在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`sqrt(9)` = 3.0。

引數類型	結果
`Int`	引數的平方根。

引數類型	結果
Decimal	引數的平方根。
Boolean	Undefined .
String	引數的平方根。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

startswith(String, String)

傳回 Boolean，無論第一個字串引數的開頭是否為第二個引數。如果引數為 Null 或 Undefined，則結果為 Undefined。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

startswith("ranger","ran") = true

引數類型 1	引數類型 2	結果
String	String	無論第一個字串的開頭是否為第
其他值	其他值	所有引數都將使用標準轉換規則字串。如果第一個字串的開頭是字串，則傳回 true。如果引數為 Undefined ，則結果為 Undef

tan(Decimal)

以弧度傳回數字的正切值。Decimal 值在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：tan(3) = -0.1425465430742778

引數類型	結果
Int	Decimal (使用雙精度)，引數的正切值。
Decimal	Decimal (使用雙精度)，引數的正切值。
Boolean	Undefined 。
String	Decimal (使用雙精度)，引數的正切值。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined 。
物件	Undefined 。
Null	Undefined 。
未定義	Undefined 。

tanh(Decimal)

以弧度傳回數字的雙曲正切值。Decimal 值在套用函數前會四捨五入至雙精度。受 SQL 版本 2015-10-08 和更新版本支援。

範例：tanh(2.3) = 0.9800963962661914

引數類型	結果
Int	Decimal (使用雙精度)，引數的雙曲正切值。
Decimal	Decimal (使用雙精度)，引數的雙曲正切值。
Boolean	Undefined 。

引數類型	結果
String	Decimal (使用雙精度)，引數的雙曲正切值。如果字串無法轉換為 Decimal，則結果為 Undefined 。
Array	Undefined .
物件	Undefined .
Null	Undefined .
未定義	Undefined .

time_to_epoch(String, String)

使用 time_to_epoch 函數將時間戳字串轉換為 Unix epoch 時間中的毫秒數。受 SQL 版本 2016-03-23 和更新版本支援。若要將毫秒轉換為格式化的時間戳記字串，請參閱 parse_time(String, Long[, String])。

time_to_epoch 函數預期下列引數：

timestamp

(字串) 自 Unix epoch 起要轉換為毫秒的時間戳記字串。如果時間戳記字串未指定時區，函數會使用 UTC 時區。

pattern

(字串) 遵循 JDK11 時間格式的日期/時間模式。

範例：

time_to_epoch("2020-04-03 09:45:18 UTC+01:00", "yyyy-MM-dd HH:mm:ss VV") = 1585903518000

time_to_epoch("18 December 2015", "dd MMMM yyyy") = 1450396800000

time_to_epoch("2007-12-03 10:15:30.592 America/Los_Angeles", "yyyy-MM-dd HH:mm:ss.SSS z") = 1196705730592

timestamp()

傳回從 1970 年 1 月 1 日星期四 00：00：00 國際標準時間 (UTC) 開始的目前時間戳記，如 AWS IoT 規則引擎所觀察。受 SQL 版本 2015-10-08 和更新版本支援。

範例：`timestamp()` = `1481825251155`

topic(Decimal)

傳回觸發該規則的訊息要傳送至的主題。如果未指定參數，則傳回整個主題。`Decimal` 參數用於指定特定主題區段，並會使用 1 指定第一個區段。對於 topic `foo/bar/baz`，topic(1) 會傳回 `foo`，topic(2) 會傳回 `bar`，依此類推。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`topic()` = "things/myThings/thingOne"

`topic(1)` = "things"

使用基本擷取時，topic() 函數無法使用主題的初始字首 (`$aws/rules/`*`rule-name`*)。例如，假定主題為：

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2"

traceid()

傳回 MQTT 的追蹤 ID (UUID)，如果訊息不是透過 MQTT 傳送，則為 `Undefined`。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

`traceid()` = "12345678-1234-1234-1234-123456789012"

transform(String, Object, Array)

傳回物件的陣列，其中包含 `Array` 參數上 `Object` 參數的所指定轉換結果。

受 SQL 版本 2016-03-23 和更新版本支援。

String

要使用的轉換模式。請參閱下表以取得支援的轉換模式，以及了解它們如何從 Object 和 Array 參數建立 Result。

物件

包含要套用至每個 Array 元素之屬性的物件。

陣列

Object 屬性套用至其中的物件陣列。

此陣列中的每個物件都對應於函數回應中的物件。函數回應中的每個物件都包含存在於原始物件的屬性，以及 Object 所提供的屬性，這是由 String 中指定的轉換模式所決定。

String 參數	**Object** 參數	**Array** 參數	結果
enrichArray	物件	物件的陣列	物件的陣列，其中每個物件都包含來自 Array 參數的元素屬性，以及 Object 參數的屬性。
任何其他值	任何值	任何值	未定義

> **Note**
>
> 此函數傳回的陣列限制為 128 KiB。

轉換函數範例 1

此範例顯示 transform() 函數如何從一個資料物件和一個陣列產生單一物件陣列。

在此範例中，下列訊息會發佈至 MQTT 主題 A/B。

```
{
```

```
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

主題規則動作的這個 SQL 陳述式會使用 transform() 函數與 enrichArray 的 String 值搭配。在此範例中，Object 是來自訊息承載的 attributes 屬性，而 Array 是 values 陣列，其中包含三個物件。

```
select value transform("enrichArray", attributes, values) from 'A/B'
```

在收到訊息承載時，SQL 陳述式會評估為下列回應。

```
[
  {
    "a": 3,
    "data1": 1,
    "data2": 2
  },
  {
    "b": 4,
    "data1": 1,
    "data2": 2
  },
  {
    "c": 5,
    "data1": 1,
    "data2": 2
  }
]
```

轉換函數範例 2

此範例顯示 transform() 函數如何使用文字值，以包含並重新命名來自訊息承載的個別屬性。

在此範例中，下列訊息會發佈至 MQTT 主題 A/B。這是用於 the section called “轉換函數範例 1” 的同一則訊息。

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

主題規則動作的這個 SQL 陳述式會使用 transform() 函數與 `enrichArray` 的 `String` 值搭配。transform() 函數中的 `Object` 在訊息承載中有一個名為 `key` 且值為 `attributes.data1` 的單一屬性，而 `Array` 是 `values` 陣列，其中包含上述範例中使用的三個物件。

```
select value transform("enrichArray", {"key": attributes.data1}, values) from 'A/B'
```

在收到訊息承載時，此 SQL 陳述式會評估為下列回應。請注意，`data1` 屬性在回應中名為 `key`。

```
[
  {
    "a": 3,
    "key": 1
  },
  {
    "b": 4,
    "key": 1
  },
```

```
  {
    "c": 5,
    "key": 1
  }
]
```

轉換函數範例 3

此範例顯示 transform() 函數如何用於巢狀 SELECT 子句中，以選取多個屬性並建立新的物件，以供後續處理。

在此範例中，下列訊息會發佈至 MQTT 主題 A/B。

```
{
  "data1": "example",
  "data2": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
  }
}
```

此轉換函數的 Object 是 SELECT 陳述式傳回的物件，其中包含訊息 data2 物件的 a 和 b 元素。Array 參數由來自原始訊息中 data2.c 陣列的兩個物件組成。

```
select value transform('enrichArray', (select a, b from data2), (select value c from
 data2)) from 'A/B'
```

在收到上述訊息時，SQL 陳述式會評估為下列回應。

```
[
  {
    "x": {
      "someInt": 5,
      "someString": "hello"
    },
    "y": true,
    "a": "first attribute",
    "b": "second attribute"
  },
  {
    "x": {
      "someInt": 10,
      "someString": "world"
    },
    "y": false,
    "a": "first attribute",
    "b": "second attribute"
  }
]
```

此回應中傳回的陣列可與支援 batchMode 的主題規則動作搭配使用。

trim(String)

移除所給的 String 前方和後方所有空格。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

Trim(" hi ") = "hi"

引數類型	結果
Int	移除 Int 前方和後方所有空格的 String 顯示方式。
Decimal	移除 Decimal 前方和後方所有空格的 String 顯示方式。

引數類型	結果
Boolean	移除 Boolean (「true」或「false」) 前方和後方所有空格的 String 顯示方式。
String	移除前方和後方所有空格的 String。
陣列	Array (使用標準轉換規則) 的 String 顯示方式。
物件	Object (使用標準轉換規則) 的 String 顯示方式。
Null	Undefined .
未定義	Undefined .

trunc(Decimal, Int)

將第一個引數截為第二的引數所指定的 Decimal 位數。若第二個引數少於零，則它會設為零。若第二個引數大於 34，則它會設為 34。結果的結尾去掉了零。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

trunc(2.3, 0) = 2。

trunc(2.3123, 2) = 2.31。

trunc(2.888, 2) = 2.88。

trunc(2.00, 5) = 2。

引數類型 1	引數類型 2	結果
Int	Int	來源值。
Int/Decimal	Int/Decimal	第一個引數會截短為第二個引數度。第二個引數如果不是 Int，捨去至最接近的 Int。

引數類型 1	引數類型 2	結果
Int/Decimal/String	Int/Decimal	第一個引數會截短為第二個引數 度。第二個引數如果不是 Int， 件捨去至最接近的 Int。Strin Decimal 值。如果字串轉換失則 Undefined 。
其他值		Undefined .

upper(String)

傳回大寫版本的特定 String。非 String 引數都會使用標準轉換規則轉換為 String。受 SQL 版本 2015-10-08 和更新版本支援。

範例：

upper("hello") = "HELLO"

upper(["hello"]) = "[\"HELLO\"]"

文字

可以直接在規則 SQL 的 SELECT 和 WHERE 子句中指定文字物件，很適合用於傳遞資訊。

> **Note**
>
> 只有在使用 2016-03-23 版本或更高版本的 SQL 時才能使用文字。

會使用 JSON 物件語法 (金鑰/值對，以逗號分隔，索引鍵為字串而值為 JSON 值，以大括號 {} 括起)。例如：

發佈在主題 topic/subtopic 的傳入承載：{"lat_long": [47.606,-122.332]}

SQL 陳述式：SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'

產生的傳出承載為：{"lat_long":{"latitude":47.606,"longitude":-122.332}}。

您也可以直接在規則 SQL 的 SELECT 和 WHERE 子句中指定陣列，即可將資訊分組。會使用 JSON 語法 (將逗號分隔項目用方括號 [] 括起，以建立陣列文字)。例如：

發佈在主題 topic/subtopic 的傳入承載：{"lat": 47.696, "long": -122.332}

SQL 陳述式：SELECT [lat,long] as lat_long FROM 'topic/subtopic'

產生的輸出承載為：{"lat_long": [47.606,-122.332]}。

案例陳述式

可用於分支執行的案例陳述式，例如切換陳述式。

語法：

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

表達式 *v* 會受到評估並對每個 WHEN 子句的 *t[i]* 值配對是否相等。如果找到匹配的結果，相對應的 *r[i]* 表達式會成為該 CASE 陳述式的結果。此 WHEN 子句會按順序進行評估，以便如果有多個符合子句，則第一個符合子句的結果會成為 CASE 陳述式的結果。如果沒有相符項目，ELSE 子句的 *r[e]* 即為結果。如果沒有相符項目，也沒有 ELSE 子句，則結果即為 Undefined。

CASE 陳述式會要求至少一個 WHEN 子句。ELSE 子句是選用的。

例如：

發佈在主題 topic/subtopic 的傳入承載：

```
{
    "color":"yellow"
}
```

SQL 陳述式：

```
SELECT CASE color
        WHEN 'green' THEN 'go'
        WHEN 'yellow' THEN 'caution'
        WHEN 'red' THEN 'stop'
        ELSE 'you are not at a stop light' END as instructions
```

```
    FROM 'topic/subtopic'
```

產生的輸出承載為：

```
{
    "instructions":"caution"
}
```

> **Note**
>
> 如果 *v* 是 Undefined，則案例陳述式的結果為 Undefined。

JSON Extensions

您可以使用下列 ANSI SQL 的延伸模組，協助使用巢狀 JSON 物件。

"." 運算子

運算子會存取嵌入式 JSON 物件的成員，功能與 ANSI SQL 和 JavaScript 完全相同。例如：

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

從傳送至 topic/subtopic 主題的下列訊息承載中選取 foo 物件中 bar 屬性的值。

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

產生的傳出承載為：

```
{
  "bar": {
    "baz": "RED"
  }
}
```

如果 JSON 屬性名稱包含連字元或數字字元，「點」標記法將無法運作。您必須改用 get 函數 來擷取屬性的值。

在此範例中，下列訊息會傳送至 iot/rules 主題。

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

正常情況下，將識別 my-key 的值，如在此查詢中一般。

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

不過，由於屬性名稱 my-key 包含連字號，且 item2 包含數字字元，所以 get 函數必須如下列查詢所示一般使用。

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

*** 運算子**

運作方式與 ANSI SQL 的 * 萬用字元相同。此僅會使用在 SELECT 子句，且會建立包含訊息資料的新 JSON 物件。如果訊息承載並非 JSON 格式，* 會以原始位元組的形式傳回整個訊息承載。例如：

```
SELECT * FROM 'topic/subtopic'
```

將函數套用在屬性值上

以下為可能為由裝置發佈的範例 JSON 承載：

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
```

```
        "longitude" : -122.3359976
    }
}
```

以下範例會將函數套用在 JSON 承載內的屬性值上：

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

此次查詢的結果為下列 JSON 物件：

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

替代範本

您可以使用替代範本來擴增觸發規則時傳回的 JSON 資料，並 AWS IoT 執行動作。替代範本的語法是${表達式}，其中表達式可以是 SELECT 子句、WHERE 子句和 AWS IoT 中 支援的任何表達式AWS IoT 規則動作。您可以將此表達式插入規則的動作欄位中，以便動態設定動作。實際上，此功能會取代動作中的資訊片段。這包含了在原始訊息中呈現的函數、運算子和資訊。

> **Important**
>
> 因為替換範本中的運算式與「SELECT ...」陳述式是分開計算的，所以不能參考使用 AS 子句建立的別名。您只能參考原始承載、函數和運算子中呈現的資訊。

如需支援的表達式的詳細資訊，請參閱 AWS IoT SQL 參考。

下列規則動作支援替代範本。每個動作都支援可以取代的不同欄位。

- Apache Kafka
- CloudWatch 警示
- CloudWatch Logs
- CloudWatch 指標
- DynamoDB
- DynamoDBv2

- Elasticsearch
- HTTP
- InfluxDB
- AWS IoT SiteWise
- Kinesis Data Streams
- Firehose
- Lambda
- Location
- OpenSearch
- 重新發佈
- S3
- SNS
- SQS
- 步驟函數
- Timestream

替代範本會顯示在規則內的動作參數中：

```
{
    "sql": "SELECT *, timestamp() AS timestamp FROM 'my/iot/topic'",
    "ruleDisabled": false,
    "actions": [{
        "republish": {
            "topic": "${topic()}/republish",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

如果這個規則是由下列發佈至 `my/iot/topic` 的 JSON 所觸發：

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
```

```
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

然後，此規則會將下列 JSON 發佈至 `my/iot/topic/republish`，以 AWS IoT 取代`${topic()}/republish`：

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

AWS IoT 規則中陣列承載的每個元素替換

透過 AWS IoT，您可以使用標準替代範本`${表達}`式。在規則的 SQL 陳述式轉換訊息之前，此範本會針對裝置發佈訊息承載解決每個訊息一次。如果您的訊息承載是 JSON 陣列，請改用每個元素的序列`@{欄位}`。此序列會針對規則 SQL 陳述式產生的承載的每個元素解析一次。規則動作會為每個元素寫入一個InfluxDB明細協定點。它使用來自相符元素的值。

> **Note**
>
> 這兩個序列會讀取不同的承載，以便解析為相同欄位名稱的不同值。如果規則的 SQL 陳述式重新命名或重塑欄位，則在`@{欄位中}`，您必須參考 SQL 輸出中顯示的欄位名稱。如果 元素不包含 欄位，序列會解析為空值。它不會回復到訊息層級值，並從其寫入的行通訊協定 AWS IoT 中捨棄空白標籤值。

> **Important**
>
> `@{欄位}`序列僅適用於 AWS IoT InfluxDB規則動作組態。它不適用於以下位置：
>
> - 規則 SQL 陳述式 (SELECT、WHERE 或 SET 子句)
> - 任何其他規則動作

- 一個函數。序列不能包含一個，例如 @{upper(room)}。您也無法將序列傳遞至替代範本函數，因為單一組態值不能同時包含 @{...}和 ${...}。
- 規則的錯誤動作，即使該錯誤動作是InfluxDB動作也是一樣。錯誤動作會對訊息執行一次，因此永遠不會解析每個元素。

如果您在這些位置使用它， CreateTopicRule和 ReplaceTopicRule操作會失敗並顯示錯誤。

每個元素序列@{欄位}有下列需求：

- 必須參考單一欄位，例如 @{room}或 @{meta.sensor_id}
- 不得包含函數、算術、比較或常值
- 必須在組態值中最多顯示一次
- 不得出現在與${...}替代範本相同的組態值中
- 必須出現在組態值中，而不是索引鍵 （ 例如，InfluxDB標籤值，而不是標籤索引鍵 ）

將陣列元素映射至InfluxDB點

在此範例中，InfluxDB規則動作會將測量名稱 (tableName) 設定為 ，@{measurement_type}並將location標籤設定為 @{room}：

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "influxDB": {
          "destinationArn": "arn:aws:iot:us-east-1:123456789012:ruledestination/
influxdb/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_influxdb",
          "databaseName": "device_metrics",
          "tableName": "@{measurement_type}",
          "organization": "my_organization",
          "tags": {
            "location": "@{room}"
```

```
          },
          "timestampUnit": "ms"
        }
      }
    ]
  }
}
```

下列陣列承載會觸發規則：

```
[
    { "measurement_type": "temperature", "room": "kitchen", "value": 5,  "timestamp":
 1700000000000 },
    { "measurement_type": "humidity",    "room": "bedroom", "value": 15, "timestamp":
 1700000001000 }
]
```

動作會為陣列的每個元素寫入一個InfluxDB行通訊協定點。每個元素都會解析`@{measurement_type}`為自己的測量名稱和`@{room}`自己的`location`標籤：

```
temperature,location=kitchen value=5i 1700000000000
humidity,location=bedroom value=15i 1700000001000
```

> **Note**
>
> AWS IoT 從欄位集排除`@{欄位}`序列參考的任何欄位。 AWS IoT 會使用該欄位的值做為測量名稱或標籤值。在此範例中， `measurement_type`提供 測量並提供 `room` `location`標籤。 AWS IoT 排除兩者，因此每個點只會保留`value`為 欄位。

上述範例使用 `SELECT *`，因此動作接收的承載符合裝置發佈的承載。如果 SQL 陳述式重新命名欄位，請使用 SQL 輸出中顯示的名稱。例如，如果陳述式包含 `room AS location_name`，請使用 `@{location_name}`而非 `@{room}`。

針對 SQL 陳述式產生的承載進行解決

動作收到的承載不一定是裝置發佈的承載。在此範例中，裝置會發佈包含讀取陣列的物件：

```
{
    "device_id": "sensor-1",
```

```
    "readings": [
        { "measurement_type": "temperature", "room": "kitchen", "value": 5,
 "timestamp": 1700000000000 },
        { "measurement_type": "humidity",    "room": "bedroom", "value": 15,
 "timestamp": 1700000001000 }
    ]
}
```

此規則使用 SELECT VALUE將readings陣列傳回為頂層承載。如需詳細資訊，請參閱the section called “輸出一個 Array 作為最上層物件”。

```
SELECT VALUE readings FROM 'iot/topic'
```

動作會收到下列陣列，即使裝置已發佈 物件。它會寫入與上述範例相同的兩個點，因為每個元素仍然包含 measurement_type、 room和 value。

```
[
    { "measurement_type": "temperature", "room": "kitchen", "value": 5,  "timestamp":
 1700000000000 },
    { "measurement_type": "humidity",    "room": "bedroom", "value": 15, "timestamp":
 1700000001000 }
]
```

> **Note**
>
> device_id 欄位位於裝置發佈的承載中，但不在 SQL 陳述式產生的陣列中。因此，@{device_id}不會在此規則中解析。僅參考出現在產生陣列元素中的欄位。

巢狀物件查詢

您可以使用巢狀 SELECT 子句來查詢陣列和內部 JSON 物件中的屬性。受 SQL 版本 2016-03-23 和更新版本支援。

請考慮下列 MQTT 訊息：

```
{
    "e": [
        { "n": "temperature", "u": "Cel", "t": 1234, "v": 22.5 },
        { "n": "light", "u": "lm", "t": 1235, "v": 135 },
```

```
        { "n": "acidity", "u": "pH", "t": 1235, "v": 7 }
    ]
}
```

Example

您可以使用以下規則將值轉換為新的陣列。

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

該規則會產生以下輸出。

```
{
    "sensors": [
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

使用相同的 MQTT 訊息，您也可以使用下列規則查詢巢狀物件中的特定值。

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

該規則會產生以下輸出。

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

您也可以使用更複雜的規則來平面化輸出。

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

該規則會產生以下輸出。

```
{
    "temperature": 22.5
}
```

使用二進位承載

若要將訊息承載做為原始二進位資料 (而不是 JSON 物件) 處理，可以使用 * 運算子在 SELECT 子句中參考它。

在本主題中：

- 二進位承載範例
- 對 Protobuf 訊息承載進行解碼

二進位承載範例

使用 * 參考作為原始二進位資料的訊息承載時，您可以將資料新增至規則。如果您有空白或 JSON 承載，則產生的承載可以使用規則新增資料。下列顯示支援 SELECT 子句的範例。

- 對於二進位承載，您可以使用下方僅具有一個 * 的 SELECT 子句。
  - `SELECT * FROM 'topic/subtopic'`
  - `SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0`
- 您也可以新增資料並使用下方 SELECT 子句。
  - `SELECT *, principal() as principal, timestamp() as time FROM 'topic/subtopic'`
  - `SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'`
- 您也可以使用這些 SELECT 子句搭配二進位承載使用。
  - 下方項目是指 WHERE 子句中的 device_type。

    ```
    SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
    ```

  - 也支援下方項目。

    ```
    {
    ```

```
 "sql": "SELECT * FROM 'topic/subtopic'",
 "actions": [
  {
   "republish": {
    "topic": "device/${device_id}"
   }
  }
 ]
}
```

下方規則動作不支援二進位承載，因此您必須將它們解碼。

- 對於某些不支援二進位承載輸入 (例如 Lambda 動作) 的規則，您必須解碼二進位承載。如果 Lambda 規則動作是 base64 編碼且在 JSON 承載中，則可以接收二進位資料。您可以將規則變更如下，以此執行此項操作。

```
SELECT encode(*, 'base64') AS data FROM 'my_topic'
```

- SQL 陳述式不支援將字串作為輸入。若要將字串輸入轉換為 JSON，您可以執行下列命令。

```
SELECT decode(encode(*, 'base64'), 'base64') AS payload FROM 'topic'
```

對 Protobuf 訊息承載進行解碼

協定緩衝區（protobuf）是一種開放原始碼資料格式，用於將結構化資料序列化為壓縮二進位形式。其可用於透過網路傳輸資料或將其儲存在檔案中。Protobuf 可讓您以小封包大小和比其他簡訊格式更快的速度傳送資料。 AWS IoT Core Rules 支援 protobuf，方法是提供 decode(value，decodingScheme) SQL 函數，這可讓您將 protobuf 編碼的訊息承載解碼為 JSON 格式，並將其路由至下游服務。本節詳細介紹在 AWS IoT Core 規則中設定 protobuf 解碼的逐步流程。

在本節中：

- 先決條件
- 建立描述項檔案
- 將描述項檔案上傳至 S3 儲存貯體
- 在規則中設定 protobuf 解碼
- 限制
- 最佳實務

先決條件

- 協定緩衝區 (protobuf) 的基本知識
- 該.proto 檔案定義了訊息類型及相關的相依性
- 在您的系統上安裝 Protobuf 編譯器 (protoc)

建立描述項檔案

如果您已有描述項檔案，則可以略過此步驟。描述項檔案 (.desc) 是 .proto 檔案的編譯版本，屬於文字檔案，用於定義在 protobuf 序列化中使用的資料結構和訊息類型。要產生描述項檔案，您必須定義 .proto 檔案並使用 protoc 編譯器對其進行編譯。

1. 建立用於定義訊息類型的 .proto 檔案。範例 .proto 檔案看起來可能與以下內容相似：

```
syntax = "proto3";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

在此範例 .proto 檔案中，您使用 proto3 語法並定義訊息類型 Person。Person 訊息定義會指定三個欄位 (名稱、ID 和電子郵件)。如需有關 .proto 檔案訊息格式的詳細資訊，請參閱 語言指南 (proto3)。

2. 使用 protoc 編譯器編譯 .proto 檔案並產生描述項檔案。用於建立描述項 (.desc) 檔案的範例命令如下所示：

```
protoc --descriptor_set_out=<FILENAME>.desc \
    --proto_path=<PATH_TO_IMPORTS_DIRECTORY> \
    --include_imports \
    <PROTO_FILENAME>.proto
```

此範例命令會產生描述項檔案 <FILENAME>.desc， AWS IoT Core 規則可用來解碼符合 中定義之資料結構的 protobuf 承載<PROTO_FILENAME>.proto。

- --descriptor_set_out

  指定所要產生的描述項檔案 (<FILENAME>.desc) 名稱。

- `--proto_path`

  指定編譯檔案所參考的任何已匯入 `.proto` 檔案的位置。如果您有多項已匯入的 `.proto` 檔案位於不同位置，則可以多次指定旗標。

- `--include_imports`

  指定任何已匯入的 `.proto` 檔案也應加以編譯，並納入 `<FILENAME>.desc` 描述項檔案中。

- `<PROTO_FILENAME>.proto`

  指定要編譯的 `.proto` 檔案名稱。

如需 protoc 參考的詳細資訊，請參閱 API 參考。

將描述項檔案上傳至 S3 儲存貯體

建立描述項檔案 之後`<FILENAME>.desc`，請使用 AWS API、 AWS SDK 或 將描述項檔案上傳至 Amazon S3 `<FILENAME>.desc`儲存貯體 AWS 管理主控台。

重要考量

- 請確定您上傳描述項檔案到 中的 Amazon S3 儲存貯 AWS 帳戶 體，與您打算設定規則 AWS 區域的位置相同。
- 請務必授予`FileDescriptorSet`從 S3 讀取 的 AWS IoT Core 存取權。如果 S3 儲存貯體已停用伺服器端的加密 (SSE)，或者 S3 儲存貯體是使用 Amazon S3 受管金鑰 (SSE-S3) 進行加密，則不需要其他政策組態。下列範例儲存貯體政策可實現此目的：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "s3:Get*",
                     "Resource": "arn:aws:s3:::<BUCKET NAME>/<FILENAME>.desc"
  }
 ]
```

```
}
```

- 如果您的 S3 儲存貯體使用 AWS Key Management Service 金鑰 (SSE-KMS) 加密，請務必在存取 S3 儲存貯體時授予使用金鑰的 AWS IoT Core 許可。作法為您可以將下列陳述式新增至金鑰政策：

```
{
 "Sid": "Statement1",
 "Effect": "Allow",
 "Principal": {
  "Service": "iot.amazonaws.com"
 },
 "Action": [
  "kms:Decrypt",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
 ],
        "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

}
```

在規則中設定 protobuf 解碼

將描述項檔案上傳到 Amazon S3 儲存貯體後，請設定一項規則以供使用 decode(value, decodingScheme) SQL 函數來解碼 protobuf 訊息承載格式。詳細的函數簽章和範例可參見 AWS IoT SQL 參考的 decode(value, decodingScheme) SQL 函數。

以下是使用 decode(value, decodingScheme) 函數的 SQL 表達式範例：

```
SELECT VALUE decode(*, 'proto', '<BUCKET NAME>', '<FILENAME>.desc', '<PROTO_FILENAME>',
 '<PROTO_MESSAGE_TYPE>') FROM '<MY_TOPIC>'
```

在此範例表達式中：

- 您可以使用 decode(value, decodingScheme) SQL 函數，來解碼 * 參考的二進位訊息承載。這可以是二進位 protobuf 編碼承載，也可以是代表 base64 編碼 protobuf 承載的 JSON 字串。
- 所提供的訊息承載會使用 PROTO_FILENAME.proto 中定義的Person 訊息類型進行編碼。
- 名為 BUCKET NAME 的 Amazon S3 儲存貯體含有從 PROTO_FILENAME.proto 產生的 FILENAME.desc。

完成組態後，請在規則訂閱的主題 AWS IoT Core 上發佈訊息至 。

限制

AWS IoT Core 規則支援具有下列限制的 protobuf：

- 不支援在替代範本中解碼 protobuf 訊息承載。
- 在解碼 protobuf 訊息承載時，您最多可以在單一 SQL 表達式中使用解碼 SQL 函數 2 次。
- 傳入承載大小上限為 128 KiB (1KiB = 1024 位元組)，傳出承載大小上限為 128 KiB，而儲存在 Amazon S3 儲存貯體中的 `FileDescriptorSet` 物件大小上限為 32 KiB。
- 不支援使用 SSE-C 加密功能對 Amazon S3 儲存貯體進行加密。

最佳實務

以下是一些最佳實務和疑難排解提示。

- 在 Amazon S3 儲存貯體中備份原型檔案。

  理想的做法是備份 proto 檔案以防患未然。例如，如果在執行 protoc 時錯誤地修改了沒有備份的原型檔案，這可能會導致生產堆疊發生問題。有多種方法在 Amazon S3 儲存貯體中備份檔案 例如，您可以在 S3 儲存貯體中使用版本控制。如需有關如何備份 Amazon S3 儲存貯體中檔案的詳細資訊，請參閱 Amazon S3 開發人員指南。
- 設定 AWS IoT 記錄以檢視日誌項目。

  設定 AWS IoT 記錄是很好的做法，讓您可以在 CloudWatch 中檢查帳戶的 AWS IoT 日誌。當規則的 SQL 查詢呼叫外部 函數時， AWS IoT Core Rules 會產生具有 之 `eventType`的日誌項目`FunctionExecution`，其中包含可協助您對失敗進行疑難排解的原因欄位。可能的錯誤包括找不到 Amazon S3 物件，或是 protobuf 檔案描述項無效。如需進一步了解如何設定 AWS IoT 記錄及查看日誌項目，請參閱設定 AWS IoT 記錄和規則引擎日誌項目。
- 使用新的物件索引鍵來更新 `FileDescriptorSet` 以及更新規則中的物件索引鍵。

  您可以將更新後的描述項檔案上傳到 Amazon S3 儲存貯體來更新 `FileDescriptorSet`。系統可能需要最多 15 分鐘的時間來反映 `FileDescriptorSet` 的更新作業。為了避免這種延遲，理想做法是使用新的物件索引鍵上傳更新後的 `FileDescriptorSet`，並在規則中更新物件索引鍵。

SQL 版本

AWS IoT 規則引擎使用類似 SQL 的語法，從 MQTT 訊息中選取資料。SQL 陳述式會以指定的 SQL 版本來解釋，版本由描述規則的 JSON 文件的 `awsIotSqlVersion` 屬性指定。如需更多關於 JSON 規則文件結構的資訊，請參閱建立規則的相關文章。`awsIotSqlVersion` 屬性可讓您指定要使用的 AWS IoT SQL 規則引擎版本。部署新版本時，可以繼續使用較早版本，或變更規則以使用新版本。目前的規則會繼續使用建立時使用的版本。

下列 JSON 範例顯示了如何使用 `awsIotSqlVersion` 屬性來指定 SQL 版本。

```
{
    "sql": "expression",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "republish": {
            "topic": "my-mqtt-topic",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

AWS IoT 目前支援下列 SQL 版本：

- `2016-03-23`：建立於 2016 年 3 月 23 日的 SQL 版本（推薦）。
- `2015-10-08`：建立於 2015 年 10 月 8 日的原始 SQL 版本。
- `beta`：最新的 SQL 版本。此版本可能導致規則受到中段變更。

2016 年 3 月 23 日 SQL 規則引擎版本的新功能

- 修正巢狀 JSON 物件的選取問題。
- 修正陣列查詢的問題。
- 支援物件內部查詢。如需詳細資訊，請參閱巢狀物件查詢。
- 支援將陣列輸出為最上層物件。
- 除了 `encode(value, encodingScheme)` 函數外，它也可以應用於 JSON 和非 JSON 格式資料。如需詳細資訊，請參閱編碼函數。

輸出一個 **Array** 作為最上層物件

此功能可讓規則將陣列作為最上層物件傳回。舉例而言，若為以下的 MQTT 訊息：

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

此外以下規則：

```
SELECT VALUE arr FROM 'topic'
```

該規則會產生以下輸出。

```
[1,2,3,4]
```

AWS IoT 規則動作中的預留字元序列

AWS IoT 規則會保留^{字元序列。您無法在規則動作組態欄位中使用它。如果組態欄位包含此序列，CreateTopicRule和 ReplaceTopicRule操作會失敗並顯示錯誤。移除或變更序列，然後重新提交請求。

AWS IoT Device Shadow 服務

AWS IoT Device Shadow 服務會將陰影新增至 AWS IoT 物件。影子可讓應用程式和其他服務使用裝置的狀態，無論裝置是否連接到 AWS IoT 。 AWS IoT 物件可以有多個具名影子，讓您的 IoT 解決方案有更多選項可將您的裝置連接到其他應用程式和服務。

AWS IoT 物件在明確建立之前沒有任何陰影。您可以使用 AWS IoT 主控台建立、更新和刪除陰影。裝置、其他 Web 用戶端和服務可使用 MQTT 及預留的 MQTT 主題、使用 Device Shadow REST API 的 HTTP，以及 AWS IoT的AWS CLI。由於陰影由 存放在 AWS 雲端，因此無論裝置是否已連線，它們都可以從應用程式和其他雲端服務收集和報告裝置狀態資料。

使用影子

影子會提供可靠的資料存放區，讓裝置、應用程式和其他雲端服務共用資料。它們可讓裝置、應用程式和其他雲端服務連線和中斷連線，而不會遺失裝置的狀態。

當裝置、應用程式和其他雲端服務連線時 AWS IoT，他們可以透過其影子來存取和控制裝置的目前狀態。例如，應用程式可以透過更新影子來請求裝置狀態的變更。 會 AWS IoT 發佈訊息，指出裝置變更。裝置會接收此訊息、更新其狀態以符合，並發佈具有更新狀態的訊息。Device Shadow 服務會在對應的影子中反映此更新的狀態。應用程式可以訂閱影子的更新，也可以查詢影子的目前狀態。

當裝置離線時，應用程式仍可與 AWS IoT 和裝置的陰影通訊。當裝置重新連線時，它會收到其影子的目前狀態，以便更新其狀態以符合其影子的狀態，然後發佈具有更新狀態的訊息。同樣地，當應用程式離線且裝置狀態在離線時變更時，裝置會保持影子更新，以便在重新連線時，應用程式可以查詢其目前狀態的影子。

如果您的裝置頻繁離線，而您希望將裝置設定為在重新連線後接收增量訊息，則可以使用持續工作階段功能。如需持續工作階段有效期限的詳細資訊，請參閱持續工作階段有效期限。

選擇使用已命名或未命名的影子

Device Shadow 服務支援已命名、未命名或傳統的影子。一個物件可以有多個已命名的影子，並且不超過一個未命名的影子。物件也可以具有預留已命名影子，其運作方式與已命名影子類似，只是您無法更新其名稱。如需詳細資訊，請參閱預留已命名影子。

一個物件物件可以同時具有已命名和未命名的影子；但是用於存取每個影子的 API 略有不同，所以決定最適合您解決方案的影子類型並僅使用該類型可能會更有效率。如需存取影子 API 的詳細資訊，請參閱 影子主題。

使用已命名的影子，您可以建立物件物件狀態的不同檢視。例如，您可以將具有許多屬性的物件物件分割成具有邏輯屬性群組的影子，每個屬性都用其影子名稱加以識別。您也可以透過將屬性分組成不同的影子，並使用原則來控制存取來限制屬性的存取權限。如需與裝置影子搭配使用之政策的詳細資訊，請參閱適用於 AWS IoT的動作、資源及條件索引鍵及 AWS IoT Core 政策。

典型、未命名的影子比已命名的影子更簡單，但限制比已命名的影子多。每個 AWS IoT 物件只能有一個未命名的影子。如果您預期 IoT 解決方案對影子資料的需求有限，這可能就是您要開始使用影子的契機。但是，如果您認為未來可能想要增加其他影子，請考慮從一開始就使用已命名的影子。

機群索引以不同方式支援未命名的影子和已命名的影子。如需詳細資訊，請參閱管理機群索引。

存取影子

每個影子都有預留的 MQTT 主題和 HTTP URL，其支援在影子上的 `get`、`update` 和 `delete` 動作。

影子會使用 JSON 影子文件 來儲存和擷取資料。影子的文件包含狀態屬性，描述裝置狀態的下列層面：

- `desired`

  應用程式會透過更新 `desired` 物件來指定裝置屬性的所需狀態。

- `reported`

  裝置會報告 `reported` 物件中的目前狀態。

- `delta`

  AWS IoT 會報告 `delta` 物件中所需狀態與報告狀態之間的差異。

儲存在影子中的資料會由更新動作訊息內文的狀態屬性決定。後續的更新動作可以修改現有資料物件的值，也可以在影子的狀態物件中新增和刪除金鑰和其他元素。如需存取影子的詳細資訊，請參閱 在裝置中使用影子 和 在應用程式和服務中使用影子。

> **Important**
>
> 提出更新要求的權限應限於受信任的應用程式和裝置。這可以防止影子的狀態屬性遭意外地變更；否則，使用影子的裝置和應用程式的設計應預期狀態屬性中的金鑰會變更。

在裝置、應用程式和其他雲端服務中使用影子

在裝置、應用程式和其他雲端服務中使用影子需要一致性和協調性。 AWS IoT Device Shadow 服務會儲存陰影狀態、在陰影狀態變更時傳送訊息，以及回應變更其狀態的訊息。IoT 解決方案中的裝置、應用程式和其他雲端服務必須管理其狀態，並使其與 Device Shadow 狀態保持一致。

影子狀態資料是動態的，可由裝置、應用程式和其他具有存取影子權限的雲端服務進行變更。基於這個原因，請務必考慮每個裝置、應用程式和其他雲端服務如何與影子互動。例如：

- 將狀態資料傳到影子時，設備應該只會寫入影子狀態的 `reported` 屬性。
- 透過影子將狀態變更要求傳給裝置時，應用程式和其他雲端服務應該只會寫入 `desired` 屬性。

> **Important**
>
> 包含在影子資料物件中的資料是獨立於其他影子和其他物件物件屬性的資料，例如物件的屬性和物件物件的裝置可能發佈的 MQTT 訊息內容。但如有必要，裝置可以在不同的 MQTT 主題和影子中報告相同的資料。
> 支援多重影子的裝置必須維持它在不同影子中報告之資料的一致性。

訊息順序

我們無法保證服務的訊息 AWS IoT 會以任何特定順序送達裝置。下列情境說明在此情況下會發生的情況。

初始狀態文件：

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

更新 1：

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

更新 2：

```
{
  "state": {
    "desired": {
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

最終狀態文件：

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

這樣會產生兩個差量訊息：

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
```

```
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
  "timestamp": 123456779
}
```

裝置可能不會依此順序收到這些訊息。由於這些訊息中的狀態是累積的，因此裝置可以安全捨棄比追蹤中版本編號更舊的所有訊息。如果裝置在接收到版本 11 的差量之前收到了版本 12 的差量，則可安全捨棄版本 11 的訊息。

裁剪影子訊息

若要減少傳送至您裝置的影子訊息大小，請定義只選擇了裝置所需之欄位的規則，然後將訊息重新發佈至裝置目前監聽的 MQTT 主題。

規則是在 JSON 中指定，並應如下所示：

```
{
  "sql": "SELECT state, version FROM '$aws/things/+/shadow/update/delta'",
  "ruleDisabled": false,
  "actions": [
    {
      "republish": {
        "topic": "${topic(3)}/delta",
        "roleArn": "arn:aws:iam:123456789012:role/my-iot-role"
      }
    }
  ]
}
```

SELECT 陳述會決定將訊息的哪些欄位重新發佈至指定主題。使用 "+" 萬用字元表示所有影子名稱。規則會指定所有符合的訊息都應重新發佈至指定主題。在這種情況下，該 "topic()" 函數是用於指定要重新發佈的主題。topic(3) 會評估原始主題中的物件名稱。如需關於建立規則的詳細資訊，請參閱 [的規則 AWS IoT]。

在裝置中使用影子

本節說明使用 MQTT 訊息與影子進行裝置通訊，這是裝置與 AWS IoT Device Shadow 服務通訊的偏好方法。

影子通訊功能會模擬使用 MQTT 的發佈/訂閱通訊模型的請求/回應模型。每個影子動作包括一個請求主題，一個成功的回應主題 (`accepted`) 和一個錯誤響應主題 (`rejected`)。

如果您希望應用程式和服務能夠判斷裝置是否已連線，請參閱 偵測裝置是否已連線。

> **Important**
>
> 因 MQTT 會使用發佈/訂閱通訊模型，因此您應先訂閱回應主題，再發佈請求主題。若您並未這麼做，則將不會收到發佈請求的回應。
> 若您使用 AWS IoT Device SDK 呼叫 Device Shadow 服務 API，則會為您處理。

本節範例使用主題的縮寫形式，其中 *`ShadowTopicPrefix`* 可參照已命名或未命名的影子，如本表格所述。

影子可以為已命名或未命名 (典型)。各影子所使用的主題只有在主題字首中有所不同。此表格會顯示每種影子類型所使用的主題字首。

`ShadowTopicPrefix` 值	影子類型
`$aws/things/` *`thingName`* `/shadow`	未命名 (經典) 影子
`$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`*	已命名影子

> **Important**
>
> 確保您的應用程式或服務對影子的使用符合一致性，並且受您裝置中對應的實作支援。例如，請考慮如何建立、更新和刪除影子。也請考慮如何在裝置中處理更新，以及透過影子存取裝置的應用程式或服務。您的設計應該清楚了解裝置狀態如何更新和報告，以及您的應用程式和服務如何與裝置及其影子互動。

若要建立完整的主題，請在 *ShadowTopicPrefix* 選取您要參照的影子類型、用對應值取代 *thingName* 及 *shadowName* (如適用) 的影子類型，然後將該類型附加至主題 stub，如下表所示。請記住，主題會區分大小寫。

如需有關影子保留主題的詳細資訊，請參閱 影子主題。

在第一次連線至 時初始化裝置 AWS IoT

裝置向 註冊後 AWS IoT，應該訂閱這些 MQTT 訊息以取得其支援的影子。

主題	意義	收到此主題時裝置應採取的動作
ShadowTopicPrefix / delete/accepted	已接受delete請求並 AWS IoT 刪除影子。	適應已刪除影子所需的動作，例如停止發佈更新。
ShadowTopicPrefix / delete/rejected	delete 請求遭到 拒絕， AWS IoT 且影子未遭到刪除。訊息內文包含錯誤資訊。	在訊息內文中回應錯誤訊息。
ShadowTopicPrefix / get/accepted	get 請求已由 接受 AWS IoT，且訊息內文包含目前的影子文件。	處理訊息內文中的狀態文件所需的動作。
ShadowTopicPrefix / get/rejected	get 請求遭到 拒絕 AWS IoT，且訊息內文包含錯誤資訊。	在訊息內文中回應錯誤訊息。
ShadowTopicPrefix / update/accepted	update 請求已由 接受 AWS IoT，且訊息內文包含目前的影子文件。	確認訊息內文中的更新資料符合裝置狀態。
ShadowTopicPrefix / update/rejected	update 請求遭到 拒絕 AWS IoT，且訊息內文包含錯誤資訊。	在訊息內文中回應錯誤訊息。
ShadowTopicPrefix / update/delta	影子文件已由請求更新 AWS IoT，且訊息內文包含請求的變更。	更新裝置的狀態，以符合訊息內文中所需的狀態。

主題	意義	收到此主題時裝置應採取的動作
ShadowTopicPrefix /update/documents	最近已完成影子的更新，而訊息內文包含目前的影子文件。	確認訊息內文中的更新狀態符合裝置的狀態。

訂閱上表中每個影子的訊息之後，裝置應該測試，看看它支援的影子是否已經透過將 /get 主題發佈到每個影子來建立。如果收到 /get/accepted 訊息，訊息內文會包含影子文件，裝置可用來初始化其狀態。如果收到 /get/rejected 訊息，應該透過發佈具有目前裝置狀態的 /update 訊息來建立影子。

例如，假設您有一個沒有任何經典或已命名影子的物件 My_IoT_Thing。如果您現在於保留主題 $aws/things/My_IoT_Thing/shadow/get 上發佈 /get 請求，則會在 $aws/things/My_IoT_Thing/shadow/get/rejected 主題上傳回錯誤，因為物件沒有任何影子。若要解決此錯誤，請先使用 $aws/things/My_IoT_Thing/shadow/update 主題發佈 /update 訊息與目前裝置狀態 (如以下承載)。

```
{
 "state": {
  "reported": {
   "welcome": "aws-iot",
   "color": "yellow"
  }
 }
}
```

現在會為物件建立一個經典影子，並將訊息發佈至 $aws/things/My_IoT_Thing/shadow/update/accepted 主題。如果發佈至主題 $aws/things/My_IoT_Thing/shadow/get，則會將回應與裝置狀態傳回給 $aws/things/My_IoT_Thing/shadow/get/accepted 主題。

對於已命名的影子，在使用 get 請求之前，必須先建立已命名的影子或發佈含有影子名稱的更新。例如，若要建立已命名的影子 namedShadow1，請先將裝置狀態資訊發佈至主題 $aws/things/My_IoT_Thing/shadow/name/namedShadow1/update。若要擷取狀態資訊，請對已命名的影子 $aws/things/My_IoT_Thing/shadow/name/namedShadow1/get 使用 /get 請求。

在裝置連線到 時處理訊息 AWS IoT

當裝置連線到 時 AWS IoT，它可以接收 /update/delta 訊息，並應透過下列方式讓裝置狀態與其影子中的變更相符：

1. 讀取接收到的所有 /update/delta 訊息，並同步處理裝置狀態以進行比對。
2. 只要裝置狀態發生變化時，便使用具有裝置目前狀態的 `reported` 訊息內文發佈 /update 訊息。

當裝置連線時，它應該在指示時發佈這些訊息。

指示	主題	承載
裝置的狀態已變更。	*ShadowTopicPrefix* /update	具有 `reported` 屬性的影子文件。
裝置可能不會與影子同步。	*ShadowTopicPrefix* /get	(空白)
裝置上的動作表示裝置將不再支援影子，例如移除或取代裝置時。	*ShadowTopicPrefix* /delete	(空白)

當裝置重新連線至 時處理訊息 AWS IoT

當具有一或多個影子的裝置連線到 時 AWS IoT，應該透過下列方式將其狀態與其支援的所有影子狀態同步：

1. 讀取接收到的所有 /update/delta 訊息，並同步處理裝置狀態以進行比對。
2. 使用具有裝置目前狀態的 `reported` 訊息內文發佈 /update 訊息。

在應用程式和服務中使用影子

本節說明應用程式或服務如何與 AWS IoT Device Shadow 服務互動。此範例假設應用程式或服務只透過影子與影子與該裝置互動。此範例不包含任何管理動作，例如建立或刪除影子。

此範例使用 AWS IoT Device Shadow 服務的 REST API 與陰影互動。與用於 在裝置中使用影子 中的範例 (使用發佈/訂閱通訊模型) 不同，此範例使用 REST API 的請求/回應通訊模型。這表示應用程式或服務必須先提出請求，才能接收來自 的回應 AWS IoT。不過，此模型的缺點是它不支援通知。如果您的應用程式或服務需要及時通知裝置狀態變更，請考慮透過 WSS 通訊協定支援發佈/訂閱通訊模型的 MQTT 或 MQTT，如中所述 在裝置中使用影子。

> **Important**
>
> 請確定您的應用程式或服務對影子的使用與裝置中的對應實作一致，並支援這些功能。例如，請考慮如何建立、更新和刪除影子，以及如何在裝置和存取影子的應用程式或服務中處理更新。您的設計應該清楚指定裝置狀態的更新和報告方式，以及您的應用程式和服務如何與裝置及其影子互動。

REST API 的已命名影子 URL 是：

```
https://endpoint/things/thingName/shadow?name=shadowName
```

和一個未命名的影子：

```
https://endpoint/things/thingName/shadow
```

其中：

端點

CLI 命令傳回的端點：

```
aws iot describe-endpoint --endpoint-type IOT:Data-ATS
```

thingName

影子所屬物件物件的名稱

shadowName

已命名影子的名稱。此參數不會與未命名影子搭配使用。

在連線至 時初始化應用程式或服務 AWS IoT

當應用程式第一次連線到 時 AWS IoT，應傳送 HTTP GET 請求至其使用的影子 URLs，以取得其使用的影子目前狀態。這可讓它將應用程式或服務同步到影子。

應用程式或服務連線到 時，處理狀態會變更 AWS IoT

當應用程式或服務連線到 時 AWS IoT，它可以在其使用的影子 URLs上傳送 HTTP GET 請求，以定期查詢目前狀態。

當使用者與應用程式或服務互動以變更裝置狀態時，應用程式或服務可以將 HTTP POST 請求傳送至用來更新 `desired` 影子狀態的影子 URL。此請求會傳回已接受的變更，但是您可能必須透過提出 HTTP GET 請求來輪詢影子，直到裝置已更新其新狀態的影子。

偵測裝置是否已連線

若要判斷裝置目前是否已連線，請在影子文件中包含 `connected` 屬性，並使用 MQTT Last Will 和 Testament (LWT) 訊息以將 `connected` 屬性設為 `false` (若裝置因錯誤而中斷連線)。

> **Note**
>
> AWS IoT Device Shadow 服務會忽略傳送至 AWS IoT 預留主題 （ 開頭為 $ 的主題 ） 的 MQTT LWT 訊息。不過，它們是由訂閱的用戶端和 AWS IoT 規則引擎處理，因此您需要建立傳送至非預留主題的 LWT 訊息，以及將 MQTT LWT 訊息作為影子更新訊息重新發佈至影子的預留更新主題 的規則*ShadowTopicPrefix*/update。

傳送 LWT 訊息 Device Shadow 服務

1. 建立重新發佈保留主題上的 MQTT LWT 訊息的規則。下列範例規則會接聽有關 `my/things/myLightBulb/update` 主題的訊息，並將其重新發佈至 `$aws/things/myLightBulb/shadow/update`。

```
{
    "rule": {
    "ruleDisabled": false,
    "sql": "SELECT * FROM 'my/things/myLightBulb/update'",
    "description": "Turn my/things/ into $aws/things/",
    "actions": [
        {
        "republish": {
            "topic": "$$aws/things/myLightBulb/shadow/update",
            "roleArn": "arn:aws:iam:123456789012:role/aws_iot_republish"
            }
        }
```

```
     ]
   }
}
```

2. 當裝置連線到 時 AWS IoT，它會將 LWT 訊息註冊到未保留的主題，以便重新發佈規則進行辨識。在此範例中，該主題是 my/things/myLightBulb/update，它會將連線的內容設定為 false。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. 連線後，裝置會在其影子更新主題上發佈訊息 $aws/things/myLightBulb/shadow/update，並報告其目前狀態，其中包括將其 connected 屬性設定為 true。

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
    }
}
```

4. 在裝置正常中斷連線之前，它會發佈其影子更新主題的訊息 $aws/things/myLightBulb/shadow/update，以報告其最新狀態，其中包括將其 connected 屬性設定為 false。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. 如果裝置因錯誤而中斷連線， AWS IoT 訊息中介裝置會代表裝置發佈裝置的 LWT 訊息。重新發佈規則會偵測到此訊息，並發佈影子更新訊息，以更新 Device Shadow 的 connected 屬性。

> **Note**
>
> 由於中斷連線處理的非同步性質，LWT 訊息不保證在重新連線期間會依序分派。我們建議您使用生命週期事件來改善連線狀態偵測的準確性，因為這些事件提供屬性來管理out-of-order的事件。

模擬 Device Shadow 服務通訊

本主題示範如何將 Device Shadow 服務如何作為中繼點，並允許裝置和應用程式使用影子來更新、儲存和擷取裝置的狀態。

若要示範本主題中所述的互動，並進一步探索，您將需要 AWS 帳戶 和可在其中執行 的系統 AWS CLI。如果您沒有這些項目，您仍然可以在程式碼範例中看到互動。

在此範例中， AWS IoT 主控台代表裝置。 AWS CLI 代表透過影子存取裝置的應用程式或服務。介面與應用程式可能用來通訊的 API AWS CLI 非常相似 AWS IoT。此範例中的裝置是智慧型燈泡，應用程式會顯示燈泡的狀態，而且可以變更燈泡的狀態。

準備模擬

以下程序會開啟模擬裝置的 AWS IoT 主控台，以及模擬應用程式的命令列視窗來初始化模擬。

準備模擬環境的步驟

1. 您需要 AWS 帳戶 ，才能自行執行本主題的範例。如果您沒有 AWS 帳戶，請建立一個，如 中所述設定 AWS 帳戶。
2. 開啟 AWS IoT 主控台，然後在左側選單中選擇 Test (測試) 以開啟 MQTT client (MQTT 用戶端)。
3. 在另一個視窗中，在已安裝 AWS CLI 的系統上開啟終端機視窗。

您應該開啟兩個視窗：一個在測試頁面上使用 AWS IoT 主控台，另一個使用命令列提示字元。

初始化裝置

在這個模擬中，我們將使用一個名為 mySimulatedThing 的物件物件，以及其名為 simShadow1 的影子。

建立物件及其 IoT 政策

如要建立物件物件，請於 AWS IoT 主控台中：

1. 選擇 Manage (管理)，然後選擇 Things (物件)。
2. 如果列出實物，請按一下建立按鈕，否則請按一下註冊單一實物來建立單一 AWS IoT 實物。
3. 輸入名稱 `mySimulatedThing`，將其他設定保留為預設值，然後按一下Next (下一步)。
4. 使用單鍵憑證建立來產生憑證，其會驗證裝置到 AWS IoT的連線。按一下 Activate (啟用)，啟用憑證。
5. 您可連接政策 `My_IoT_Policy`，其將授與裝置發佈和訂閱 MQTT 預留主題的許可。如需如何建立 AWS IoT 物件以及如何建立此政策的詳細步驟，請參閱 建立物件。

為物件建立已命名的影子

您可以向主題 `$aws/things/mySimulatedThing/shadow/name/simShadow1/update` 發佈更新請求來為物件建立已命名的影子，如下方所述。

或者，若要建立具名影子：

1. 在 AWS IoT Console (主控台) 中，在顯示的物件清單中選擇您的物件，然後選擇 Shadows (影子)。
2. 選擇 Add a shadow (新增影子)，輸入名稱 `simShadow1`，接著選擇 Create (建立)，以新增命名的影子。

訂閱並發佈至預留 MQTT 主題

於主控台中，訂閱預留 MQTT 影子主題。這些主題是對 `get`、`update` 和 `delete` 動作的回應，以便您的裝置在發佈動作之後就可以接收回應。

訂閱 MQTT 用戶端中的 MQTT 主題

1. 於 MQTT client (MQTT 用戶端) 中，選擇 Subscribe to a topic (訂閱主題)。
2. 輸入 `get`、`update` 和 `delete` 主題以進行訂閱。從下方清單一次複製一個主題，將其貼到Topic filter (主題篩選條件) 欄位，再按一下 Subscribe (訂閱)。您應會看到顯示於 Subscriptions (訂閱) 之下的主題。

   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted`

- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/documents

此時，您的模擬裝置已準備好接收由 AWS IoT發佈的主題。

如要訂閱 MQTT 用戶端中的 MQTT 主題

在裝置自行初始化並訂閱回應主題之後，它會查詢所支援的影子。這個模擬支援只有一個影子，該影子命名為 simShadow1，支援命名為 mySimulatedThing 的物件物件。

從 MQTT 用戶端取得目前影子狀態

1. 於 MQTT 用戶端 中，選擇 Publish to a topic (發佈至主題)。
2. 在 Publish (發佈) 之下，輸入下列主題，然後刪除下方主題訊息內文視窗的任何內容，您可在其中輸入要取得的主題。接著，您可選擇 Publish to topic (發佈至主題)，發佈請求。$aws/things/mySimulatedThing/shadow/name/simShadow1/get。

   若您尚未建立命名的影子 simShadow1， 您收到了 $aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected 主題中的一則訊息，且 code 為 404，(如此範例中所示)，則系統尚未建立該影子，因此我們接下來將其建立。

```
{
  "code": 404,
  "message": "No shadow exists with name: 'simShadow1'"
}
```

建立具有裝置目前狀態的影子

1. 在 MQTT 用戶端中，選擇Publish to a topic (發佈到主題)，然後輸入此主題：

```
$aws/things/mySimulatedThing/shadow/name/simShadow1/update
```

2. 在您輸入主題的下方訊息內文視窗中，輸入此影子文件以顯示裝置正在報告其 ID 及其目前顏色的 RGB 值。選擇 Publish (發佈)，發佈請求。

```
{
```

```
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

若您在主題中收到一則訊息：

- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted：這表示已建立影子，而訊息內文包含目前的影子文件。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected：檢閱訊息內文中的錯誤。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted：影子已存在，且訊息內文具有目前的影子狀態，例如在此範例中。有了這個，您可以設置您的裝置或確認它符合影子狀態。

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
```

```
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591140517,
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

從應用程式傳送更新

本節使用 AWS CLI 來示範應用程式如何與影子互動。

使用 取得陰影的目前狀態 AWS CLI

在命令列輸入此命令：

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1 /
dev/stdout
```

在 Windows 平台上，您可使用 con，而非 /dev/stdout。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1
 con
```

因為影子已存在，而且已由裝置初始化以反映其目前狀態，所以它會傳回下列的影子文件。

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
```

```
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

應用程式可以使用此回應來初始化對其裝置狀態的表示法。

如果應用程式更新狀態，例如當使用者將智慧型燈泡的顏色變更為黃色時，應用程式會傳送 update-thing-shadow 命令。此命令會對應 UpdateThingShadow REST API。

從應用程式更新影子

在命令列輸入此命令：

AWS CLI v2.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --cli-binary-format raw-in-base64-out \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

AWS CLI v1.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

如果成功，這個命令應該會傳回下列影子文件。

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    }
  },
  "version": 4,
  "timestamp": 1591141596,
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

回應裝置中的更新

返回 AWS 主控台中的 MQTT 用戶端時，您應該會看到 AWS IoT 發佈的訊息，以反映上一節中發出的更新命令。

在 MQTT 用戶端中檢視更新訊息

於 MQTT 用戶端中，在 Subscriptions (訂閱) 欄中選擇 $aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta。如果主題名稱被截斷，您可以暫停以查看完整主題。在本主題的主題記錄中，您應會看到類似此訊息的 `/delta` 訊息。

```
{
  "version": 4,
  "timestamp": 1591141596,
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

您的裝置會處理此訊息的內容，以將裝置狀態設定為符合訊息中的 `desired` 狀態。

裝置更新狀態以符合訊息中的`desired`狀態後，必須透過 AWS IoT 發佈更新訊息將新的報告狀態傳回 。此程序會在 MQTT 用戶端中模擬這個部分。

從裝置更新影子

1. 於 MQTT 用戶端 中，選擇 Publish to a topic (發佈至主題)。
2. 在訊息內文視窗的訊息內文視窗上方的主題欄位中，輸入影子的主題，接著輸入 /update 動作：$aws/things/mySimulatedThing/shadow/name/simShadow1/update，並在訊息內文中輸入此更新的影子文件，其中說明裝置的目前狀態。選擇 Publish (發佈)，發佈更新的裝置狀態。

```
{
  "state": {
    "reported": {
      "ColorRGB": [255,255,0]
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

如果成功收到訊息 AWS IoT，您應該會在 MQTT 用戶端的 $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted 訊息日誌中看到新的回應，其中包含影子的目前狀態，例如此範例。

```
{
  "state": {
    "reported": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "reported": {
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
```

```
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

成功更新回報的裝置狀態，也會導致 AWS IoT 將訊息中影子狀態的完整描述傳送至update/documents主題，例如由裝置在上述程序中執行的影子更新所產生的此訊息內文。

```
{
  "previous": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          128,
          128,
          128
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
```

```
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          }
        ]
      }
    },
    "version": 4
  },
  "current": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          255,
          255,
          0
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
```

```
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          }
        ]
      }
    },
    "version": 5
  },
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

觀察應用程式中的更新

應用程式現在可以查詢裝置報告的目前狀態的影子。

使用 取得陰影的目前狀態 AWS CLI

1. 在命令列輸入此命令：

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 /dev/stdout
```

在 Windows 平台上，您可使用 con，而非 /dev/stdout。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 con
```

2. 因為裝置剛剛更新影子，以反映其目前的狀態，所以應該傳回下列影子文件。

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    },
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    },
```

```
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591143269
}
```

超越模擬

試驗 AWS CLI (代表應用程式) 與主控台 (代表裝置) 之間的互動，以建立 IoT 解決方案的模型。

與影子互動

本主題說明與 AWS IoT 提供使用影子的三種方法中的每一種相關聯的訊息。這些方法包括下列各項：

UPDATE

建立一個影子，如果它不存在，或者使用訊息正文中提供的狀態訊息更新現有影子的內容。 AWS IoT 會記錄每次更新的時間戳記，以指出上次更新狀態的時間。當影子的狀態變更時， AWS IoT 會傳送訊息/delta給所有 MQTT 訂閱者，並顯示 desired與 reported 狀態之間的差異。接收 /delta 訊息的裝置或應用程式會根據差異執行動作。例如，裝置可以更新其狀態至所需的狀態，或者應用程式可以更新 UI 以反映裝置狀態的改變。

GET

擷取包含影子完整狀態的目前影子文件，包括中繼資料。

DELETE

刪除裝置影子及其內容。

您無法還原已刪除的裝置影子文件，但可以使用已刪除的裝置影子文件的名稱建立新的裝置影子。如果您建立的裝置影子文件名稱與過去 48 小時內刪除的文件名稱相同，則新裝置影子文件的版本號碼會跟隨已刪除影子的版本號碼。如果裝置影子文件已刪除超過 48 小時，則使用相同名稱的新裝置影子文件的版本號碼將會是 0。

通訊協定支援

AWS IoT 透過 HTTPS 通訊協定支援 MQTT 和 REST API 來與影子互動。 為 MQTT 發佈和訂閱動作 AWS IoT 提供一組預留請求和回應主題。裝置和應用程式應先訂閱回應主題，再發佈至請求主題，以取得 AWS IoT 如何處理請求的資訊。如需詳細資訊，請參閱Device Shadow MQTT 主題及Device Shadow REST API。

請求和報告狀態

使用 AWS IoT 和影子設計 IoT 解決方案時，您應該判斷將請求變更的應用程式或裝置，以及將實作變更的應用程式或裝置。一般而言，裝置會實作並報告變更回影子，而應用程式和服務會回應並請求影子中的變更。您的解決方案可能不同，但本主題中的範例假設用戶端應用程式或服務請求變更影子中，而裝置會執行變更，並將它們回報給影子。

更新影子

您的應用程式或服務可以使用 UpdateThingShadow API 或發佈至 /update 主題來更新影子的狀態。更新只會影響請求中所指定的欄位。

當用戶端請求狀態變更時更新影子

當用戶端使用 MQTT 通訊協定請求影子中的狀態變更

1. 用戶端應具有目前的影子文件，以便識別要變更的屬性。請參閱 /get 動作，了解如何取得目前的影子文件。
2. 用戶端會訂閱下列 MQTT 主題：

   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`

- $aws/things/*thingName*/shadow/name/*shadowName*/update/delta
- $aws/things/*thingName*/shadow/name/*shadowName*/update/documents

3. 用戶端會發佈 $aws/things/*thingName*/shadow/name/*shadowName*/update 請求主題，其中包含所需的影子狀態的狀態文件。只有要變更的屬性需要包含在文件中。這是具有所需狀態的文件的範例。

```
{
  "state": {
    "desired": {
      "color": {
        "r": 10
      },
      "engine": "ON"
    }
  }
}
```

4. 如果更新請求有效，會在影子中 AWS IoT 更新所需的狀態，並發佈這些主題的訊息：

   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta

   /update/accepted 訊息包含 /accepted response state document 影子文件，而 /update/delta 訊息包含 /delta response state document 影子文件。

5. 如果更新請求無效，會使用描述錯誤的錯誤回應文件影子文件 AWS IoT 發佈具有 $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主題的訊息。

當用戶端請求在影子中使用 API 的狀態變更

1. 用戶端呼叫 UpdateThingShadow API，包含作為其訊息內文的 請求狀態文件 狀態文件。
2. 如果請求有效，會 AWS IoT 傳回 HTTP 成功回應碼和/accepted response state document影子文件作為其回應訊息內文。

   AWS IoT 也會發佈 MQTT 訊息到 $aws/things/*thingName*/shadow/name/*shadowName*/update/delta主題，其中包含訂閱該訊息的任何裝置或用戶端的/delta response state document影子文件。
3. 如果請求無效，會 AWS IoT 傳回 HTTP 錯誤回應代碼 錯誤回應文件作為其回應訊息內文。

當裝置收到有關 /update/delta 主題的 /desired 狀態時，它會在裝置中進行所需的變更。然後，它會將訊息傳送至 /update 主題，向影子報告其目前狀態。

當裝置回報其目前狀態時更新影子

當裝置使用 MQTT 通訊協定向影子報告其目前的狀態

1. 裝置應該在更新影子之前訂閱這些 MQTT 主題：
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/documents
2. 裝置會將訊息發佈至報告目前狀態的 $aws/things/*thingName*/shadow/name/*shadowName*/update 主題，例如在此範例中，以報告其目前狀態。

   ```
   {
       "state": {
           "reported" : {
               "color" : { "r" : 10 },
               "engine" : "ON"
           }
       }
   }
   ```

3. 如果 AWS IoT 接受更新，它會發佈訊息到具有/accepted response state document影子文件$aws/things/*thingName*/shadow/name/*shadowName*/update/accepted的主題。
4. 如果更新請求無效， 會使用描述錯誤的錯誤回應文件影子文件 AWS IoT 發佈具有 $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主題的訊息。

當裝置藉由使用 API 向影子報告其目前的狀態

1. 裝置會呼叫 UpdateThingShadow API，包含 請求狀態文件 狀態文件作為其訊息正文。
2. 如果請求有效， 會 AWS IoT 更新影子，並傳回 HTTP 成功回應碼，其中包含影/accepted response state document子文件作為其回應訊息內文。

AWS IoT 也會發佈 MQTT 訊息到 $aws/things/*thingName*/shadow/name/*shadowName*/update/delta主題，其中包含訂閱該訊息的任何裝置或用戶端的/delta response state document影子文件。

3. 如果請求無效， 會 AWS IoT 傳回 HTTP 錯誤回應代碼 錯誤回應文件作為其回應訊息內文。

樂觀鎖定

您可以使用狀態文件版本，確認您正在更新的裝置影子文件為最新版本。當您為更新請求提供版本，若狀態文件的目前版本與提供的版本不符，則服務會拒絕請求並顯示 HTTP 409 衝突回應代碼。衝突回應代碼也會出現在任何修改 ThingShadow 的 API 上，包括 DeleteThingShadow。

例如：

初始文件：

```
{
  "state": {
    "desired": {
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

更新：(版本不符合；系統將拒絕請求)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

結果：

```
{
  "code": 409,
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

更新：(版本符合；將接受請求)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

最終狀態：

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
}
```

擷取影子文件

您可以使用 GetThingShadow API 或訂閱並發行至 /get 主題來擷取影子文件。這會擷取完整的影子文件，包括 `desired` 和 `reported` 狀態之間的任何差異。無論裝置或用戶端正在提出請求，此工作的程序都是相同的。

使用 MQTT 通訊協定擷取影子文件

1. 裝置或用戶端應該在更新影子之前訂閱這些 MQTT 主題：
   - `$aws/things/thingName/shadow/name/shadowName/get/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/get/rejected`
2. 裝置或用戶端會將訊息發佈至具有空白訊息內文的 `$aws/things/thingName/shadow/name/shadowName/get` 主題。
3. 如果請求成功，會在訊息內文/accepted response state document中使用 AWS IoT 發佈訊息至`$aws/things/thingName/shadow/name/shadowName/get/accepted`主題。
4. 如果請求無效，會將訊息 AWS IoT 發佈至訊息內文錯誤回應文件中具有 的 `$aws/things/thingName/shadow/name/shadowName/get/rejected`主題。

使用 REST API 擷取影子文件

1. 裝置或用戶端會使用空的訊息內文呼叫 GetThingShadow API。
2. 如果請求有效，會 AWS IoT 傳回 HTTP 成功回應程式碼，並將/accepted response state document影子文件作為回應訊息內文。
3. 如果請求無效，會 AWS IoT 傳回 HTTP 錯誤回應代碼 錯誤回應文件作為其回應訊息內文。

刪除影子資料

有兩種方法可以刪除影子資料：您可以刪除影子文件中的特定屬性，也可以完全刪除影子。

- 若要從影子中刪除特定屬性，請更新影子；不過，請將您要刪除的屬性值設定為 `null`。值為 `null` 的欄位會從影子文件中移除。
- 若要刪除整個影子，請使用 DeleteThingShadow API 或發佈至 /delete 主題。

> **Note**
>
> 刪除影子不會一次將其版本編號重設為零。其將於 48 小時後重設為零。

從影子文件中刪除屬性

使用 MQTT 通訊協定從影子刪除屬性

1. 裝置或用戶端應該有目前的影子文件，以便識別要變更的屬性。如需如何取得目前影子文件的詳細資訊，請參閱 擷取影子文件。
2. 裝置或用戶端會訂閱下列 MQTT 主題：
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected
3. 裝置或用戶端會發佈 $aws/things/*thingName*/shadow/name/*shadowName*/update 請求主題，其中包含狀態文件，該文件會將 null 值指派給要刪除的影子屬性。只有要變更的屬性需要包含在文件中。這是刪除 engine 屬性的文件範例。

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   }
   ```

4. 如果更新請求有效， 會 AWS IoT 刪除影子中指定的屬性，並在訊息內文中發佈具有影/accepted response state document子文件之$aws/things/*thingName*/shadow/name/*shadowName*/update/accepted主題的訊息。
5. 如果更新請求無效， 會使用描述錯誤的錯誤回應文件影子文件 AWS IoT 發佈具有 $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主題的訊息。

使用 REST API 從影子刪除屬性

1. 裝置或用戶端會呼叫 UpdateThingShadow API，其具有將 null 值指派給要刪除之影子屬性的請求狀態文件。只包含您要在文件中刪除的屬性。這是刪除 engine 屬性的文件範例。

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   ```

```
}
```

2. 如果請求有效，會 AWS IoT 傳回 HTTP 成功回應碼和/accepted response state document影子文件作為其回應訊息內文。
3. 如果請求無效，會 AWS IoT 傳回 HTTP 錯誤回應代碼 錯誤回應文件作為其回應訊息內文。

刪除影子

以下是刪除裝置影子時的一些考量事項。

- 將裝置的影子狀態設定為 null 不會刪除影子。影子版本將在下一次更新時遞增。
- 刪除裝置的影子並不會刪除物件物件。刪除物件物件也不會刪除對應的裝置影子。
- 刪除影子不會一次將其版本編號重設為零。其將於 48 小時後重設為零。

使用 MQTT 通訊協定刪除影子

1. 裝置或用戶端會訂閱下列 MQTT 主題：
   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected
2. 裝置或用戶端會發佈具有空白訊息緩衝區的 $aws/things/*thingName*/shadow/name/*shadowName*/delete。
3. 如果刪除請求有效，會 AWS IoT 刪除影子，並在訊息內文中發佈具有$aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted主題和縮寫影/accepted response state document子文件的訊息。以下是接受的刪除訊息範例：

   ```
   {
     "version": 4,
     "timestamp": 1591057529
   }
   ```

4. 如果更新請求無效，會使用描述錯誤的錯誤回應文件影子文件 AWS IoT 發佈具有 $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected主題的訊息。

使用 REST API 刪除影子

1. 裝置或用戶端會使用空的訊息緩衝區呼叫 DeleteThingShadow API。

2. 如果請求有效，會在訊息內文中 AWS IoT 傳回 HTTP 成功回應碼和 /accepted response state document以及縮寫/accepted response state document陰影文件。以下是接受的刪除訊息範例：

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. 如果請求無效，會 AWS IoT 傳回 HTTP 錯誤回應代碼 錯誤回應文件作為其回應訊息內文。

Device Shadow REST API

影子會公開下列 URI 以更新狀態資訊：

```
https://account-specific-prefix-ats.iot.region.amazonaws.com/things/thingName/shadow
```

端點專屬於您的 AWS 帳戶。如要尋找您的端點，您可以：

- 使用來自 AWS CLI的 describe-endpoint 命令。
- 使用 AWS IoT 主控台設定。於 Settings (設定) 中，端點會列於 Custom endpoint (自訂端點) 之下
- 使用 AWS IoT 主控台物件詳細資訊頁面。在主控台中：
  1. 開啟 Manage (管理)，在 Manage (管理) 之下，選擇 Things (物件)。
  2. 於物件清單中，選擇要取得端點 URI 的物件。
  3. 選擇 Device Shadows 索引標籤，選擇您的影子。您可在 Device Shadow 詳細資訊頁面的 Device Shadow URL 區段中檢視端點 URI。

端點格式如下：

```
identifier.iot.region.amazonaws.com
```

影子 REST API 會遵循相同的 HTTPS 通訊協定/連接埠映射，如 裝置通訊協定 中所述。

> **Note**
>
> 若要使用 API，您必須使用 `iotdevicegateway` 作為身分驗證的服務名稱。如需詳細資訊，請參閱 IoTDataPlane。

API 動作

- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

您也可以透過提供 name=*shadowName* 作為 API 的查詢參數的一部分，使用 API 建立已命名的影子。

GetThingShadow

取得特定物件的影子。

回應狀態文件包括 `desired` 和 `reported` 狀態之間的差量。

請求

此請求包括標準 HTTP 標頭加上以下 URI：

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

未命名 (典型) 影子不需要 name 查詢參數。

回應

成功時，回應會包括標準 HTTP 標頭以及下列程式碼與本文：

```
HTTP 200
Response Body: response state document
```

如需詳細資訊，請參閱回應狀態文件範例。

Authorization

擷取影子需要一項允許呼叫者可執行 `iot:GetThingShadow` 動作的政策。Device Shadow 服務接受兩種形式的身分驗證：使用 IAM 憑證的 Signature 第 4 版，或使用用戶端憑證的 TLS 交互身分驗證。

下列為允許呼叫者擷取裝置影子的政策範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

UpdateThingShadow

更新指定之物件的影子。

更新僅會影響請求狀態文件中所指定的欄位。值為 null 的任何欄位，皆會從裝置影子中移除。

請求

此請求包含標準 HTTP 標頭和內文加上以下 URI 與本文：

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

未命名 (典型) 影子不需要 name 查詢參數。

如需詳細資訊，請參閱請求狀態文件範例。

回應

成功時，回應會包括標準 HTTP 標頭以及下列程式碼與本文：

```
HTTP 200
Response body: response state document
```

如需詳細資訊，請參閱回應狀態文件範例。

Authorization

更新影子需要一項允許呼叫者執行 iot:UpdateThingShadow 動作的政策。Device Shadow 服務接受兩種形式的身分驗證：使用 IAM 憑證的 Signature 第 4 版，或使用用戶端憑證的 TLS 交互身分驗證。

下列為允許呼叫者更新裝置影子的政策範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:UpdateThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

DeleteThingShadow

刪除特定物件的影子。

請求

此請求包括標準 HTTP 標頭加上以下 URI：

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

未命名 (典型) 影子不需要 name 查詢參數。

回應

成功時，回應會包括標準 HTTP 標頭以及下列程式碼與本文：

```
HTTP 200
Response body: Empty response state document
```

請注意，刪除影子並不會將其版本號碼重設為 0。

Authorization

刪除裝置影子需要一項允許呼叫者執行 `iot:DeleteThingShadow` 動作的政策。Device Shadow 服務接受兩種形式的身分驗證：使用 IAM 憑證的 Signature 第 4 版，或使用用戶端憑證的 TLS 交互身分驗證。

下列為允許呼叫者刪除裝置影子的政策範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

ListNamedShadowsForThing

列出指定物件的影子。

請求

此請求包括標準 HTTP 標頭加上以下 URI：

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

用來擷取下一組結果的符記。

此值會在分頁結果上傳回，並在傳回下一頁的呼叫中使用。

pageSize

影子名稱在每個呼叫中傳回的數目。另請參閱nextToken。

thingName

要列出命名影子的物件的名稱。

回應

成功時，回應會包括標準 HTTP 標頭以及下列回應程式碼與 影子名稱清單回應文件。

> **Note**
>
> 未命名 (傳統) 影子不會出現在此清單中。如果您只有一個經典的影子，或者您指定的 thingName 不存在，則回應將是一個空清單。

```
HTTP 200
Response body: Shadow name list document
```

Authorization

列出裝置影子需要一項允許呼叫者執行 iot:ListNamedShadowsForThing 動作的政策。Device Shadow 服務接受兩種形式的身分驗證：使用 IAM 憑證的 Signature 第 4 版，或使用用戶端憑證的 TLS 交互身分驗證。

以下為允許呼叫者更新物件已命名影子的範例政策：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:ListNamedShadowsForThing",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
```

```
    ]
}
```

Device Shadow MQTT 主題

Device Shadow 服務會使用預留的 MQTT 主題，讓應用程式與裝置可取得、更新或刪除裝置 (影子) 的狀態資訊。

在影子主題上發佈和訂閱需要主題為基礎的授權。 AWS IoT 有權新增主題到現有的主題結構。因此，我們建議您避免使用萬用字元訂閱影子主題。例如，避免像 一樣訂閱主題篩選條件，`$aws/things/thingName/shadow/#`因為當 AWS IoT 引入新的影子主題時，符合此主題篩選條件的主題數量可能會增加。關於發佈至這些主題的訊息範例，請參閱 與影子互動。

影子可以為已命名或未命名 (典型)。各影子所使用的主題只有在主題字首中有所不同。此表格會顯示每種影子類型所使用的主題字首。

ShadowTopicPrefix 值	影子類型
`$aws/things/` *`thingName`* `/shadow`	未命名 (經典) 影子
`$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`*	已命名影子

若要建立完整的主題，請在 *`ShadowTopicPrefix`* 選取您要參照的影子類型、用對應值取代 *`thingName`* 及 *`shadowName`* (如適用) 的影子類型，然後將該類型附加至主題 stub，如下節所示。

下列為用來與影子互動的 MQTT 主題。

主題

- /get
- /get/accepted
- /get/rejected
- /update
- /update/delta
- /update/accepted

- /update/documents
- /update/rejected
- /delete
- /delete/accepted
- /delete/rejected

/get

發佈空白訊息至此主題可取得裝置的影子：

```
ShadowTopicPrefix/get
```

AWS IoT 透過發佈至 /get/accepted或 來回應 /get/rejected。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get"
            ]
        }
    ]
}
```

/get/accepted

AWS IoT 傳回裝置的影子時， 會將回應影子文件發佈至此主題：

```
ShadowTopicPrefix/get/accepted
```

如需詳細資訊，請參閱回應狀態文件。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get/accepted"
            ]
        }
    ]
}
```

/get/rejected

AWS IoT 當無法傳回裝置的影子時， 會將錯誤回應文件發佈至此主題：

```
ShadowTopicPrefix/get/rejected
```

如需詳細資訊，請參閱錯誤回應文件。

範例 政策

以下為所需政策的範例：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/thingName/
shadow/get/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/shadow/get/
rejected"
      ]
    }
  ]
}
```

/update

對此主題發佈要求狀態文件，以更新裝置的影子：

```
ShadowTopicPrefix/update
```

訊息內文包含部分請求狀態文件。

嘗試更新裝置狀態的用戶端會發送一個 JSON 請求狀態文件，其 `desired` 屬性如下：

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

更新其影子的裝置將發送一個 JSON 請求狀態文件與該 `reported` 屬性，例如：

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT 透過發佈至 /update/accepted或 來回應 /update/rejected。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update"
            ]
        }
    ]
}
```

/update/delta

AWS IoT 當回應狀態文件接受裝置影子的變更，且回應狀態文件包含 desired和 reported 狀態的不同值時， 會將回應狀態文件發佈至此主題：

```
ShadowTopicPrefix/update/delta
```

訊息緩衝區包含 /delta response state document。

訊息內文詳細資料

- 發佈於 update/delta 的訊息，僅包括 desired 和 reported 部分之間不同的所需屬性。這些屬性全都會包含在其中，無論這些屬性是否原本就包含於目前的更新訊息中或是已經存放於 AWS IoT。在 desired 和 reported 部分之間沒有差異的屬性則不會包含在內。
- 如果某個屬性出現在 reported 部分，但在 desired 部分卻沒有，則此屬性不會包含在內。
- 如果某個屬性出現在 desired 部分，但在 reported 部分卻沒有，則此屬性會包含在內。
- 如果某個屬性在 reported部分被刪除，但仍存在於 desired 部分，則此屬性會包含在內。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/delta"
            ]
        }
    ]
}
```

/update/accepted

AWS IoT 接受裝置影子的變更時，會將回應狀態文件發佈至此主題：

```
ShadowTopicPrefix/update/accepted
```

訊息緩衝區包含 /accepted response state document。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
```

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/accepted"
            ]
        }
    ]
}
```

/update/documents

AWS IoT 每當成功執行影子更新時，就會發佈狀態文件至此主題：

```
ShadowTopicPrefix/update/documents
```

訊息內文包含 /documents response state document。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/documents"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/documents"
```

```
            ]
        }
    ]
}
```

/update/rejected

AWS IoT 當拒絕裝置影子的變更時， 會將錯誤回應文件發佈至此主題：

```
ShadowTopicPrefix/update/rejected
```

訊息內文包含 錯誤回應文件。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/rejected"
            ]
        }
    ]
```

```
}
```

/delete

若要刪除裝置的影子，發佈空白訊息即可刪除主題：

```
ShadowTopicPrefix/delete
```

訊息的內容會被忽略。

請注意，刪除影子並不會將其版本號碼重設為 0。

AWS IoT 透過發佈至 /delete/accepted或 來回應 /delete/rejected。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete"
            ]
        }
    ]
}
```

/delete/accepted

AWS IoT 刪除裝置的影子時， 會將訊息發佈至此主題：

```
ShadowTopicPrefix/delete/accepted
```

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/accepted"
            ]
        }
    ]
}
```

/delete/rejected

AWS IoT 當無法刪除裝置的影子時， 會將錯誤回應文件發佈至此主題：

```
ShadowTopicPrefix/delete/rejected
```

訊息內文包含 錯誤回應文件。

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/rejected"
            ]
        }
    ]
}
```

Device Shadow 服務文件

Device Shadow 服務遵守 JSON 規格的所有規則。值、物件與陣列均存放於裝置的影子文件中。

目錄

- 影子文件範例
- 文件屬性
- 差量狀態
- 版本控制影子文件
- 影子文件中的用戶端字符
- 空白影子文件屬性
- 影子文件中的陣列值

影子文件範例

在使用 REST API 或 MQTT 發佈/訂閱訊息的 UPDATE、GET、DELETE 操作中，Device Shadow 服務會使用下列文件。

範例

- 請求狀態文件
- 回應狀態文件
- 錯誤回應文件
- 影子名稱清單回應文件

請求狀態文件

請求狀態文件具有以下格式：

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state`：更新僅會影響所指定的欄位。一般而言，您會使用 `desired` 或 `reported` 屬性，但不會在同一個要求中使用兩者。
  - `desired`：請求在裝置中更新的狀態屬性和值。
  - `reported`：裝置回報的狀態屬性和值。

- `clientToken`：如已使用，您可透過用戶端字符來比對請求和相應的回應。
- `version`：若使用此項目，Device Shadow 服務只有在指定版本與其擁有的最新版本相符時，才會處理更新。

回應狀態文件

根據回應類型，回應狀態文件的格式如下。

/accepted response state document

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/delta response state document

```
{
```

```
    "state": {
        "attribute1": integer2,
        "attribute2": "string2",
        ...
        "attributeN": boolean2
    },
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/documents response state document

```
{
  "previous" : {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "metadata": {
        "desired": {
```

```
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version-1
},
"current": {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
```

```
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version
  },
  "timestamp": timestamp,
  "clientToken": "token"
}
```

回應狀態文件屬性

- previous：成功更新之後，會在更新前包含物件的 state。
- current：成功更新之後，會在更新後包含物件的 state。
- state
  - reported：僅當物件於 reported 區段回報任何資料，且包含僅位於請求狀態文件中的欄位時才出現。
  - desired：僅當裝置於 desired 區段回報任何資料，且包含僅位於請求狀態文件中的欄位時才出現。
- metadata：包含 desired 和 reported 部分中每個屬性的時間戳記，因此您可確定狀態的更新時間。

- timestamp — 產生回應的 Epoch 日期和時間 AWS IoT。
- clientToken：僅在發佈有效 JSON 至 /update 主題並使用用戶端符記時才會出現。
- version： AWS IoT中目前共用之裝置影子的文件版本。此版本會在先前的文件版本編號加一。

錯誤回應文件

錯誤回應文件的格式如下：

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- code：用於指明錯誤類型的 HTTP 回應碼。
- message：可提供額外資訊的文字訊息。
- timestamp — 產生回應的日期和時間 AWS IoT。此屬性不會出現在所有錯誤回應文件中。
- clientToken：僅於已發佈的訊息中使用用戶端字符時才會出現。

如需詳細資訊，請參閱Device Shadow 錯誤訊息。

影子名稱清單回應文件

影子名稱清單回應文件的格式如下：

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- results：影子名稱的陣列。

- `nextToken`：在分頁請求中使用的字符值，以取得序列中的下一頁。當沒有更多的影子名稱返回時，這個屬性不存在。
- `timestamp` — 產生回應的日期和時間 AWS IoT。

文件屬性

裝置的影子文件具有下列屬性：

`state`

`desired`

裝置所需的狀態。應用程式可將此部分編寫入文件，在無需直接連線的狀況下更新裝置狀態。

`reported`

裝置回報的狀態。裝置會在文件中編寫此部分以回報其新狀態。應用程式會讀取文件的這部分，以判斷裝置上次報告的狀態。

`metadata`

存放於文件 `state` 部分之資料的相關資訊。其中包括針對 `state` 部分中各屬性的時間戳記 (Epoch 時間)，可讓您確定資料是在何時更新的。

> **Note**
>
> 中繼資料不會影響服務限制或定價的文件大小。如需詳細資訊，請參閱 AWS IoT 服務限制。

`timestamp`

指出訊息的傳送者。 AWS IoT藉由使用訊息中的時間戳記和 `desired` 或 `reported` 區段中的個別屬性的時間戳記，裝置可以判斷屬性的保留天數，即使裝置沒有內部時鐘也是如此。

`clientToken`

裝置特有的字串能夠讓您在 MQTT 環境中將回應與請求建立關聯。

`version`

文件版本。每次文件更新時，版本編號便會自動增加。如此可確保更新的文件為最新版本。

如需詳細資訊，請參閱影子文件範例。

差量狀態

差量 (delta) 狀態是一種虛擬類型的狀態，包含了 desired 和 reported 之間的差異。在 desired 部分的欄位若未出現於 reported 部分，則會納入差量之中。在 reported 部分的欄位若未出現於 desired 部分，則不會納入差量之中。差量包含中繼資料，且其值與 desired 欄位裡的中繼資料相同。例如：

```
{
  "state": {
    "desired": {
      "color": "RED",
      "state": "STOP"
    },
    "reported": {
      "color": "GREEN",
      "engine": "ON"
    },
    "delta": {
      "color": "RED",
      "state": "STOP"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
      "state": {
        "timestamp": 12345
      }
      },
      "reported": {
        "color": {
          "timestamp": 12345
        },
        "engine": {
          "timestamp": 12345
        }
      },
      "delta": {
        "color": {
```

```
          "timestamp": 12345
        },
        "state": {
          "timestamp": 12345
        }
      }
    },
    "version": 17,
    "timestamp": 123456789
  }
}
```

當巢狀物件不同，差量則會包含根路徑。

```
{
  "state": {
    "desired": {
      "lights": {
        "color": {
          "r": 255,
          "g": 255,
          "b": 255
        }
      }
    },
    "reported": {
      "lights": {
        "color": {
          "r": 255,
          "g": 0,
          "b": 255
        }
      }
    },
    "delta": {
      "lights": {
        "color": {
          "g": 255
        }
      }
    }
  },
  "version": 18,
```

```
  "timestamp": 123456789
}
```

Device Shadow 服務會重複查看 desired 狀態內的所有欄位，然後將其與 reported 狀態相比較，藉此計算出差量。

陣列的處理和值相同。如果 desired 部分的某個陣列與 reported 部分的陣列不符，則整組所需陣列都會複製到差量中。

版本控制影子文件

Device Shadow 服務支援每個更新訊息的版本控制，包括請求和回應。這表示，隨著影子的每次更新，JSON 文件的版本都會遞增。如此可確保兩種情況：

- 如果用戶端嘗試以較舊的版本編號覆寫影子，就會收到錯誤訊息。通知用戶端必須先重新同步裝置的影子，才可進行更新。
- 如果此訊息的版本低於存放於用戶端的訊息版本，則用戶端可以決定不採取行動。

用戶端可以透過不在影子文件中包含版本來略過版本比對。

影子文件中的用戶端字符

您可以使用用戶端符記收發以 MQTT 為基礎的訊息，藉此驗證請求與針對請求的回應中是否包含了同樣的用戶端符記。如此可確保回應與請求之間具有關聯。

> **Note**
>
> 用戶端字符不能超過 64 位元組。長於 64 位元組的用戶端字符會導致 400 (Bad Request) 回應以及一個 Invalid clientToken 錯誤訊息。

空白影子文件屬性

當影子文件中的 reported 和 desired 屬性不適用於目前影子狀態時，它們可以是空的，您也可以省略它們。例如，影子文件只有在具有所需狀態時才會包含 desired 屬性。以下是沒有 desired 屬性的狀態文件的有效範例：

```
{
```

```
    "reported" : { "temp": 55 }
}
```

該 reported 屬性也可以是空白，例如，如果影子尚未由裝置更新：

```
{
    "desired" : { "color" : "RED" }
}
```

如果更新導致 desired 或 reported 屬性變為 null，則會從文件中移除。下面顯示了如何透過將屬性設定為 null 來刪除 desired。例如，當裝置更新其狀態時，您可能會這麼做。

```
{
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

影子文件也可以具有 desired 或 reported 屬性，使影子文件變成空白。這是一個空白但有效的影子文件範例。

```
{
}
```

影子文件中的陣列值

影子支援陣列，但會將其視為一般值，即更新某陣列會取代整組陣列。您無法只更新陣列中的某部分。

初始狀態：

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

更新：

```
{
```

```
    "desired" : { "colors" : ["RED"] }
}
```

最終狀態：

```
{
    "desired" : { "colors" : ["RED"] }
}
```

陣列不能有 Null 值。例如，以下陣列無效並且會被拒絕。

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

Device Shadow 錯誤訊息

當嘗試變更狀態文件失敗時，Device Shadow 服務會透過 MQTT 發佈訊息至錯誤主題。僅當對其中一個預留 $aws 主題的發佈請求回應時，此訊息才會進行發送。如果用戶端使用 REST API 更新文件，則會收到 HTTP 錯誤代碼作為部分回應，而 MQTT 錯誤訊息也不會發出。

HTTP 錯誤代碼	錯誤訊息
400 (錯誤的請求)	• 無效的 JSON • 缺少必要的節點：狀態 • 狀態節點必須是物件 • 所需節點必須是物件 • 報告節點必須是物件 • 版本無效 • clientToken 無效 **Note** 用戶端字符超過 64 位元組將會導致這個回應。

HTTP 錯誤代碼	錯誤訊息
	• JSON 包含太多巢狀層級；最大數量為 6 • 狀態包含無效的節點
401 (未經授權)	• 未經授權
403 (禁止)	• 禁止
404 (未找到)	• 找不到物件 • 不存在具有名稱的影子：*shadowName*
409 (衝突)	• 版本衝突
413 (承載過大)	• 承載超過允許的最大值
415 (不支援的媒體類型)	• 不支援的文件編碼；支援的編碼為 UTF-8
429 (太多請求)	• 當單一連接上的請求超過 10 則時，Device Shadow 服務就會產生此錯誤訊息。進行中的請求是已啟動但尚未完成的進行中請求。
500 (內部伺服器錯誤)	• 內部服務失敗

AWS IoT Device Management 軟體套件目錄

使用 AWS IoT Device Management 軟體套件目錄，您可以維護軟體套件及其版本的庫存。您可以將套件版本與個別物件和 AWS IoT 動態物件群組建立關聯，並透過內部程序或AWS IoT 任務進行部署。

軟體套件包含一或多個套件版本，它是可以部署為單一單元的檔案集合。套件版本可以包含韌體、作業系統更新、裝置應用程式、組態和安全性修補程式。軟體會隨著時間發展，您可以建立新套件版本並將其部署到您的機群。

AWS IoT 軟體套件中樞位於 內 AWS IoT Core。您可以使用該中樞集中註冊和維護軟體套件庫存和中繼資料，以建立軟體套件及其版本的目錄。您可以選擇根據裝置上部署的軟體套件和套件版本，對裝置進行分組。此功能可讓您將裝置端套件庫存保留為已命名影子、根據版本為裝置建立關聯和分組，並使用機群指標來視覺化跨機群的套件版本分佈。

若您已建立內部軟體部署系統，就能繼續使用該程序來部署套件版本。若您尚未建立部署程序，或若您偏好的話，建議您使用 AWS IoT 任務，以使用軟體套件目錄中的功能。如需詳細資訊，請參閱準備 AWS IoT 任務。

本章包含下列部分：

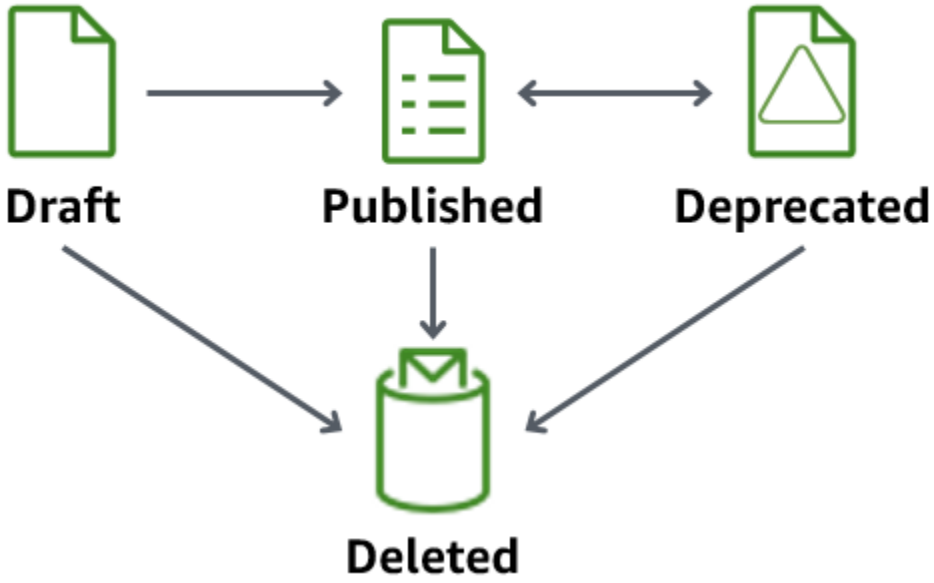
- 準備使用軟體套件目錄
- 準備安全性
- 準備機群索引
- 準備 AWS IoT 任務
- 開始使用軟體套件目錄

準備使用軟體套件目錄

下節提供套件版本生命週期的概觀，以及使用 AWS IoT Device Management 軟體套件目錄的資訊。

套件版本生命週期

套件版本可以經歷下列生命週期狀態：`draft`、`published`、和 `deprecated`。也可以為 `deleted`。



- 草稿

  當您建立套件版本時，其處於 `draft` 狀態。此狀態表示軟體套件正在準備或不完整。

  當套件版本處於此狀態時，您無法部署它。您可以編輯套件版本的描述、屬性和標籤。

  您可以使用主控台或發出 UpdatePackageVersion 或 DeletePackageVersion API 操作，將 `draft` 狀態的套件版本轉換為 `published` 或 `deleted` 狀態。

- 已發佈

  當您的套件版本準備好部署時，請將套件版本轉換為 `published` 狀態。在此狀態下，您可以透過在主控台中編輯軟體套件或透過 UpdatePackage API 操作，選擇將套件版本識別為預設版本。在此狀態下，您只能編輯描述和標籤。

  您可以使用主控台或發出 UpdatePackageVersion 或 DeletePackageVersion API 操作，將 `published` 狀態的套件版本轉換為 `deprecated` 或 `deleted` 狀態。

- 已棄用

  若有新的套件版本可用，您可以將較早的套件版本轉換為 `deprecated`。您仍然可以使用已棄用的套件版本部署任務。您也可以將已棄用的套件版本命名為預設版本，並僅編輯描述和標籤。

  考慮在版本過期`deprecated`時將套件版本轉換為 ，但您仍有使用舊版的 欄位中的裝置，或由於執行時間相依性而需要維護它。

您可以使用主控台或發出 UpdatePackageVersion 或 DeletePackageVersion API 操作，將 `deprecated` 狀態的套件版本轉換為 `published` 或 `deleted` 狀態。

- Deleted (已刪除)

  當您不需再使用套件版本，您可使用主控台或發出 DeletePackageVersion API 操作來刪除該套件版本。

  > **Note**
  >
  > 若您在有待處理任務參照套件版本時刪除該套件版本，當任務成功完成並嘗試更新預留已命名影子時，您會收到錯誤訊息。
  > 若您要刪除的軟體套件版本已指定為預設套件版本，您必須先更新該套件，將其他版本指定為預設版本，或將該欄位保留為未指定狀態。您可以透過主控台或 UpdatePackageVersion API 操作來執行這項作業。(若要刪除作為預設版本的任何指定套件版本，請在發出 UpdatePackage API 操作時將 unsetDefaultVersion 參數設定為 true)。

  若您透過主控台刪除軟體套件，則會刪除與該套件相關聯的所有套件版本，除非某個套件版本已指定為預設版本。

套件版本命名慣例

命名套件版本時，請務必規劃並套用邏輯命名策略，以便您和其他人員皆可輕易識別最新的套件版本和版本進度。建立套件版本時，您必須提供版本名稱，但策略和格式主要取決於您的商業案例。

根據最佳實務，建議您使用語意版本控制 SemVer 格式。例如，`1.2.3`，其中 1 是功能上不相容變更的主要版本，2 是功能上相容變更的主要版本，3 是修補程式版本 (用於錯誤修正)。如需詳細資訊，請參閱語意版本控制 2.0.0。如需套件版本名稱需求的詳細資訊，請參閱 AWS IoT API 參考指南中的 versionName。

預設版本

您不一定要將版本設定為預設版本。您可以新增或移除預設套件版本。您也可以部署未指定為預設版本的套件版本。

套件版本在建立時會處於 `draft` 狀態，在您將套件版本轉換為「已發佈」之前，無法將其指定為預設版本。軟體套件目錄不會自動選取某個版本作為預設版本，也不會自動將較新的套件版本更新為預設版本。您必須透過主控台或發出 UpdatePackageVersion API 操作，自行為所選套件版本命名。

版本屬性

版本屬性及其值皆包含套件版本的重要資訊。建議您定義套件或套件版本的一般用途屬性。例如，您可以為平台、架構、作業系統、發行日期、作者或 Amazon S3 URL 建立名稱/值配對。

當您使用 AWS IoT 任務文件建立任務時，您也可以選擇使用參考屬性值的替代變數 (`$parameter`)。如需詳細資訊，請參閱準備 AWS IoT 任務。

套件版本中使用的版本屬性不會自動新增至預留的具名影子，也無法直接透過機群索引編製索引或查詢。若要透過機群索引編製索引或查詢套件版本屬性，您可以在保留的具名影子中填入版本屬性。

我們建議保留具名影子擷取裝置報告屬性 中的版本屬性參數，例如操作系統和安裝時間。也可以透過機群索引對它們進行索引和查詢。

遵循特定命名慣例不需要版本屬性。您可以建立名稱/值配對，以滿足您的商業需求。套件版本上所有屬性的整體大小限制為 3 KB。如需詳細資訊，請參閱軟體套件目錄軟體套件和套件版本限制。

使用任務文件中的所有屬性

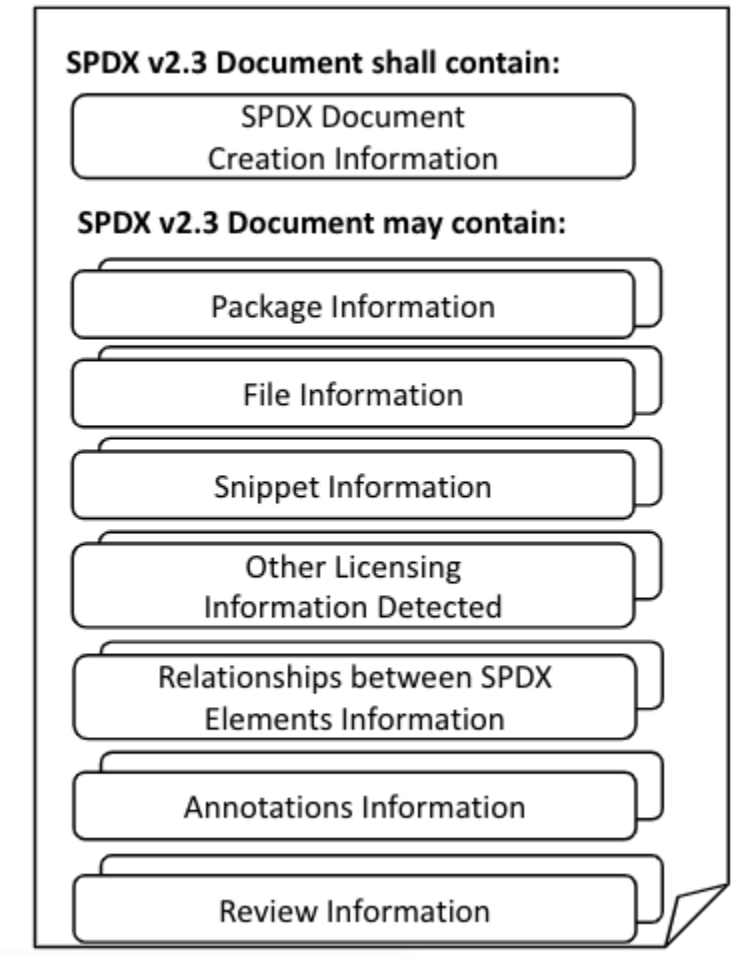
您可以讓所有套件版本屬性自動新增至所選裝置的任務部署。若要在 API 或 CLI 命令中以程式設計方式自動使用所有套件版本屬性，請參閱下列任務文件範例：

```
"TestPackage": "${aws:iot:package:TestPackage:version:PackageVersion:attributes}"
```

軟體物料清單

軟體物料清單 (SBOM) 提供軟體套件所有層面的中央儲存庫。除了存放軟體套件和套件版本之外，您還可以在軟體套件目錄中存放與每個套件版本相關聯的 AWS IoT Device Management 軟體物料清單 (SBOM)。軟體套件包含一或多個套件版本，且每個套件版本都包含一或多個元件。您可以使用軟體物料清單來描述和編目支援特定套件版本合成的每個元件。支援的 軟體物料清單的業界標準為 SPDX 和 CycloneDX。第一次建立 SBOM 時，它會根據 SPDX 和 CycloneDX 產業標準格式進行驗證。如需 SPDX 的詳細資訊，請參閱系統套件資料交換。如需 CycloneDX 的詳細資訊，請參閱 CycloneDX。

軟體物料清單說明特定套件版本元件的所有層面，例如套件資訊、檔案資訊和其他相關中繼資料。請參閱以下 SPDX 格式的軟體物料清單文件結構範例：



軟體物料清單優點

在 Software Package Catalog 中為套件版本新增軟體物料清單的主要優點之一是漏洞管理。

漏洞管理

評估和減輕軟體元件中存在明顯安全風險的漏洞，對於保護裝置機群的完整性至關重要。透過為每個套件版本新增儲存在軟體套件目錄中的軟體物料清單，您可以使用自己的內部漏洞管理解決方案，根據裝置套件版本和 SBOM 來了解哪些裝置存在風險，以主動公開安全漏洞。您可以將修正部署到受影響的裝置，並保護您的裝置機群。

軟體物料儲存帳單

每個軟體套件版本的軟體物料清單 (SBOM) 會使用 Amazon S3 版本控制功能存放在 Amazon S3 儲存貯體中。存放 SBOM 的 Amazon S3 儲存貯體必須位於建立套件版本的相同區域中。使用版本控制功能的 Amazon S3 儲存貯體會維護相同儲存貯體中物件的多個變體。如需在 Amazon S3 儲存貯體中使用版本控制的詳細資訊，請參閱在 Amazon S3 儲存貯體中使用版本控制。

> **Note**
>
> 每個軟體套件版本可以連接多個 SBOM 檔案，但 SBOM 檔案必須存放在單一 zip 封存檔案中。

儲存貯體的特定 Amazon S3 金鑰和版本 ID 是用來唯一識別套件版本之軟體物料清單的每個版本。

> **Note**
>
> 對於具有單一 SBOM 檔案的套件版本，您可以將該 SBOM 檔案存放在 Amazon S3 儲存貯體中，做為 zip 封存檔案。
> 對於具有多個 SBOM 檔案的套件版本，您必須將所有 SBOM 檔案放在單一 zip 封存檔案中，然後將該 zip 封存檔案存放在 Amazon S3 儲存貯體中。
> 在這兩種情況下，存放在單一 zip 封存檔中的所有 SBOM 檔案都會格式化為 SPDX 或 CycloneDX .json 檔案。

許可政策

為了 AWS IoT 做為指定主體來存取存放在 Amazon S3 儲存貯體中的 SBOM zip 封存檔案，您需要資源型許可政策。如需正確的資源型許可政策，請參閱下列範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iot.amazonaws.com"
                ]
        },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucketName/*"
        }
    ]
}
```

如需資源型許可政策的詳細資訊，請參閱 AWS IoT 資源型政策

更新 SBOM

您可以視需要更新軟體物料清單，以保護和增強裝置機群。每次在 Amazon S3 儲存貯體中更新軟體物料清單時，版本 ID 都會變更，而且您必須將新的 Amazon S3 儲存貯體 URL 與適當的軟體套件版本建立關聯。您會在 的套件版本頁面上的 Amazon S3 物件版本 ID 欄中看到新的版本 ID AWS 管理主控台。此外，您可以使用 API 操作`GetPackageVersion`或 CLI 命令`get-package-version`來檢視新版本 ID。

> **Note**
>
> 更新您的軟體物料清單，這將導致新的版本 ID，不會導致建立新的套件版本。

如需 Amazon S3 物件金鑰的詳細資訊，請參閱建立物件金鑰名稱。

啟用 AWS IoT 機群索引

啟用 AWS IoT 機群索引是使用 AWS IoT Device Management 軟體套件目錄的必要條件。若要使用 Software Package Catalog 利用 AWS IoT 機群索引，請將預留的具名影子 ($package) 設定為您要為其編製索引的每個裝置的資料來源，並收集指標。如需預留具名影子的詳細資訊，請參閱 預留已命名影子。

機群索引提供支援，讓 AWS IoT 物件可透過依軟體套件版本篩選的動態物件群組進行分組。例如，機群索引可以識別已安裝或未安裝特定套件版本、未安裝任何套件版本或符合特定名稱/值配對的物件。最後，機群索引提供標準和自訂指標，您可以用來深入了解裝置機群的狀態。如需詳細資訊，請參閱準備機群索引。

> **Note**
>
> 啟用軟體套件目錄的機群索引會產生標準服務費用。如需詳細資訊，請參閱 AWS IoT Device Management定價。

預留已命名影子

預留已命名影子 $package 會反映安裝於裝置的軟體套件和套件版本狀態。機群索引使用預留已命名影子做為資料來源，以建立標準和自訂指標，讓您可以查詢機群狀態。如需詳細資訊，請參閱準備機群索引。

預留已命名影子類似已命名影子，但其名稱為預先定義，無法變更。此外，預留已命名影子不會隨中繼資料更新，且只會使用 `version` 和 `attributes` 關鍵字。

包含其他關鍵字的更新請求，例如 `description`，將在 `rejected`主題下收到錯誤回應。如需詳細資訊，請參閱裝置影子錯誤訊息。

當您透過主控台建立 AWS IoT 物件、 AWS IoT 任務成功完成並更新影子，以及發出 UpdateThingShadow API 操作時，都可以建立。如需詳細資訊，請參閱《 AWS IoT Core 開發人員指南》中的 UpdateThingShadow。

> **Note**
>
> 索引預留已命名影子不會計入機群索引可以索引的已命名影子數量。如需詳細資訊，請參閱 AWS IoT Device Management 機群索引限制和配額。此外，如果您選擇讓 AWS IoT 任務在任務成功完成時更新保留的具名影子，則 API 呼叫會計入您的 Device Shadow 和登錄操作，並可能產生費用。如需詳細資訊，請參閱 AWS IoT Device Management 任務限制和配額和 IndexingFilter API 資料類型。

$package 影子的結構

預留已命名影子包含以下內容：

```
{
    "state": {
        "reported": {
            "<packageName>": {
                "version": "",
                "attributes": {
                }
            }
        }
    },
    "version" : 1
    "timestamp" : 1672531201
}
```

影子屬性會更新為下列資訊：

- `<packageName>`：已安裝軟體套件的名稱，此套件會以 PackageName 參數更新。

- `version`：已安裝套件版本的名稱，此版本會以 versionName 參數更新。
- `attributes`：由裝置儲存並依機群索引編製索引的選用中繼資料。這可讓客戶根據儲存的資料查詢其索引。
- `version`：影子的版本號碼。每次影子更新並開始於 1 時，皆會自動遞增。
- `timestamp`：顯示上次影子更新時間並以 Unix 時間記錄。

如需詳細了解已命名影子的格式和行為，請參閱 AWS IoT Device Shadow 服務 訊息順序。

刪除軟體套件及其套件版本

刪除軟體套件之前，請執行下列動作：

- 確認套件及其版本並未主動部署。
- 先刪除所有相關版本。若其中一個版本為預設版本，您必須從套件中移除具名預設版本。因為您不一定要指定預設版本，移除預設版本並不會造成衝突。若要從軟體套件中移除預設版本，請透過主控台或使用 UpdatePackageVersion API 操作來編輯套件。

只要沒有具名預設套件版本，您便可以使用主控台刪除軟體套件，且其所有套件版本也將一併刪除。若您使用 API 呼叫來刪除軟體套件，您必須先刪除套件版本再刪除軟體套件。

準備安全性

本節討論 AWS IoT Device Management 軟體套件目錄的主要安全需求。

以資源為基礎的身分驗證

軟體套件目錄在更新機群上的軟體時，會使用以資源為基礎的授權來提升安全性。這表示您必須建立 AWS Identity and Access Management (IAM) `read`政策，授予對軟體套件和套件版本執行 `create`、`delete`、、 和 `updatelist`動作的權限，並參考您想要在 `Resources`區段中部署的特定軟體套件和套件版本。您也需要這些權限，以便更新預留已命名影子。您可以在每個實體中納入 Amazon Resource Name (ARN) 來參考軟體套件和套件版本。

> **Note**
>
> 若您希望該政策能授予套件版本 API 呼叫的權限 (例如 CreatePackageVersion、UpdatePackageVersion、DeletePackageVersion)，則您必須在政策中同時納入軟體套件和套件版本 ARN。若您希望該政策能授予軟體套件 API 呼叫的權限 (例

如 CreatePackage、UpdatePackage 和 DeletePackage)，則您在政策中僅能納入軟體套件 ARN。

請遵循下方說明，建構軟體套件和套件版本 ARN：

- 軟體套件：`arn:aws:iot:<region>:<accountID>:package/<packageName>/package`
- 套件版本：`arn:aws:iot:<region>:<accountID>:package/<packageName>/version/<versionName>`

> **Note**
>
> 您可能會在此政策中納入其他相關權利。例如，您可以會納入 job、thinggroup 和 jobtemplate 的 ARN。如需詳細資訊和政策選項的完整清單，請參閱使用 AWS IoT 任務保護使用者和裝置。

例如，若您有如下名稱的軟體套件和套件版本：

- AWS IoT 物件： myThing
- 套件名稱：samplePackage
- 版本 1.0.0

該政策看起來類似以下範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:createPackage",
                "iot:createPackageVersion",
                "iot:updatePackage",
                "iot:updatePackageVersion"
            ],
            "Resource": [
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage",
```

```
                "arn:aws:iot:us-east-1:111122223333:package/samplePackage/
version/1.0.0"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:111122223333:thing/myThing/$package"
        }
    ]
}
```

AWS IoT 部署套件版本的任務權限

基於安全考量，您必須授予部署套件和套件版本的權限，並為允許部署的特定套件和套件版本命名。若要執行這項動作，您可以建立 IAM 角色和政策，以授予透過套件版本部署任務的權限。政策必須將目的地套件版本指定為資源。

IAM 政策

IAM 政策授予建立任務的權限，其中包含在 Resource 區段中命名的套件和版本。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob",
                "iot:CreateJobTemplate"
            ],
            "Resource":[
                "arn:aws:iot:*:111122223333:job/<jobId>",
                "arn:aws:iot:*:111122223333:thing/<thingName>/$package",
                "arn:aws:iot:*:111122223333:thinggroup/<thingGroupName>",
                "arn:aws:iot:*:111122223333:jobtemplate/<jobTemplateName>",
                "arn:aws:iot:*:111122223333:package/<packageName>/
version/<versionName>"
```

```
            ]
        }
    ]
}
```

> **Note**
>
> 如果您想要部署解除安裝軟體套件和套件版本的任務，您必須授權套件版本為 的 ARN$null，如下列所示：

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

AWS IoT 更新預留具名影子的任務權限

若要允許任務在完成時更新物件的預留已命名影子，您必須建立 IAM 角色和政策。您可以在 AWS IoT 控制台透過兩種方式執行此動作。第一，在控制台建立軟體套件時。若您看到啟用套件管理的相依性對話方塊，您可以選擇使用現有角色或建立新角色。或者，在 AWS IoT 主控台選擇設定，接著選擇管理索引以及管理裝置套件和版本的索引。

> **Note**
>
> 如果您選擇讓 AWS IoT 任務服務在任務成功完成時更新保留的具名影子，則 API 呼叫會計入您的 Device Shadow 和登錄操作，並可能產生費用。如需詳細資訊，請參閱 AWS IoT Core 定價。

使用建立角色選項時產生的角色名稱開頭為 `aws-iot-role-update-shadows` 且包含下列政策：

設定角色

許可

權限政策授予查詢和更新物件影子的權限。資源 ARN 中的 `$package` 參數則針對預留已命名影子。

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "iot:DescribeEndpoint",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:111122223333:thing/<thingName>/$package"
            ]
        }
    ]
}
```

信任關係

除了權限政策之外，角色也需要與 AWS IoT Core 的信任關係，以便實體擔任該角色並更新預留已命名影子。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

設定使用者政策

iam:PassRole 權限

最後，當您呼叫 UpdatePackageConfiguration API 操作 AWS IoT Core 時，您必須擁有將角色傳遞至 的許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iot:UpdatePackageConfiguration"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

AWS IoT 從 Amazon S3 下載的任務許可

任務文件會儲存於 Amazon S3。當您透過 AWS IoT 任務分派時，您會參考此檔案。您必須提供 AWS IoT Jobs 下載檔案 (`s3:GetObject`) 的權限。您還必須在 Amazon S3 和 AWS IoT 任務之間設定信任關係。如需建立這些政策的指示，請參閱管理任務中的預先簽章的 URL。

更新套件版本之軟體物料清單的許可

若要更新 `Draft`、 `Published`或 `Deprecated`生命週期狀態下套件版本的軟體物料清單，您需要 AWS Identity and Access Management 角色和政策，以在 Amazon S3 中尋找新的軟體物料清單，並在其中更新套件版本 AWS IoT Core。

首先，您將在版本控制的 Amazon S3 儲存貯體中放入更新的軟體物料清單，並使用包含的 `sboms` 參數呼叫 `UpdatePackageVersion` API 操作。接著，您的授權委託人將擔任您建立的 IAM 角色，在 Amazon S3 中尋找更新的軟體物料清單，並在 中更新軟體套件目錄 AWS IoT Core 的套件版本。

執行此更新需要下列政策：

政策

- 信任政策 與擔任 IAM 角色的授權委託人建立信任關係的政策，以便從 Amazon S3 中的版本控制儲存貯體找到更新的軟體物料清單，並在其中更新套件版本 AWS IoT Core。
  -

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

-

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- 許可政策：存取 Amazon S3 版本控制的儲存貯體的政策，其中儲存套件版本的軟體物料清單，並更新套件版本 AWS IoT Core。

  -

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1"
            ]
        }
    ]
}
```

- ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "iot:UpdatePackageVersion"
              ],
              "Resource": [
                  "arn:aws:iot:*:111122223333:package/<packageName>/
  version/<versionName>"
              ]
          }
      ]
  }
  ```

- 傳遞角色許可：當您呼叫 UpdatePackageVersion API 操作 AWS IoT Core 時，授予許可將 IAM 角色傳遞至 Amazon S3 的政策。

  - ```
    {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "iam:PassRole",
            "s3:GetObject"
          ],
          "Resource": [
            "arn:aws:s3:::awsexamplebucket1"
          ]
        }
      ]
    }
    ```

  - ```
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
    ```
    ```

```
                "iam:PassRole",
                "iot:UpdatePackageVersion"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

> **Note**
>
> 您無法更新已轉換為Deleted生命週期狀態之套件版本的軟體物料清單。

如需為 AWS 服務建立 IAM 角色的詳細資訊，請參閱建立角色以將許可委派給 AWS 服務。

如需建立 Amazon S3 儲存貯體和上傳物件至其中的詳細資訊，請參閱建立儲存貯體和上傳物件。

準備機群索引

使用 AWS IoT 機群索引，您可以使用預留的具名影子 () 來搜尋和彙總資料$package。您也可以透過查詢 預留已命名影子和動態 AWS IoT 物件群組來分組物件。 https://docs.aws.amazon.com/iot/latest/developerguide/dynamic-thing-groups.html例如，您可以找到哪些 AWS IoT 物件使用特定套件版本、未安裝特定套件版本，或未安裝任何套件版本的相關資訊。您可以組合屬性以獲得進一步分析。例如，識別具有特定版本且屬於特定物件類型的物件 （ 例如 1.0.0 版和 pump_sensor 的物件類型 ）。如需詳細資訊，請參閱機群索引。

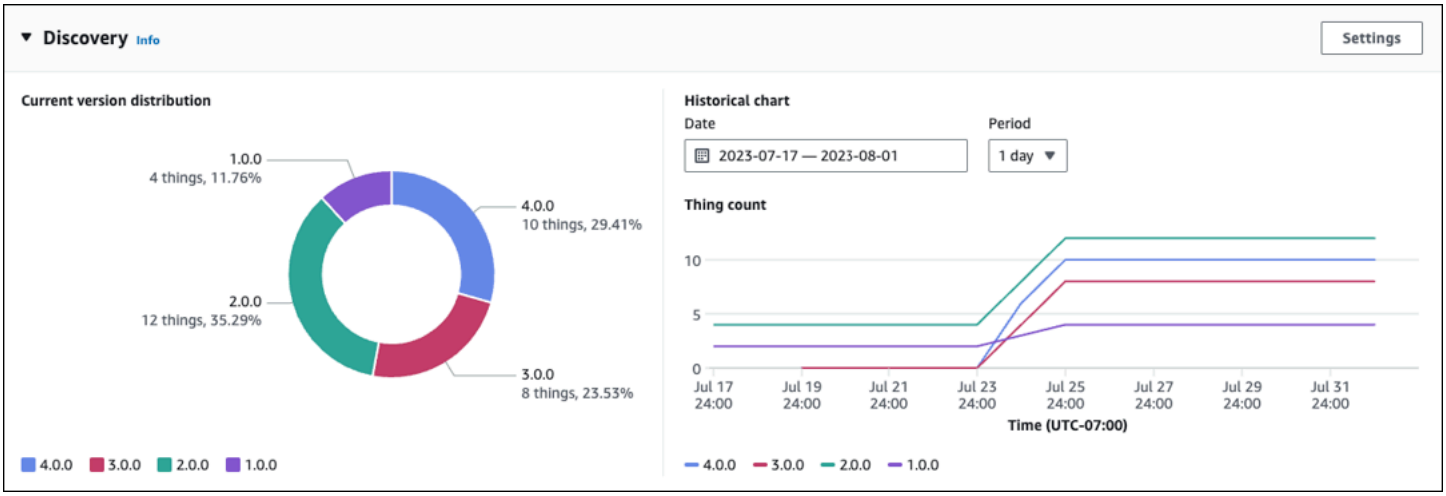
將 $package 影子設定為資料來源

若要搭配使用機群索引與軟體套件目錄，您必須啟用機群索引、將已命名影子設定為資料來源，並定義 $package 為已命名影子篩選條件。若您尚未啟用機群索引，您可以在此程序中啟用。在主控台的 AWS IoT Core 開啟設定，選擇管理索引，接著選擇新增已命名影子、新增裝置軟體套件和版本，以及更新。如需詳細資訊，請參閱管理物件索引。

或者，您也可以在建立第一個套件時啟用機群索引。當啟用套件管理的相依性對話方塊出現時，請選擇將裝置軟體套件和版本新增為資料來源至機群索引。透過此選項，您也可以啟用機群索引。

> **Note**
>
> 啟用軟體套件目錄的機群索引會產生標準服務費用。如需詳細資訊，請參閱 AWS IoT Device Management定價。

控制台顯示的指標



在 AWS IoT 主控台軟體套件詳細資訊頁面上，探索面板會顯示透過$package影子擷取的標準指標。

- 目前版本分佈圖顯示 10 個最新套件版本的裝置數量和百分比，這些版本與與此軟體套件相關聯的所有裝置中的 AWS IoT 物件相關聯。注意：若軟體套件的套件版本多於圖表中標示的版本，您在其他中會發現它們已分組。
- 歷史圖表會顯示指定期間內，與所選套件版本相關聯的裝置數量。圖表一開始會是空白的，您可以選取最多 5 個套件版本並定義日期範圍和時間間隔。若要選取圖表的參數，請選擇設定。歷史圖表顯示的資料可能與目前版本分佈圖表不同，這是因為兩者顯示的套件版本數量不同，也因為歷史圖表可以選擇要分析哪些套件版本。注意：選取要進行視覺化的套件版本時，會計入機群指標限制的上限數量。如需詳細資訊，請參閱機群索引限制和配額。

如需了解收集套件版本分佈的其他方法，請參閱透過 getBucketsAggregation 收集套件版本分佈。

查詢模式

軟體套件目錄的機群索引使用大多數標準支援的功能（例如詞彙、片語和搜尋欄位）。不過，比較運算子（例如，小於<和大於 >) 和range查詢不適用於預留具名影子 ($package) version索引鍵。這些查詢適用於 attributes金鑰。如需詳細資訊，請參閱查詢語法。

範例資料

注意：如需詳細了解預留已命名影子及其結構，請參閱預留已命名影子。

在此範例中，第一部裝置名為 AnyThing 且已安裝下列套件：

- 軟體套件：SamplePackage

  套件版本：1.0.0

  套件 ID：1111

影子如下所示：

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                    }
            }
        }
    }
}
```

第二部裝置名為 AnotherThing 且已安裝下列套件：

- 軟體套件：SamplePackage

  套件版本：1.0.0

  套件 ID：1111

- 軟體套件：OtherPackage

  套件版本：1.2.5

  套件 ID：2222

影子如下所示：

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                }
            },
            "OtherPackage": {
                "version": "1.2.5",
                "attributes": {
                    "s3UrlForOtherPackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile2",
                    "packageID": "2222"
                    }
            },
        }
    }
}
```

範例查詢

下表列出根據 AnyThing 和 AnotherThing 範例裝置影子的查詢範例。如需詳細資訊，請參閱範例物件查詢。

FreeRTOS AWS IoT Device Tester 的最新版本

已請求的資訊	Query	結果
已安裝特定套件版本的物件	shadow.name.$package.reported.Sample	AnyThing, OtherThing

已請求的資訊	Query	結果
	`Package.version:1.0.0`	
未安裝特定套件版本的物件	`NOT shadow.name.$package.reported.OtherPackage.version:1.2.5`	`AnyThing`
所使用套件版本的套件 ID 大於 1500 的任何裝置	`shadow.name.$package.reported.*.attributes.packageID>1500"`	`OtherThing`
已安裝特定套件且安裝多個套件的物件	`shadow.name.$package.reported.SamplePackage.version:1.0.0 AND shadow.name.$package.reported.totalCount:2`	`OtherThing`

透過 **getBucketsAggregation** 收集套件版本分佈

除了 AWS IoT 主控台中的探索面板之外，您也可以使用 GetBucketsAggregation API 操作取得套件版本分佈資訊。若要取得套件版本分佈資訊，您必須執行下列動作：

- 在機群索引中為每個軟體套件定義自訂欄位。注意：建立自訂欄位會計入 AWS IoT 機群索引服務配額。
- 自訂欄位的格式如下所示：

  `shadow.name.$package.reported.<packageName>.version`

如需詳細資訊，請參閱 AWS IoT 機群索引中的自訂欄位區段。

準備 AWS IoT 任務

AWS IoT Device Management Software Package Catalog 透過替代參數擴展 AWS IoT 任務，並與 AWS IoT 機群索引、動態物件群組以及 AWS IoT 物件預留的具名影子整合。

> **Note**
>
> 若要使用軟體套件目錄提供的所有功能，您必須建立這些 AWS Identity and Access Management (IAM) 角色和政策：AWS IoT 部署套件版本的任務權限和AWS IoT 更新預留具名影子的任務權限。如需詳細資訊，請參閱準備安全性。

AWS IoT 任務的替代參數

您可以使用替代參數做為 AWS IoT 任務文件中的預留位置。當任務服務遇到替換參數時，會將任務指向參數值的具名軟體版本屬性。您可以使用此程序建立單一任務文件，並透過一般用途屬性將中繼資料傳遞至任務。例如，您可以透過套件版本屬性將 Amazon Simple Storage Service (Amazon S3) URL、軟體套件 Amazon Resource Name (ARN) 或簽章傳遞至任務文件。

替代參數應該在任務文件中格式化，如下所示：

- 軟體套件名稱和套件版本
  - 之間的空字串`package::version`代表軟體套件名稱替換參數。之間的空字串`version::attribute`代表軟體套件版本替換參數。請參閱下列範例，以使用任務文件中的套件名稱和套件版本取代參數：`${aws:iot:package::version::attributes:<attributekey>}`。
  - 任務文件將使用套件版本詳細資訊中的版本 ARN 自動填入這些取代參數。如果您要使用 API 或 CLI 命令為單一套件部署建立任務或任務範本，則套件版本的版本 ARN 會以 `CreateJob`和 中的 `destinationPackageVersions` 參數表示`DescribeJob`。
- 軟體套件版本的所有屬性
  - 請參閱下列範例，以了解在任務文件中使用軟體套件版本替換參數的所有屬性：`${aws:iot:package:<packageName>:version:<versionName>:attributes}`

> **Note**
>
> 套件名稱、套件版本和所有屬性替代參數可以一起使用。請參閱下列範例，以使用任務文件中所有三個替代參數： `${aws:iot:package::version::attributes}`

在下列範例中，有一個名為 的軟體套件，`samplePackage`並且有一個名為 `2.1.5` 的套件版本，具有下列屬性：

- 名稱：`s3URL`，值：`https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/exampleCodeFile`
  - 此屬性會識別儲存於 Amazon S3 的程式碼檔案位置。
- 名稱：`signature`，值：`aaaaabbbbbccccddddeeeeeffffgggggghhhhhiiiiijjjj`
  - 此屬性會提供裝置做為安全措施所需的程式碼簽章值。如需詳細資訊，請參閱適用於任務的程式碼簽署。注意：此屬性僅為範例，不是軟體套件目錄或任務的必要部分。

針對 `s3URL`，任務文件參數的寫入方式如下：

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
}
```

針對 `signature`，任務文件參數的寫入方式如下：

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
}
```

完整的任務文件寫入方式如下：

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage":
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
```

```
      },
    ],
    "signature": [
      "samplePackage" :
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
    ]
  }
}
```

進行替換後，下列任務文件會部署至裝置：

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage": "https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/
exampleCodeFile"
      },
    ],
    "signature": [
      "samplePackage" : "aaaaabbbbbcccccdddddeeeeeffffffgggggghhhhhiiiiijjjj"
    ]
  }
}
```

替代參數（檢視之前和之後）

替代參數使用各種旗標簡化任務文件的建立，例如$default預設套件版本。這樣就不需要為每個任務部署手動輸入特定套件版本中繼資料，因為這些旗標會在特定套件版本中自動填入參考的中繼資料。如需套件版本屬性的詳細資訊，例如$default預設套件版本的 ，請參閱 準備任務文件和套件版本以進行部署。

在 中 AWS 管理主控台，在套件版本的任務部署期間，切換部署指示檔案編輯器視窗中的預覽替代按鈕，以檢視包含和不含替代參數的任務文件。

使用 DescribeJob和 GetJobDocument APIs中的「替代前」參數，您可以在移除替代參數之前和之後檢視 API 回應。請參閱下列使用 DescribeJob和 GetJobDocument APIs的範例：

- DescribeJob
  - 預設檢視

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/1.0.2"]
}
```

- 取代檢視之前

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/$default"]
}
```

- GetJobDocument
  - 預設檢視

```
{
    "attributes": {
        "location": "prod-artifacts.s3.us-east-1.amazonaws.com/mqtt-core",
        "signature": "IQoJb3JpZ2luX2VjEIrwEaCXVzLWVhc3QtMSJHMEUCIAofPNPpZ9cI",
        "streamName": "mqtt-core",
        "fileId": "0"
    },
}
```

  - 取代檢視之前

```
{
    "attributes": "${aws:iot:package:TestPackage:version:$default:attributes}",
}
```

如需 AWS IoT 任務、建立任務文件和部署任務的詳細資訊，請參閱任務。

準備任務文件和套件版本以進行部署

建立套件版本時，它處於 `draft` 狀態，表示它已準備好進行部署。若要準備套件版本以供部署，您必須建立任務文件、將文件儲存在任務可存取的位置 （例如 Amazon S3)，並確認套件版本具有您希望任務文件使用的屬性值。（注意：您只能在套件版本處於 `draft` 狀態時更新其屬性。)

當您為單一套件部署建立 AWS IoT 任務或任務範本時，您有下列選項可自訂您的任務文件：

部署指示檔案 (**`recipe`**)

- 套件版本的部署指示檔案包含部署指示，包括內嵌任務文件，用於將套件版本部署到多個裝置。檔案會將特定部署指示與套件版本建立關聯，以快速且有效率地進行任務部署。

  在 中 AWS 管理主控台，您可以在建立新套件工作流程的版本部署組態索引標籤的部署指示檔案預覽視窗中建立 檔案。您可以利用 從建議的檔案使用 Start 從 AWS IoT 套件版本屬性 AWS IoT 自動產生指令檔案，或使用儲存在 Amazon S3 儲存貯體中的現有任務文件使用您自己的部署指令檔案。

  > **Note**
  >
  > 如果您使用自己的任務文件，可以直接在部署指示檔案預覽視窗中進行更新，但不會自動更新存放在 Amazon S3 儲存貯體中的原始任務文件。

  使用 AWS CLI 或 API 命令時，例如 `CreatePackageVersion`、 `GetPackageVersion`或 `UpdatePackageVersion`， `recipe`代表部署指令檔案，其中包含內嵌任務文件。

  如需任務文件的詳細資訊，請參閱 基本概念。

  請參閱下列範例，了解以 表示的部署指示檔案`recipe`：

```
{
    "packageName": "sample-package-name",
    "versionName": "sample-package-version",
    ...
    "recipe": "{...}"
}
```

> **Note**
>
> 當套件版本處於 published 狀態時，recipe可以更新 表示的部署指示檔案，因為它與套件版本中繼資料分開。它在任務部署期間變得不可變。

Artifact 版本屬性

- 您可以在軟體套件版本artifact中使用版本屬性，為套件版本成品新增 Amazon S3 位置。使用 AWS IoT Jobs 觸發套件版本的任務部署時，任務文件中預先簽章的 URL 預留位置將使用 Amazon S3 儲存貯 ${aws:iot:package:<*packageName*>:version:<*versionName*>:artifact-location:*s3-presigned-url*}體、儲存貯體金鑰和存放在 Amazon S3 儲存貯體中的檔案版本進行更新。儲存套件版本成品的 Amazon S3 儲存貯體所在區域必須與建立套件版本的區域相同。

  > **Note**
  >
  > 若要將相同檔案的多個物件版本儲存在 Amazon S3 儲存貯體中，您必須在儲存貯體上啟用版本控制。如需詳細資訊，請參閱在儲存貯體上啟用版本控制。

  若要在使用 CreatePackageVersion或 UpdatePackageVersion API 操作時存取 Amazon S3 儲存貯體中的套件版本成品，您必須具有下列許可：

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": "s3:GetObjectVersion",
              "Resource": "arn:aws:s3:::bucket-name/key-name"
          }
      ]
  }
  ```

  如需 CreatePackageVersion和 UpdatePackageVersion API 操作artifact中版本屬性的詳細資訊，請參閱 CreatePackageVersion 和 UpdatePackageVersion。

  請參閱下列範例，其中顯示建立新套件版本時artifact支援 Amazon S3 中成品位置的版本屬性：

```
{
    "packageName": "sample package name",
    "versionName": "1.0",
    "artifact": {
        "s3Location": {
            "bucket": "firmware",
            "key": "image.bin",
            "version": "12345"
        }
    }
}
```

> **Note**
>
> 當套件版本從draft狀態狀態更新為published狀態時，套件版本屬性和成品位置會變得不可變。若要更新此資訊，您需要建立新的套件版本，並在draft處於 狀態時執行這些更新。

套件版本

- 預設軟體套件版本可以在軟體套件的可用版本中表示，以提供安全且穩定的套件版本。在使用 AWS IoT Jobs 將預設套件版本部署到裝置機群時，這可做為軟體套件的基準版本。建立任務以部署軟體$default套件的套件版本時，任務文件和新任務部署中的套件版本必須與 相符$default。任務部署中的套件版本由 表示destinationPackageVersions，適用於 API 和 CLI 命令，以及VersionARN中的 AWS 管理主控台。任務文件中的套件版本由下列任務文件預留位置表示，如下所示：

  ```
  arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$default
  ```

  若要使用預設套件版本建立任務或任務範本，請使用 CreateJob或 CreateJobTemplate API 命令中的 $default旗標，如下所示：

  ```
  "$ aws iot create-job \
      --destination-package-versions "arn:aws:iot:us-west-2:123456789012:package/
  TestPackage/version/$default"
      --document file://jobdoc.json
  ```

> **Note**
>
> 參考預設版本的`$default`套件版本屬性是選用屬性，只有在透過 AWS IoT Jobs 參考任務部署的預設套件版本時才需要。

當您滿意套件版本時，請透過 AWS IoT 主控台中的軟體套件詳細資訊頁面或發出 UpdatePackageVersion API 操作來發佈套件版本。然後，您可以在透過 AWS IoT 主控台或發出 CreateJob API 操作建立任務時參考套件版本。

部署時命名套件和版本

若要將軟體套件版本部署到裝置，請確認任務文件中參考的軟體套件和套件版本符合 `CreateJob` API 操作中 `destinationPackageVersions` 參數中所述的軟體套件和套件版本。如果不相符，您會收到錯誤訊息，提示您讓兩個參考相符。如需軟體套件目錄錯誤訊息的詳細資訊，請參閱 對錯誤訊息進行一般故障診斷。

除了任務文件中參考的軟體套件和套件版本之外，您還可以在任務文件中未參考的 `CreateJob` API 操作中的 `destinationPackageVersions` 參數中包含其他軟體套件和套件版本。確保裝置的任務文件中包含必要的安裝資訊，以正確安裝其他軟體套件版本。如需 `CreateJob` API 操作的詳細資訊，請參閱 CreateJob。

透過 AWS IoT 動態物件群組將任務設為目標

軟體套件目錄可與機群索引、AWS IoT 任務和AWS IoT 動態物件群組搭配使用，以篩選並鎖定機群中的裝置，選取要部署至裝置的套件版本。您可以根據裝置目前的套件資訊執行機群索引查詢，並將這些物件設為 AWS IoT 任務的目標。您也可以發行軟體更新，但僅限符合資格的目標裝置。例如，您可以指定只要將組態部署到目前執行 `iot-device-client 1.5.09` 的裝置。如需詳細資訊，請參閱建立動態物件群組。

預留已命名影子和套件版本

如果已設定， AWS IoT Jobs 可以在任務成功完成時更新物件的預留具名影子 (`$package`)。如此一來，您便不需要手動將套件版本與物件的預留已命名影子建立關聯。

在下列情況下，您可以選擇手動更新套件版本，或將其與物件的預留已命名影子建立關聯：

- 您可以將物件註冊到 ， AWS IoT Core 而不關聯已安裝的套件版本。

- AWS IoT 任務未設定為更新物件的預留具名影子。
- 您可以使用內部程序將套件版本分派至您的機群，該程序不會在完成 AWS IoT Core 時更新。

> **Note**
>
> 我們建議您使用 AWS IoT Jobs 來更新預留具名影子 () 中的套件版本$package。當 AWS IoT 工作也設定為更新影$package子時，透過其他程序 （ 例如手動或程式設計 API 呼叫 ） 更新影子中的版本參數，可能會導致裝置的實際版本與報告給預留具名影子的版本之間不一致。

您可以透過主控台或 UpdateThingShadow API 操作，將套件版本新增或更新至物件的預留已命名影子 ($package)。如需詳細資訊，請參閱將套件版本與 AWS IoT 物件建立關聯。

> **Note**
>
> 將套件版本與 AWS IoT 物件建立關聯並不會直接更新裝置軟體。您必須將套件版本部署至裝置，才能更新裝置軟體。

解除安裝軟體套件及其套件版本

$null 是預留預留位置，提示 AWS IoT Jobs 服務從裝置的預留具名影子 中移除現有的軟體套件和套件版本$package。如需詳細資訊，請參閱預留已命名影子。

若要使用此功能，請將 destinationPackageVersion Amazon Resource Name (ARN) 結尾的版本名稱取代為 $null。之後，您必須指示您的服務從裝置中移除軟體。

授權的 ARN 使用以下格式：

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

例如

```
$ aws iot create-job \
    ... \
    --destinationPackageVersions ["arn:aws:iot:us-east-1:111122223333:package/
samplePackage/version/$null"]
```

開始使用軟體套件目錄

您可以透過 AWS 管理主控台、 AWS IoT Core API 操作和 AWS Command Line Interface () 來建置和維護 AWS IoT Device Management 軟體套件目錄AWS CLI。

> **Note**
>
> 啟用 AWS IoT 機群索引是使用軟體套件目錄的必要條件。基本操作，例如在 中建立軟體套件版本 AWS 管理主控台 ，以及使用 `CreatePackage` API 命令，會在未啟用 AWS IoT 機群索引的情況下失敗。
> 如需搭配 Software Package Catalog 使用 AWS IoT 機群索引的詳細資訊，請參閱 準備機群索引。

使用主控台

若要使用 AWS 管理主控台，請登入 AWS 您的帳戶並導覽至 AWS IoT Core。在導覽窗格中，選擇軟體套件。然後，您可以從本章節建立管理套件及其版本。

使用 API 或 CLI 操作

您可以使用 AWS IoT Core API 操作來建立和管理 Software Package Catalog 功能。如需詳細資訊，請參閱 AWS IoT API 參考和 AWS SDK 與工具組。這些 AWS CLI 命令也會管理您的目錄。如需詳細資訊，請參閱 AWS IoT CLI 指令參考。

本章包含下列部分：

- 建立軟體套件和套件版本
- 透過 AWS IoT 任務部署套件版本
- 將套件版本與 AWS IoT 物件建立關聯

建立軟體套件和套件版本

您可以使用下列步驟，透過 AWS 管理主控台建立套件和初始版本物件。

若要建立軟體套件

1. 登入 AWS 您的帳戶並導覽至 AWS IoT 主控台。

2. 在導覽窗格中，選擇軟體套件。
3. 在 AWS IoT 軟體套件頁面上，選擇建立套件。啟用套件管理的相依性對話方塊隨即出現。
4. 在機群索引下，選取新增裝置軟體套件和版本。這是軟體套件目錄的必要條件，並會提供機群的機群索引和指標。
5. 【選用】 如果您希望 AWS IoT 任務在任務成功完成時更新保留的具名影子，請選取從任務自動更新影子。如果您不希望 AWS IoT 任務進行此更新，請取消選取此核取方塊。
6. 【選用】 若要授予 AWS IoT 任務更新預留具名影子的權限，請在選取角色下，選擇建立角色。如果您不希望 AWS IoT 任務進行此更新，則不需要此角色。
7. 建立或選取角色。

   a. 若您沒有此用途的角色：顯示建立角色對話方塊時，請輸入角色名稱，然後選擇建立。

   b. 若您具有此用途的角色：針對選取角色，請選擇您的角色，然後確定已選取將政策附加至 IAM 角色核取方塊。
8. 選擇確認。建立新套件頁面隨即出現。
9. 在套件詳細資訊下，輸入套件名稱。
10. 在套件描述下，輸入相關資訊協助您識別管理此套件。
11. [選用] 您可以使用標籤來協助您分類管理此套件。若要新增標籤，請展開標籤，選擇新增標籤，然後輸入金鑰-值配對。您最多可輸入 50 個標籤。如需詳細資訊，請參閱標記您的 AWS IoT 資源。

若要在建立新套件時新增套裝版本

1. 在初始版本下，輸入版本名稱。

   建議您使用 SemVer 格式 (例如 `1.0.0.0`) 來專屬識別您的套件版本。您也可以使用更適合您使用案例的其他格式方式。如需詳細資訊，請參閱套件版本生命週期。
2. 在版本描述下，輸入相關資訊協助您識別管理此套件版本。

   > **Note**
   >
   > 預設版本核取方塊會停用，因為套件版本是在 `draft` 狀態下建立的。您可以在建立套件版本之後，以及將狀態變更為 時，命名預設版本`published`。如需詳細資訊，請參閱套件版本生命週期。
3. [選用] 若要協助您管理此版本或將資訊傳送給您的裝置，請為版本屬性輸入一或多個名稱/值配對。為您輸入的每個名稱/值配對選擇新增屬性。如需詳細資訊，請參閱版本屬性。

4. [選用] 您可以使用標籤來協助您分類管理此套件。若要新增標籤，請展開標籤，選擇新增標籤，然後輸入金鑰-值配對。您最多可輸入 50 個標籤。如需詳細資訊，請參閱標記您的 AWS IoT 資源。
5. 選擇下一步。

將軟體物料清單與套件版本建立關聯（選用）

1. 在步驟 3：SBOMs（選用）上，選擇預設 SBOM 檔案格式和驗證模式，用於驗證您的軟體物料清單，然後再與套件版本相關聯。
2. 在新增 SBOM 檔案視窗中，輸入代表您版本控制的 Amazon S3 儲存貯體的 Amazon Resource Name (ARN)，如果預設類型無法運作，則輸入偏好的 SBOM 檔案格式。

> **Note**
>
> 如果您的套件版本有多個軟體物料清單，您可以新增單一 SBOM 檔案或包含多個 SBOMs 的單一 zip 檔案。

3. 在新增的 SBOM 檔案視窗中，您可以檢視您為套件版本新增的 SBOM 檔案。
4. 選擇建立套件和版本。套件版本頁面隨即出現，您可以在新增的 SBOM 檔案視窗中查看 SBOM 檔案的驗證狀態。當 SBOM 檔案經過驗證`In progress`時，初始狀態將為 。

> **Note**
>
> SBOM 檔案驗證狀態為 `Invalid file`、`Not started`、`Validated (CycloneDX)`、、 `In progress` `Validated (SPDX)`和驗證失敗原因。

透過 AWS IoT 任務部署套件版本

您可以使用下列步驟，透過 AWS 管理主控台部署套件版本。

先決條件：

開始之前，請執行以下動作：

- 向 註冊 AWS IoT 物件 AWS IoT Core。如需將裝置新增至其中的指示 AWS IoT Core，請參閱建立物件。

- 【選用】 建立 AWS IoT 物件群組或動態物件群組，以您要部署套件版本的裝置為目標。如需相關指示建立物件群組，請參閱建立靜態物件群組。如需相關指示建立動態物件群組，請參閱建立動態物件群組。
- 建立軟體套件和套件版本。如需詳細資訊，請參閱建立軟體套件和套件版本。
- 建立任務文件。如需詳細資訊，請參閱準備任務文件和套件版本以進行部署。

部署 AWS IoT 任務

1. 在 AWS IoT 主控台上，選擇軟體套件。
2. 選擇您要部署的軟體套件。軟體套件詳細資訊頁面隨即顯示。
3. 在版本之下選擇您要部署的套件版本，然後選擇部署任務版本。
4. 若這是您第一次透過此入口網站部署任務，您會看到附有規範說明的對話方塊。請檢閱資訊，然後選擇確認。
5. 輸入部署操作的名稱，或在名稱欄位保留自動產生的名稱。
6. [選用] 在描述欄位中，輸入識別部署目的或內容的描述，或保留自動產生的資訊。

   注意：建議您不要在任務名稱和描述欄位中使用個人識別資訊。
7. [選用] 新增任何要與此任務建立關聯的標籤。
8. 選擇下一步。
9. 在任務目標之下，選擇應負責接收任務的物件或物件群組。
10. 在任務檔案欄位中，指定任務文件 JSON 檔案。
11. 開啟與套件目錄服務整合的任務。
12. 選取任務文件中所指定的套件和版本。

    > **Note**
    >
    > 您必須選擇在任務文件中所指定的相同套件和套件版本。您可以納入更多項目，但該任務只會針對任務文件中包含的套件和版本發出指示。如需詳細資訊，請參閱部署時命名套件和版本。
13. 選擇下一步。
14. 在任務組態頁面上，選取任務組態對話方塊中下列任一任務類型：

    - 快照任務：快照任務在目標裝置和群組上完成執行後即完成。
    - 連續任務：連續任務適用於物件群組，並在稍後新增至所指定目標群組的任何裝置上執行。

15. 在其他組態 - 選用對話方塊中，檢閱下列選用任務組態，並視需要進行選擇。如需詳細資訊，請參閱任務推展、排程和中止組態和任務執行逾時和重試組態。
    - 推展組態
    - Scheduling configuration (排程組態)
    - 任務執行逾時組態
    - 任務執行重試組態
    - 中止組態
16. 檢視任務選擇，然後選擇送出。

建立任務後，主控台會產生 JSON 簽章，並將其放置在您的任務文件中。您可以使用 AWS IoT 主控台來檢視任務的狀態，或取消或刪除任務。若要管理任務，請前往主控台的任務中心。

將套件版本與 AWS IoT 物件建立關聯

在裝置上安裝軟體後，您可以將套件版本與 AWS IoT 物件的預留具名影子建立關聯。如果 AWS IoT 任務已設定為在任務部署並成功完成之後更新物件的預留具名影子，則您不需要完成此程序。如需詳細資訊，請參閱預留已命名影子。

先決條件：

開始之前，請執行以下動作：

- 建立 AWS IoT 實物或 實物，並透過 建立遙測 AWS IoT Core。如需詳細資訊，請參閱 入門 AWS IoT Core。
- 建立軟體套件和套件版本。如需詳細資訊，請參閱建立軟體套件和套件版本。
- 在裝置上安裝套件版本軟體。

> **Note**
>
> 將套件版本與 AWS IoT 物件建立關聯並不會更新或在實體裝置上安裝軟體。套件版本必須部署至裝置。

將套件版本與 AWS IoT 物件建立關聯

1. 在 AWS IoT 主控台瀏覽窗格中，展開所有裝置選單，然後選擇物件。

2. 從清單中識別您要更新的 AWS IoT 物件，然後選擇物件名稱以顯示其詳細資訊頁面。
3. 在詳細資訊區段中，選擇套件和版本。
4. 選擇新增至套件和版本。
5. 針對選擇裝置套件，選擇您想要的軟體套件。
6. 針對選擇版本，選擇您想要的軟體版本。
7. 選擇新增裝置套件。

   套件和版本會顯示在所選套件和版本清單中。
8. 針對您要與此物件建立關聯的每個套件和版本重複這些步驟。
9. 完成後，請選擇新增套件和版本詳細資訊。物件詳細資訊頁面隨即開啟，您可以在清單中看到新的套件與版本。

AWS IoT 任務

使用 AWS IoT 任務來定義一組遠端操作，這些操作可以傳送到 ，並在一或多個連接的裝置上執行 AWS IoT。例如，您可以定義一個任務，指示一組裝置下載並安裝應用程式、執行韌體更新、重新啟動、輪換憑證，或者執行遠端故障排除操作。

存取 AWS IoT 任務

您可以使用 主控台或 AWS IoT Core API 來開始使用 AWS IoT 任務。

使用主控台

登入 AWS 管理主控台，然後前往 AWS IoT 主控台。在導覽窗格中，選擇 Manage (管理)，然後選擇 Jobs (任務)。您可以從此區段建立和管理任務。如果您想建立和管理任務範本，請在導覽窗格中選擇 Job templates (任務範本)。如需詳細資訊，請參閱使用 建立和管理任務 AWS 管理主控台。

使用 API 或 CLI

您可以使用 AWS IoT Core API 操作開始。如需詳細資訊，請參閱 AWS IoT API 參考。SDK 支援建立任務的 AWS IoT Core AWS IoT API AWS 。如需詳細資訊，請參閱AWS 開發套件與工具組。

您可以使用 AWS CLI 執行命令來建立和管理任務和任務範本。如需詳細資訊，請參閱 AWS IoT CLI 參考。

AWS IoT 任務區域和端點

AWS IoT 任務支援 專屬的控制平面和資料平面 API 端點 AWS 區域。資料平面 API 端點專屬於您的 AWS 帳戶 和 AWS 區域。如需 AWS IoT 任務端點的詳細資訊，請參閱《 AWS 一般參考》中的 AWS IoT Device Management 任務資料端點。

什麼是遠端操作？

> **Note**
>
> Over-the-air(OTA) 更新不適用於下列區域：
>
> - 歐洲 (西班牙)

- 亞太地區 (馬來西亞)

遠端操作是您可以在實體裝置、虛擬裝置或端點上執行的任何更新或動作，無需操作員或技術人員的實體存在即可遠端完成。遠端操作使用over-the-air(OTA) 更新執行，因此您的裝置不需要實際存在。在中管理裝置機群， AWS 雲端 可讓您在裝置註冊時對裝置執行遠端操作 AWS IoT Core。

AWS IoT Device Management 任務提供可擴展的方法，可讓您在註冊的裝置上執行遠端動作 AWS IoT Core。任務會在 中建立 AWS 雲端 ，並透過 MQTT 或 HTTP 通訊協定使用 OTA 更新推送至所有目標裝置。

AWS IoT Device Management 任務可讓您以安全、可擴展且更具成本效益的方式執行遠端操作，例如原廠重設、裝置重新啟動和軟體 OTA 更新。

如需 的詳細資訊 AWS IoT Core，請參閱 什麼是 AWS IoT？。

如需 AWS IoT Device Management 任務的詳細資訊，請參閱 什麼是 AWS IoT 任務？。

使用 AWS IoT Device Management 任務進行遠端操作的優點

使用 AWS IoT Device Management 任務來執行遠端操作可簡化裝置機群的管理。下列清單重點介紹使用 AWS IoT Device Management 任務來執行遠端操作的一些主要優點：

- 與其他 無縫整合 AWS 服務
  - AWS IoT Device Management 任務與下列附加價值 AWS 服務 和 功能緊密整合：
    - Amazon S3：將遠端操作說明存放在安全 Amazon S3 儲存貯體中，您可以在其中控制該內容的存取許可。使用 Amazon S3 儲存貯體提供可擴展且耐用的儲存解決方案，可原生地與 AWS IoT Device Management Software Package Catalog 互動，允許 AWS IoT Device Management 任務在更新指示中參考和替換。如需詳細資訊，請參閱什麼是 Amazon S3？。
    - Amazon CloudWatch：監控並記錄每個裝置的任務執行遠端操作實作狀態，以及追蹤和分析 AWS IoT Device Management 任務中整體任務效能的其他裝置活動。如需詳細資訊，請參閱什麼是 Amazon CloudWatch？ 監控任務日誌並擷取歷史資料以進行故障診斷。它如何與 任務搭配使用。
    - AWS IoT Device Shadow 服務：使用 AWS IoT Device Management Jobs 透過裝置影子維護 AWS IoT 物件的數位表示，因此無論裝置連線為何，您裝置的狀態都可供應用程式和其他 服務使用。如需詳細資訊，請參閱AWS IoT Device Shadow 服務。
- 安全最佳實務

  - 許可控制：使用 Amazon S3 控制遠端操作說明的存取許可，並判斷哪些 IAM 使用者可以使用 AWS IoT 政策和 IAM 使用者角色將遠端操作說明部署到您的裝置機群。
    - 如需 AWS IoT 政策的詳細資訊，請參閱 建立 AWS IoT 政策。
    - 如需 IAM 使用者角色的詳細資訊，請參閱 的身分和存取管理 AWS IoT。
- 可擴展性
  - 目標任務部署：使用建立任務時在任務文件中輸入的特定裝置分組條件，控制哪些裝置從具有目標任務部署的任務接收任務文件。為每個裝置建立 AWS IoT 物件，並將該資訊儲存在 AWS IoT 登錄檔中，可讓您使用機群索引執行目標搜尋。您可以根據機群索引搜尋結果建立自訂群組，以支援您的目標任務部署。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。使用任務執行快照與連續任務。
  - 任務狀態：追蹤任務文件推展至裝置機群的狀態，以及裝置機群層級的整體任務狀態，以及每個裝置上的任務文件個別實作狀態。如需詳細資訊，請參閱任務和任務執行狀態。
  - 新的裝置可擴展性：透過將任務文件新增至透過持續任務使用機群索引建立的現有自訂群組，輕鬆將任務文件部署到新裝置。這將節省您單獨將任務文件部署到每個新裝置所需的時間。或者，您可以透過將任務文件部署到預定的裝置群組一次，然後任務完成，來使用更有針對性的快照方法。
- 彈性
  - 任務組態：使用選用的任務組態推展、排程、中止、逾時和重試來自訂您的任務和任務文件，以滿足您的特定需求。如需詳細資訊，請參閱任務 組態。
- 具成本效益
  - 透過利用 AWS IoT Device Management 任務部署關鍵更新並執行例行維護任務，引進更有效率的成本結構來維護裝置機群。維護裝置機群do-it-yourself(DIY) 解決方案包括經常性、可變成本，例如託管和管理 DIY 解決方案所需的基礎設施、開發、維護和擴展 DIY 解決方案的人力成本，以及資料傳輸成本。利用 AWS IoT Device Management 任務的透明固定成本結構，除了促進任務文件推展到裝置機群和追蹤每個裝置的任務執行狀態所需的資料傳輸成本之外，您確切知道裝置的每個任務執行成本。如需詳細資訊，請參閱 AWS IoT Core 定價。

什麼是 AWS IoT 任務？

使用 AWS IoT 任務來定義一組遠端操作，這些操作可以傳送到 ，並在一或多個連接的裝置上執行 AWS IoT。

若要建立任務，請先定義任務文件，其中包含說明裝置必須從遠端執行之操作的指示清單。若要執行這些操作，請指定目標清單，其為個別物件、物件群組，或兩者皆是。任務文件和目標一起構成部署。

每個部署都可以有其他組態：

- 推展：此組態定義每分鐘接收任務文件的裝置數量。
- 中止：如果有特定數量的裝置未接收到任務通知，請使用此組態取消任務。這樣可避免將錯誤的更新傳送至整個機群。
- 逾時：如果在特定期間內未接收到任務目標的回應，則任務可能會失敗。您可以追蹤在這些裝置上執行的任務。
- 重試：如果裝置報告失敗或任務逾時，您可以使用 AWS IoT 任務自動將任務文件重新傳送至裝置。
- 排程：此組態可讓您排定未來日期與時間的任務。它也可讓您建立週期性維護時段，在預先定義的低流量期間更新裝置。

AWS IoT 任務會傳送訊息，通知目標有可用的任務。目標透過下載任務文件、執行其指定的操作，以及報告其進度，開始執行任務 AWS IoT。您可以執行 Jobs 提供的命令，追蹤特定目標或所有目標 AWS IoT 的任務進度。任務開始時，它的狀態為 In progress (進行中)。然後，裝置會在顯示此狀態的同時報告增量更新，直到任務成功、失敗或逾時為止。

以下主題介紹任務的一些關鍵概念，以及任務和任務執行的生命週期。

主題

- 任務的重要概念
- 任務和任務執行狀態

任務的重要概念

下列概念提供有關 AWS IoT 任務的詳細資訊，以及如何建立和部署任務，以在裝置上執行遠端操作。

基本概念

以下是使用 AWS IoT 任務時必須知道的基本概念。

任務

任務為傳送至連接至 AWS IoT的一或多個裝置，並在其上執行的遠端作業。例如，您可以定義一個任務，指示一組裝置下載並安裝應用程式或執行韌體更新、重新啟動、輪換憑證，或者執行遠端故障排除操作。

任務文件

若要建立任務，您必須先建立任務文件，而其為裝置要執行之遠端操作的描述。

任務文件為 UTF-8 編碼的 JSON 文件，其中包含您裝置執行任務所需的資訊。任務文件包含一個或多個 URL，讓裝置可以下載更新或其他資料。任務文件可以存放於 Amazon S3 儲存貯體，或者內嵌於建立任務的命令。

> **Note**
>
> 在 MQTT 回應中， `jobDocument` 欄位是 JSON 物件。在 HTTP 回應中，它是 JSON 物件的字串表示法。

目標

當您建立任務時，要指定目標的清單，也就是要執行操作的裝置。目標可以是物件或物件群組，或同時為兩者。 AWS IoT Jobs 服務會傳送訊息至每個目標，通知該目標有可用的任務。

部署

提供任務文件並指定目標清單來建立任務後，隨後將任務文件部署至要為其執行更新的遠端目標裝置。若為快照任務，任務將在部署至目標裝置之後完成。若為持續任務，任務會在裝置新增至群組時部署到裝置群組。

任務執行

任務執行即目標裝置上任務的執行個體。目標會下載任務文件，藉此開始執行任務。然後，它會執行文件中指定的操作，並向 報告其進度 AWS IoT。執行編號是特定目標上不會重複的任務執行之識別符。 AWS IoT Jobs 服務提供命令來追蹤目標的任務執行進度，以及所有目標的任務進度。

任務類型概念

下列概念可協助您進一步了解您可以使用 任務建立的不同類型的 AWS IoT 任務。

快照任務

在預設情況下，當您建立任務，該任務會傳送至您指定的所有目標。這些目標完成任務之後 (或者回報無法完成任務)，任務即視為完成。

持續任務

當您建立任務，持續任務會傳送至您指定的所有目標。它會繼續執行並傳送任何新的裝置 (物件) 到已新增的目標群組。例如：持續任務加入群組時，可用於部署或更新裝置。您可以在建立任務時設定選用參數，藉此讓任務可以持續。

> **Note**
>
> 使用動態物件群組定位 IoT 機群時，我們建議您使用連續任務而不是快照任務。使用連續任務，加入群組的裝置即使在任務建立之後，也會收到任務執行。

預先簽章的 URL

為了以安全、限時的方式存取任務文件以外的資料，您可以使用預先簽章的 Amazon S3 URL。將資料置於 Amazon S3 儲存貯體，並在任務作文件中的資料新增一個預留位置連結。當 AWS IoT 任務收到任務文件的請求時，它會透過尋找預留位置連結來剖析任務文件，然後使用預先簽章的 Amazon S3 URLs 取代連結。

預留位置連結的格式如下：

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

其中，*bucket* 是您的儲存貯體名稱，而 *key* 是您所連結在儲存貯體中的物件。

在北京和寧夏區域中，僅當資源擁有者具有 ICP (網際網路內容提供者) 授權時，預先簽章的 URL 才能運作。如需詳細資訊，請參閱中國服務入門文件中的 AWS Amazon Simple Storage Service。

任務組態概念

以下概念可幫助您瞭解如何設定任務。

推展

您可以指定何時通知目標有一項待執行的任務。這樣可以建立一個分階段推展的任務，讓您更輕易管理更新、重新啟動以及其他操作。您可以使用靜態推展率或指數推展率來建立推展組態。若要指定每分鐘通知任務目標的數量上限，請使用靜態推展率。

如需設定推展率的範例以及設定任務推展的更多資訊，請參閱 任務推展、排程和中止組態。

排程

任務排程可讓您針對連續和快照任務，排定將任務文件推展至目標群組中所有裝置的時間範圍。此外，您可以建立選用的維護時段，其中包含任務將任務文件推展到目標組中所有裝置的特定日期和時間。維護時段是週期性執行個體，其頻率為每日、每週、每月或在初始任務或任務範本建立期間選取的自訂日期與時間。只有連續任務才能排定在維護時段期間執行推展。

任務排程是專屬於您特定任務的功能。個別任務執行無法進行排程。如需詳細資訊，請參閱任務推展、排程和中止組態。

中止

您可以建立一組條件，能夠在滿足指定條件時取消推展。如需詳細資訊，請參閱任務推展、排程和中止組態。

逾時

當任務部署長時間非預期卡在 IN_PROGRESS 狀態時，任務逾時即會通知您。計時器有兩種類型：進行中計時器和步驟計時器。當任務處於 IN_PROGRESS 狀態時，您可以監控和追蹤任務部署的進度。

推展和中止組態與任務相關，逾時組態則與任務部署相關。如需詳細資訊，請參閱任務執行逾時和重試組態。

重試

任務重試可以在任務失敗、逾時，或兩者同時發生時重試任務執行。您最多可以對任務執行嘗試 10 次重試。您可以監控和追蹤重試嘗試的進度，以及任務執行是否成功。

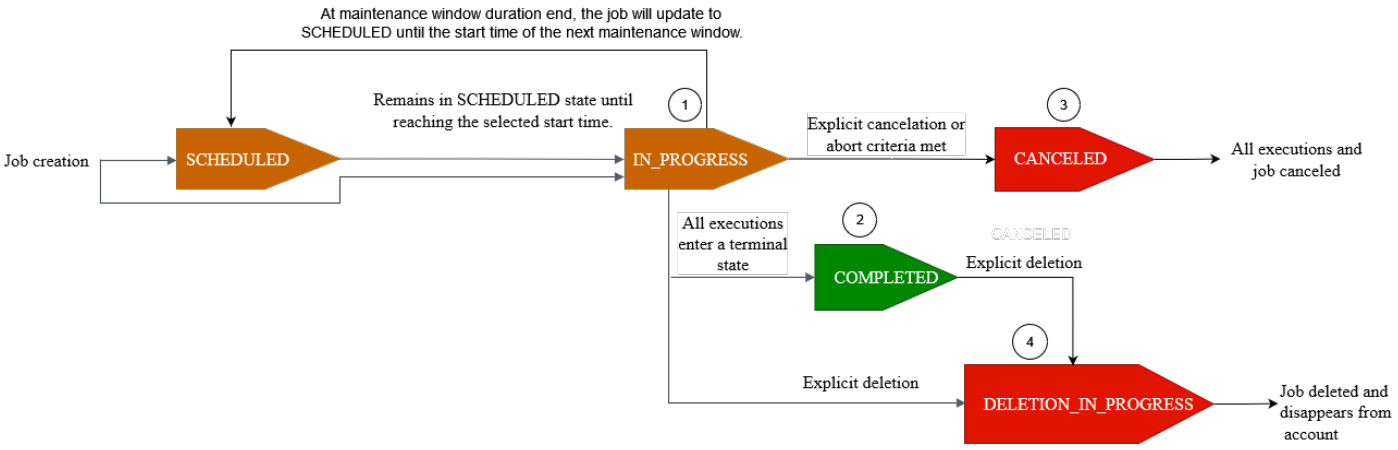
推展和中止組態與任務相關，逾時和重試組態則與任務執行相關。如需詳細資訊，請參閱任務執行逾時和重試組態。

任務和任務執行狀態

下列各節說明 AWS IoT 任務的生命週期和任務執行的生命週期。

任務狀態

下圖顯示 AWS IoT 任務的不同狀態。



您使用 任務建立 AWS IoT 的任務可以處於下列其中一種狀態：

- 已排程

  在使用 AWS IoT 主控台、CreateJob API 或 CreateJobTemplate API 建立初始任務或任務範本期間，您可以在 AWS IoT 主控台或 CreateJob API 或 CreateJobTemplate API SchedulingConfig 中選取選用的排程組態。當您啟動包含特定 startTime、endTime 和 endBehavior 的排程任務時，任務狀態會更新為 SCHEDULED。當任務達到您選取的 startTime 或下一個維護時段的 startTime (如果您選取了在維護時段推展任務)，狀態會從 SCHEDULED 更新為 IN_PROGRESS，並開始將任務文件推展至目標群組中的所有裝置。

- IN_PROGRESS

  當您使用 AWS IoT 主控台或 CreateJob API 建立任務時，任務狀態會更新為 IN_PROGRESS。在建立任務期間, AWS IoT 任務開始將任務執行推展到目標群組中的裝置。在所有任務執行推展完畢之後， AWS IoT 任務會等待裝置完成遠端動作。

  如需套用至進行中任務的並行和限制的相關資訊，請參閱 AWS IoT 任務限制。

  > **Note**
  >
  > 當 IN_PROGRESS 任務到達目前維護時段結束時，任務文件的展開就會停止。任務將更新為 SCHEDULED，直到下一個維護時段的 startTime。

- COMPLETED (已完成)

  會以下列其中一種方式處理連續任務：

  - 對於未選取選用排程組態的連續任務，它會始終在進行中，持續為新增至目標群組的任何新裝置執行。它永遠不會達到 COMPLETED 的狀態。
  - 對於具有選用排程組態的連續任務，則會發生下列情況：
    - 如果有提供 endTime，則連續任務將在超過 endTime 且所有任務執行都達到結束狀態時進入 COMPLETED 狀態。
    - 如果選用排程組態中未提供 endTime，則連續任務會繼續執行任務文件推展。

  對於快照任務，任務狀態會在所有任務執行進入終端狀態 (例如 SUCCEEDED、FAILED、TIMED_OUT、REMOVED 或CANCELED) 時變更為 COMPLETED。

- CANCELED

當您使用 AWS IoT 主控台、CancelJob API 或 取消任務時 任務中止組態，任務狀態會變更為 CANCELED。在任務取消期間， AWS IoT 任務會開始取消先前建立的任務執行。

如需套用至取消中任務的並行和限制的相關資訊，請參閱 AWS IoT 任務限制。

- DELETION_IN_PROGRESS

  當您使用 AWS IoT 主控台或 DeleteJob API 刪除任務時，任務狀態會變更為 DELETION_IN_PROGRESS。在刪除任務期間， AWS IoT 任務會開始刪除先前建立的任務執行。刪除所有任務執行後，任務會從 AWS 您的帳戶中消失。

任務執行狀態

下表顯示 AWS IoT 任務執行的不同狀態，以及狀態變更是由裝置或 AWS IoT 任務啟動。

任務執行狀態與來源

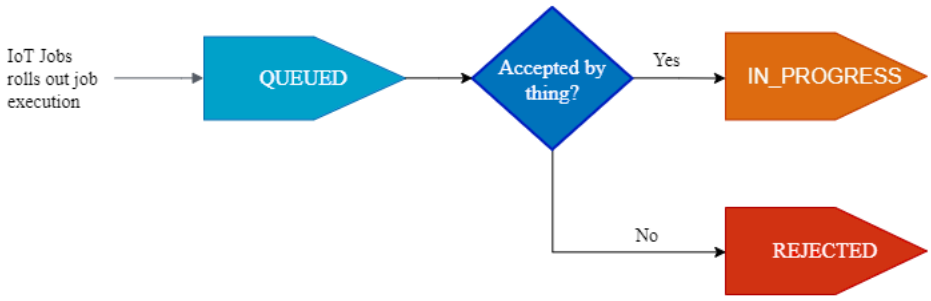
任務執行狀態	由裝置起始？	由 AWS IoT 任務啟動？	結束狀態？	可否重試？
QUEUED	否	是	否	不適用
IN_PROGRESS	是	否	否	不適用
SUCCEEDED	是	否	是	不適用
FAILED	是	否	是	是
TIMED_OUT	否	是	是	是
REJECTED	是	否	是	否
REMOVED	否	是	是	否
CANCELED	否	是	是	否

下一節將詳細說明當您使用 任務建立任務時，所推出 AWS IoT 之任務執行的狀態。

- QUEUED

當 AWS IoT 任務為目標裝置推展任務執行時，任務執行狀態會設為 QUEUED。任務執行會維持在 QUEUED 狀態，直到：

- 您的裝置接收任務執行，並叫用任務 API 操作，將狀態報告為 IN_PROGRESS。
- 您取消任務或任務執行，或在符合您指定的中止條件時，狀態會變更為 CANCELED。
- 您的裝置已從目標群組中移除，且狀態變更為 REMOVED。



- IN_PROGRESS

  如果您的 IoT 裝置訂閱預留 任務主題$notify和 $notify-next，而且您的裝置調用狀態為 的 StartNextPendingJobExecution API 或 UpdateJobExecution APIIN_PROGRESS， AWS IoT Jobs 會將任務執行狀態設定為 IN_PROGRESS。

  此 UpdateJobExecution API 可多次叫用，且狀態為 IN_PROGRESS。您可以使用 statusDetails 物件來指定關於執行步驟的其他詳細資訊。

  > **Note**
  >
  > 如果您為每個裝置建立多個任務， AWS IoT Jobs 和 MQTT 通訊協定不保證交付順序。

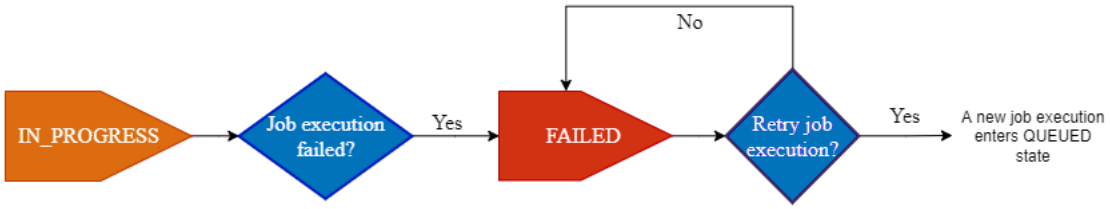
- SUCCEEDED (成功)

  當您的裝置成功完成遠端操作時，裝置必須調用狀態為 的 UpdateJobExecution APISUCCEEDED，以指出任務執行成功。 AWS IoT 任務接著會更新並傳回任務執行狀態為 SUCCEEDED。



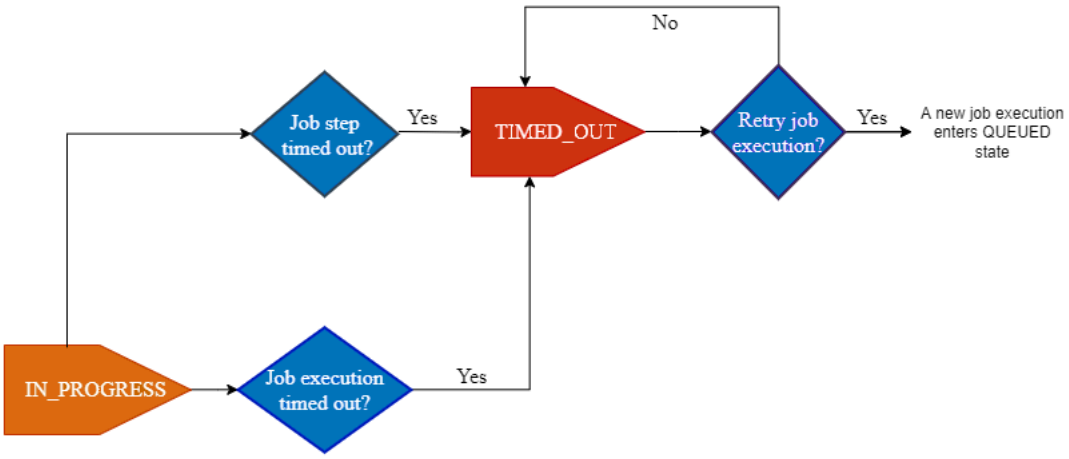
- 失敗

當您的裝置無法完成遠端操作時，裝置必須調用狀態為 的 UpdateJobExecution APIFailed，以指出任務執行失敗。 AWS IoT 任務接著會更新並傳回任務執行狀態為 Failed。您可以使用 任務執行重試組態 未裝置重試此任務執行。



- TIMED_OUT

  當您的裝置在狀態為 時無法完成任務步驟IN_PROGRESS，或無法在進行中計時器的逾時期間內完成遠端操作時， AWS IoT Jobs 會將任務執行狀態設定為 TIMED_OUT。您也可以為進行中任務的每個任務步驟設定一個步驟計時器，且僅適用於任務執行。此進行中計時器期間使用 任務執行逾時組態 的 inProgressTimeoutInMinutes 屬性指定。您可以使用 任務執行重試組態 未裝置重試此任務執行。



- REJECTED

  當您的裝置收到無效或不相容的請求時，裝置必須調用狀態為 REJECTED. AWS IoT Jobs 的 UpdateJobExecution API，然後更新並傳回任務執行狀態為 REJECTED。

- REMOVED

  當您的裝置不再是任務執行的有效目標時，例如當它與動態物件群組分離時， AWS IoT 任務會將任務執行狀態設置為 REMOVED。您可以將物件重新附加到目標群組，然後重新啟動裝置的任務執行。

- CANCELED

當您使用主控台或 或 CancelJob CancelJobExecution API 取消任務或取消任務執行，或 任務中止組態滿足使用 指定的中止條件時， AWS IoT 任務會取消任務，並將任務執行狀態設定為 CANCELED。

管理任務

使用任務通知裝置軟體或韌體更新。您可以使用 AWS IoT 主控台、任務管理和控制 API 操作、AWS Command Line Interface 或 AWS SDK 來建立和管理任務。

任務的程式碼簽署

傳送程式碼至裝置時，若要讓裝置偵測程式碼在傳輸過程中是否遭到修改，我們建議您使用 AWS CLI 對程式碼檔案進行簽署。如需相關說明，請參閱使用 AWS CLI建立和管理任務。

如需詳細資訊，請參閱什麼是程式碼簽署 AWS IoT？。

任務文件

建立任務之前，您必須建立任務文件。如果您使用的是 的程式碼簽署 AWS IoT，則必須將任務文件上傳至版本控制的 Amazon S3 儲存貯體。如需建立 Amazon S3 儲存貯體並將檔案上傳至其中的詳細資訊，請參閱《Amazon S3 入門指南》中的 Amazon Simple Storage Service 入門。

> **Tip**
>
> 如需任務文件範例，請參閱適用於 JavaScript 的 AWS IoT SDK 中的 job-agent.js 範例。

預先簽章的 URL

您的任務文件可以包含指向程式碼檔 (或其他檔案) 的預先簽章 Amazon S3 URL。預先簽章的 Amazon S3 URL 僅在有限的時間內有效，因此在裝置請求任務文件之後才會產生。由於建立任務文件時未建立預先簽章的 URL，所以請在任務文件中使用預留位置 URL。預留位置 URL 如下所示：

```
${aws:iot:s3-presigned-url-v2:https://
s3.region.amazonaws.com/<bucket>/<code file>}
```

其中：

- *bucket* 為含有程式碼檔案的 Amazon S3 儲存貯體。
- *code file* 為程式碼檔案的 Amazon S3 金鑰。

當裝置請求任務文件時，AWS IoT 會產生預先簽章的 URL，並將預留位置 URL 取代為預先簽章的 URL。您的任務文件接著會傳送到裝置。

IAM 角色授予從 S3 下載檔案的許可

當建立一個使用預先簽章 Amazon S3 URL 的任務時，您必須提供 IAM 角色。此角色必須授予自資料或更新儲存所在 Amazon S3 儲存貯體中下載檔案的許可。此角色也必須授與 AWS IoT 許可來擔任此角色。

您可指定預先簽章 URL 的選擇性逾時值。如需詳細資訊，請參閱 CreateJob。

授予 AWS IoT 任務擔任您角色的許可

1. 前往 IAM 主控台的角色中心，然後選擇您的角色。
2. 在 Trust Relationships (信任關係) 索引標籤中，選擇 Edit Trust Relationship (編輯信任關係)，並將政策文件取代為以下 JSON。選擇 Update Trust Policy (更新信任政策)。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iot.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 建議您新增全域條件內容金鑰 aws:SourceArn 和 aws:SourceAccount 至政策，保護自己免受混淆代理人問題的困擾。

> **Important**
>
> aws:SourceArn 必須符合以下格式：arn:aws:iot:*region*:*account-id*:*。請確定*##*與您的 AWS IoT 區域相符，且*## *ID* 與您的客戶帳戶 ID 相符。如需詳細資訊，請參閱預防跨服務混淆代理人。

```
{
  "Effect": "Allow",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service":
          "iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole",
      "Condition": {
         "StringEquals": {
            "aws:SourceAccount": "123456789012"
         },
         "ArnLike": {
              "aws:SourceArn": "arn:aws:iot:*:123456789012:job/*"
         }
       }
     }
   ]
}
```

4. 如果您的任務使用本身為 Amazon S3 物件的任務文件，請選擇許可並使用下列 JSON。這會新增一個政策，授予從 Amazon S3 儲存貯體下載檔案的許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
```

```
    ]
}
```

檔案上傳的預先簽章 URL

如果您的裝置需要在任務部署期間將檔案上傳至 Amazon S3 儲存貯體，則可以在任務文件中包含下列預先簽章的 URL 預留位置：

```
${aws:iot:s3-presigned-url-upload:https://s3.region.amazonaws.com/<bucket>/<key>}
```

您可以在任務文件中的檔案上傳預留位置 URL 中的 key 屬性內${jobId}，使用每個 ${thingName}、 和 中最多兩個${executionNumber}預留關鍵字。建立任務執行時，將剖析和取代代表 key 屬性中預留關鍵字的本機預留位置。使用具有每個裝置特定預留關鍵字的本機預留位置，可確保從裝置上傳的每個檔案都專屬於該裝置，而不會被相同任務部署鎖定之其他裝置上傳的類似檔案覆寫。如需預先簽章的 URL 預留位置中本機預留位置的故障診斷資訊，以便在任務部署期間上傳檔案，請參閱 對錯誤訊息進行一般故障診斷。

> **Note**
>
> Amazon S3 儲存貯體名稱不能包含代表所上傳檔案預留關鍵字的本機預留位置。本機預留位置必須位於 key 屬性中。

此預先簽章的 URL 預留位置會在裝置收到時轉換為任務文件中的 Amazon S3 預先簽章上傳 URL。您的裝置將使用此功能將檔案上傳至目的地 Amazon S3 儲存貯體。

> **Note**
>
> 當上述預留位置 URL 中未提供 Amazon S3 儲存貯體和金鑰時， AWS IoT 任務會使用 ${thingName}、 ${jobId}和 中的每個裝置最多兩個自動產生金鑰${executionNumber}。

使用 Amazon S3 版本控制的預先簽章 URL

保護存放在 Amazon S3 儲存貯體中檔案的完整性對於確保使用該檔案對裝置機群進行安全任務部署至關重要。透過使用 Amazon S3 版本控制，您可以為存放在 Amazon S3 儲存貯體中的檔案的每個變體

新增版本識別符，以追蹤檔案的每個版本。這可讓您深入了解使用 AWS IoT Jobs 部署到裝置機群的檔案版本。如需使用版本控制之 Amazon S3 儲存貯體的詳細資訊，請參閱在 Amazon S3 儲存貯體中使用版本控制。

如果檔案存放在 Amazon S3 中，且任務文件包含預先簽章的 URL 預留位置， AWS IoT Jobs 將使用存放在 Amazon S3 儲存貯體中的檔案的 Amazon S3 儲存貯體、儲存貯體金鑰和版本，在任務文件中產生預先簽章的 URLAmazon S3。任務文件中產生的此預先簽章 URL 將取代任務文件中最初的預先簽章 URL 預留位置。如果您更新存放在 Amazon S3 儲存貯體中的檔案，`versionId`則會建立新的檔案版本和後續版本，以表示所做的更新，並提供在未來任務部署中以該特定檔案為目標的能力。

請參閱下列範例，了解使用 在任務文件中尋找 Amazon S3 預先簽章 URLs 之前和期間： `versionId`

Amazon S3 預先簽章的 URL 預留位置 （在任務部署之前）

```
//Virtual-hosted style URL
${aws:iot:s3-presigned-url-v2:https://bucket-name.s3.region-code.amazonaws.com/key-name%3FversionId%3Dversion-id}

//Path-style URL
${aws:iot:s3-presigned-url-v2:https://s3.region-code.amazonaws.com/bucket-name/key-name%3FversionId%3Dversion-id}
```

Amazon S3 預先簽章 URL （在任務部署期間）

```
//Virtual-hosted style URL
${aws:iot:s3-presigned-url-v2:https://sample-bucket-name.s3.us-west-2.amazonaws.com/sample-code-file.png%3FversionId%3Dversion1}

//Path-style
${aws:iot:s3-presigned-url-v2:https://s3.us-west-2.amazonaws.com/sample-bucket-name/sample-code-file.png%3FversionId%3Dversion1}
```

如需 Amazon S3 虛擬託管和路徑樣式物件 URLs的詳細資訊，請參閱Virtual-hosted-style請求和路徑樣式請求。

> **Note**
>
> 如果您想要附加`versionId`到 Amazon S3 預先簽章的 URL，則必須符合支援的 URL 編碼 AWS SDK for Java 2.x。如需詳細資訊，請參閱剖析 Amazon S3 URIs 從版本 1 到版本 2 的變更。

Amazon S3 預先簽章的 URL 預留位置版本差異

下列清單概述 Amazon S3 預先簽章 URL 預留位置 `${aws:iot:s3-presigned-url-v1`（第 1 版）和 `${aws:iot:s3-presigned-url-v2`（第 2 版）之間的差異：

- Amazon S3 預先簽章的 URL 預留位置`${aws:iot:s3-presigned-url-v1`不支援 `version-id`。
- Amazon S3 預先簽章的 URL 預留位置`${aws:iot:s3-presigned-url-v1`會收到未編碼的 Amazon S3 URL。Amazon S3 預先簽章 URL 預留位置`${aws:iot:s3-presigned-url-v2`需要對 Amazon S3 URL 進行編碼，以符合 Amazon S3 SDK 標準。

主題

- 使用 建立和管理任務 AWS 管理主控台
- 使用 建立和管理任務 AWS CLI

使用 建立和管理任務 AWS 管理主控台

本節說明如何從 AWS IoT 主控台建立和管理任務。建立任務之後，您可以在詳細資訊頁面上檢視任務的相關資訊，並管理任務。

> **Note**
>
> 如果您想要為 AWS IoT 任務執行程式碼簽署，請使用 AWS CLI。如需詳細資訊，請參閱使用建立和管理任務 AWS CLI。

主題

- 使用 建立管理任務 AWS 管理主控台
- 使用 檢視和管理任務 AWS 管理主控台

使用 建立管理任務 AWS 管理主控台

若要建立任務，請登入 AWS IoT 主控台，然後前往遠端動作區段中的任務中樞。然後，執行下列步驟。

1. 在任務頁面上的任務對話方塊中，選擇建立任務。

2. 視您使用的裝置而定，您可以建立自訂任務、FreeRTOS OTA 更新任務或 AWS IoT Greengrass 任務。在此範例中，請選擇 Create a custom job (建立自訂任務)。選擇下一步。
3. 在 Custom job properties(自訂任務屬性) 頁面的 Job properties (任務屬性) 對話方塊中，輸入下列各欄資訊：

   - Name (姓名)：輸入任務的唯一英數名稱。
   - Description - optional (說明 - 選擇性)：輸入有關任務的選擇性說明。
   - 標籤 – 選用：

   > **Note**
   >
   > 建議您不要在任務 ID 和描述中使用個人識別資訊。

   選擇下一步。
4. 在 Job targets (任務 目標) 對話方塊的 File configuration (檔案組態) 頁面上，選取要執行此任務的Things (物件) 或 Thing groups (物件群組)。

   在 Job document (任務文件) 對話方塊中選取下列其中一個選項：

   - From file (從檔案)：您先前上傳至 Amazon S3 儲存貯體的 JSON 任務檔案。
     - 程式碼簽署

       在 Amazon S3 URL 中的任務文件中，`${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}` 必須做為預留位置，直到使用程式碼簽署設定檔將其取代為已簽署的程式碼檔案路徑為止。新簽署的程式碼檔案一開始會出現在 Amazon S3 來源儲存貯體的 `SignedImages` 資料夾中。系統會建立包含 `Codesigned_` 字首的新任務文件，其中包含已簽署的程式碼檔案路徑取代程式碼簽署預留位置，且放置在 Amazon S3 URL 中以建立新任務。
     - 預先簽署資源 URL

       在預先簽署角色下拉式清單中，選擇您在預先簽署 URL 中建立的 IAM 角色。對於從 Amazon S3 下載物件的裝置，對位於 Amazon S3 中的物件使用 `${aws:iot:s3-presigned-url:` 預先簽署 URL 是最佳安全實務。

       如果您想將預先簽署的 URL 用於程式碼簽署預留位置，請使用下列範例範本：

```
${aws:iot:s3-presigned-url:${aws:iot:code-sign-signature:<S3 URL>}
```

- From template (從範本)：包含任務文件和任務組態的任務範本。任務範本可以是您建立的自訂任務範本或 AWS 受管範本。

  如果您要建立執行常用遠端動作的任務，例如重新啟動裝置，您可以使用 AWS 受管範本。這些範本已預先設定，可供使用。如需詳細資訊，請參閱建立自訂任務範本及從受管範本建立自訂任務範本。

5. 在 Job configuration (任務組態) 對話方塊的 Job configuration (任務組態) 頁面上，選取下列其中一種任務類型：

   - 快照任務：快照任務在目標裝置和群組上完成執行後即完成。
   - 連續任務：連續任務適用於物件群組，並在稍後新增至所指定目標群組的任何裝置上執行。

6. 在 Additional configurations - optional (其他組態 - 選擇性) 對話方塊中，檢閱下列選擇性任務組態，並視需要進行選擇：

   - 推展組態
   - Scheduling configuration (排程組態)
   - 任務執行逾時組態
   - Job executions retry configuration - new (任務執行重試組態 - 新增)
   - 中止組態

   如需有關任務組態的其他資訊，請參閱下列各節：

   - 任務推展、排程和中止組態
   - 任務執行逾時和重試組態

   檢閱所有任務選項，然後選擇 Submit (提交) 以建立任務。

使用 檢視和管理任務 AWS 管理主控台

建立任務後，主控台會產生 JSON 簽章，並將其放置在您的任務文件中。您可以使用 AWS IoT 主控台來檢視狀態，或取消、刪除任務。

如果您選擇您建立的任務，您可以找到：

- 一般任務詳細資訊，例如任務名稱、描述、類型、建立時間、上次更新時間，以及預估的開始時間。
- 您指定的任何任務組態及其狀態。
- 工作文件。
- 任務執行和您指定的任何選用標籤。

若要管理任務，請前往 主控台的任務中樞，然後選擇是否要編輯、刪除或取消任務。

使用 建立和管理任務 AWS CLI

本節說明如何建立和管理任務。

建立任務

若要建立 AWS IoT 任務，請使用 CreateJob命令。任務會排入所指定目標 (物件或物件群組) 的執行佇列。若要建立 AWS IoT 任務，您需要可以包含在請求內文中的任務文件，或做為 Amazon S3 文件的連結。如果任務包含使用預先簽章的 Amazon S3 URLs 下載檔案，您需要 IAM 角色 Amazon Resource Name (ARN)，其具有下載檔案的許可，並授予 AWS IoT Jobs 服務擔任該角色的許可。

如需使用 API 命令或 輸入日期和時間時語法的詳細資訊 AWS CLI，請參閱時間戳記。

使用任務進行程式碼簽署

如果您使用 的程式碼簽署 AWS IoT，則必須啟動程式碼簽署任務，並在任務文件中包含輸出。這會取代任務文件中的程式碼簽署簽章預留位置，在使用程式碼簽署設定檔將其替換為已簽署程式碼檔案路徑之前，這個預留位置是必要的。程式碼簽署簽章預留位置將如下所示：

```
    ${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}
```

使用 start-signing-job 命令來建立程式碼簽署任務。`start-signing-job` 會傳回任務 ID。若要取得存放簽章的 Amazon S3 位置，請使用 describe-signing-job 命令。然後，您可以從 Amazon S3 下載簽章。如需程式碼簽署任務的詳細資訊，請參閱適用於 AWS IoT的程式碼簽署。

您的任務文件必須包含程式碼檔的預先簽章 URL 預留位置，以及使用在 start-signing-job 命令放置在 Amazon S3 儲存貯體的 JSON 簽章輸出：

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/
image}",
```

```
}
```

使用任務文件建立任務

下列命令顯示如何使用 Amazon S3 儲存貯體 (*jobBucket*) 中存放的任務文件 (*job-document.json*) 來建立任務，以及建立有權從 Amazon S3 下載檔案的角色 (*S3DownloadRole*)。

```
aws iot create-job  \
      --job-id 010  \
      --targets arn:aws:iot:us-east-1:123456789012:thing/thingOne  \
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute
\": 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings
\": 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

任務在 *thingOne* 執行。

選用的 `timeout-config` 參數，會指定每個裝置必須完成其任務執行的時間。任務執行狀態設定為 `IN_PROGRESS` 時，計時器即會開始。在時間過期之前，如果任務執行狀態未設定為其他終止狀態，就會將其設定為 `TIMED_OUT`。

進行中的計時器無法更新，並會套用到任務的所有任務執行。每當任務執行保持 `IN_PROGRESS` 狀態超過此間隔時，就會失敗並切換到終端機`TIMED_OUT`狀態。 AWS IoT 也會發佈 MQTT 通知。

如需建立任務推展和中止組態的詳細資訊，請參閱任務推展和中止組態。

> **Note**
>
> 指定為 Amazon S3 檔案的任務文件會在您建立任務時擷取。如果您在建立任務文件後變更了作為任務文件來源的 Amazon S3 檔案的內容，則傳送到任務目標的內容不會變更。

更新任務

若要更新任務，請使用 UpdateJob 命令。您可以更新任務的 `description`、`presignedUrlConfig`、`jobExecutionsRolloutConfig`、`abortConfig` 和 `timeoutConfig` 欄位。

```
aws iot update-job  \
  --job-id 010  \
  --description "updated description" \
  --timeout-config inProgressTimeoutInMinutes=100 \
  --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}, \"maximumPerMinute\": 1000}}" \
  --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
  --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

如需詳細資訊，請參閱任務推展和中止組態。

取消任務

若要取消任務，請使用 CancelJob 命令。取消任務 AWS IoT 會停止推出任務的任何新任務執行。它也會取消處於 QUEUED 狀態的任何任務執行。 會 AWS IoT 保留處於結束狀態的任何任務執行，因為裝置已完成任務。如果任務執行的狀態為 IN_PROGRESS，則該任務執行也將維持不變，除非您使用選用的 `--force` 參數。

以下命令說明如何使用 ID 010 來取消任務。

```
aws iot cancel-job --job-id 010
```

該命令會顯示下列輸出：

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

當您取消任務時，狀態為 QUEUED 的任務執行將會遭到取消。處於 IN_PROGRESS 狀態的任務執行會遭到取消，前提是您指定了可選 --force 參數。處於結束狀態的任務執行不會取消。

> **Warning**
>
> 取消狀態為 IN_PROGRESS (透過設定 --force 參數) 的任務將取消任何進行中的任務執行，並導致正在執行任務的裝置無法更新任務執行狀態。請謹慎執行，並確認每個裝置所執行的已取消任務可以復原至有效狀態。

已取消任務或其其中一個任務執行的狀態最終是一致的。 會停止盡快將該任務的新任務執行和QUEUED任務執行 AWS IoT 排程到裝置。將任務執行的狀態變更為 CANCELED (取消) 可能會需要一些時間，視裝置數量及其他因素而定。

如果任務由於符合 AbortConfig 物件定義的條件而被取消，則服務新增自動填入 comment 和 reasonCode 欄位的值。當任務取消是由使用者驅動時，您可以建立自己的 reasonCode 值。

取消任務執行

若要取消裝置上的任務執行，請使用 CancelJobExecution 命令。這麼做會取消狀態為 QUEUED 的任務執行。如果想取消進行中的任務執行，您必須使用 --force 參數。

以下命令說明如何從 myThing 上執行的任務 010 取消任務執行。

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

此命令不會顯示輸出。

系統會取消狀態為 QUEUED 的任務執行。處於 IN_PROGRESS 狀態的任務執行會遭到取消，前提是您指定了可選 --force 參數。處於結束狀態的任務執行無法被取消。

> **Warning**
>
> 取消狀態為 IN_PROGRESS 的任務執行時，將導致裝置無法更新任務執行狀態。請謹慎使用並確保裝置可以還原為有效狀態。

如果任務執行處於終端狀態，或如果任務執行處於 IN_PROGRESS 狀態且 --force 參數未設定為 true，此命令會導致 InvalidStateTransitionException。

取消的任務執行之狀態最終會保持一致。將任務執行的狀態變更為 CANCELED 可能會需要一些時間，依不同因素而定。

刪除任務

若要刪除任務及其任務執行，請使用 DeleteJob 命令。依預設，您只能刪除處於結束狀態 (SUCCEEDED 或 CANCELED) 的任務。否則會發生例外狀況。不過，只在 IN_PROGRESS 參數設定為 `force` 時，您才能刪除處於 `true` 狀態的任務。

若要刪除任務，請執行下列命令：

```
aws iot delete-job --job-id 010 --force|--no-force
```

此命令不會顯示輸出。

> **Warning**
>
> 刪除狀態為 IN_PROGRESS 的任務時，正在部署任務的裝置將無法存取任務資訊或更新任務執行狀態。請謹慎執行，並確保每個裝置所部署的已刪除任務可以復原至有效狀態。

刪除任務可能會需要一些時間，根據為任務建立的任務執行數量與其他因素而定。在刪除任務期間，任務狀態會顯示為「DELETION_IN_PROGRESS」。如果嘗試刪除或取消狀態已為 DELETION_IN_PROGRESS 的任務將會導致錯誤。

只有 10 個任務可以同時具有 DELETION_IN_PROGRESS 狀態。否則會發生 `LimitExceededException`。

取得任務文件

若要從任務中擷取任務文件，請使用 GetJobDocument 命令。任務文件為由裝置執行遠端操作的描述。

若要取得取得任務文件，請執行下列命令：

```
aws iot get-job-document --job-id 010
```

此命令會傳回指定任務的任務文件：

```
{
    "document": "{\n\t\"operation\":\"install\",\n\t\"url\":\"http://amazon.com/
firmWareUpate-01\",\n\t\"data\":\"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
amzn-s3-demo-bucket/datafile}\"\n}"
}
```

> **Note**
>
> 在使用此命令擷取任務文件時，預留位置 URL 並不會被預先簽章的 Amazon S3 URL 取代。當裝置呼叫 GetPendingJobExecutions API 操作時，預留位置 URL 才會被任務文件中的預先簽章 Amazon S3 URL 取代。

列出任務

若要取得 中所有任務的清單 AWS 帳戶，請使用 ListJobs命令。任務資料和任務執行資料會保留一段有限的時間。執行下列命令來列出 中的所有任務 AWS 帳戶：

```
aws iot list-jobs
```

此命令會傳回您帳戶中所有任務，並依任務狀態排序：

```
{
    "jobs": [
        {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486687079.743,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/013",
            "createdAt": 1486687079.743,
            "targetSelection": "SNAPSHOT",
            "jobId": "013"
        },
        {
            "status": "SUCCEEDED",
            "lastUpdatedAt": 1486685868.444,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/012",
            "createdAt": 1486685868.444,
            "completedAt": 148668789.690,
            "targetSelection": "SNAPSHOT",
            "jobId": "012"
```

```
        },
        {
            "status": "CANCELED",
            "lastUpdatedAt": 1486678850.575,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/011",
            "createdAt": 1486678850.575,
            "targetSelection": "SNAPSHOT",
            "jobId": "011"
        }
    ]
}
```

描述任務

若要取得任務的狀態，請執行 DescribeJob 命令。以下命令說明如何描述任務：

```
$ aws iot describe-job --job-id 010
```

此命令會傳回指定任務的狀態。例如：

```
{
    "documentSource": "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-
document.json",
    "job": {
        "status": "IN_PROGRESS",
        "jobArn": "arn:aws:iot:us-east-1:123456789012:job/010",
        "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/myThing"
        ],
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfFailedThings": 0,
            "numberOfInProgressThings": 0,
            "numberOfQueuedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfTimedOutThings": 0,
            "processingTargets": [
                arn:aws:iot:us-east-1:123456789012:thing/thingOne,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupOne,
                arn:aws:iot:us-east-1:123456789012:thing/thingTwo,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupTwo
```

```
            ]
        },
        "presignedUrlConfig": {
            "expiresInSec": 60,
            "roleArn": "arn:aws:iam::123456789012:role/S3DownloadRole"
        },
        "jobId": "010",
        "lastUpdatedAt": 1486593195.006,
        "createdAt": 1486593195.006,
        "targetSelection": "SNAPSHOT",
        "jobExecutionsRolloutConfig": {
            "exponentialRate": {
                "baseRatePerMinute": integer,
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                    "numberOfNotifiedThings": integer, // Set one or the other
                    "numberOfSucceededThings": integer // of these two values.
                },
            "maximumPerMinute": integer
         }
        },
        "abortConfig": {
            "criteriaList": [
                {
                    "action": "string",
                    "failureType": "string",
                    "minNumberOfExecutedThings": integer,
                    "thresholdPercentage": integer
                }
            ]
        },
        "timeoutConfig": {
           "inProgressTimeoutInMinutes": number
          }
    }
}
```

列出任務的執行

任務執行物件代表在特定裝置上運作的任務。執行 ListJobExecutionsForJob 命令以列出任務中的所有任務執行。以下顯示如何列出任務執行：

```
aws iot list-job-executions-for-job --job-id 010
```

此命令會傳回一份任務執行清單：

```
{
    "executionSummaries": [
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 1234567890
        }
    },
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingTwo",
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486593345.659,
            "queuedAt": 1486593196.378,
            "startedAt": 1486593345.659,
            "executionNumber": 4567890123
        }
    }
    ]
}
```

列出物件的任務執行

執行 ListJobExecutionsForThing 命令以列出在物件上運作的所有任務執行。以下顯示如何列出物件的任務執行：

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

此命令會傳回一份清單，列出特定物件上正在運作或已經運作的任務執行：

```
{
    "executionSummaries": [
    {
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486687082.071,
            "queuedAt": 1486687082.071,
```

```
            "executionNumber": 9876543210
        },
        "jobId": "013"
    },
    {
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "startAt": 1486685870.729,
            "lastUpdatedAt": 1486685870.729,
            "queuedAt": 1486685870.729,
            "executionNumber": 1357924680
        },
        "jobId": "012"
    },
    {
        "jobExecutionSummary": {
            "status": "SUCCEEDED",
            "startAt": 1486678853.415,
            "lastUpdatedAt": 1486678853.415,
            "queuedAt": 1486678853.415,
            "executionNumber": 4357680912
        },
        "jobId": "011"
    },
    {
        "jobExecutionSummary": {
            "status": "CANCELED",
            "startAt": 1486593196.378,
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 2143174250
        },
        "jobId": "010"
    }
    ]
}
```

描述任務執行

執行 DescribeJobExecution 命令以取得任務執行的狀態。您必須指定任務 ID 及物件名稱，或者可選擇是否指定執行編號來識別任務執行。以下顯示如何描述任務執行：

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

此命令會傳回 JobExecution。例如：

```
{
    "execution": {
        "jobId": "017",
        "executionNumber": 4516820379,
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "versionNumber": 123,
        "createdAt": 1489084805.285,
        "lastUpdatedAt": 1489086279.937,
        "startedAt": 1489086279.937,
        "status": "IN_PROGRESS",
        "approximateSecondsBeforeTimedOut": 100,
        "statusDetails": {
            "status": "IN_PROGRESS",
            "detailsMap": {
                "percentComplete": "10"
            }
        }
    }
}
```

刪除任務執行

執行 DeleteJobExecution 命令以刪除任務執行。必須指定任務 ID (物件名稱) 和執行編號來識別任務執行。以下顯示如何刪除任務執行：

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

此命令不會顯示輸出。

根據預設，任務執行的狀態必須為 QUEUED 或處於終端狀態 (SUCCEEDED、FAILED、REJECTED、TIMED_OUT、REMOVED 或 CANCELED)。否則會發生錯誤。若要刪除狀態為 IN_PROGRESS 的任務執行，您可以將 force 參數設為 true。

> **Warning**
>
> 當您刪除狀態為 IN_PROGRESS 的任務執行時，正在執行任務的裝置將無法存取任務資訊或更新任務執行狀態。請謹慎使用並確保裝置可以還原為有效狀態。

任務範本

使用任務範本預先設定任務，以便將它們部署至多組目標裝置。若要將經常執行的遠端動作 (例如重新啟動或安裝應用程式) 部署至裝置，您可以使用範本來定義標準組態。若要執行部署安全性修補程式和錯誤修正等操作，您可以從現有任務建立範本。

建立任務範本時，請指定下列其他組態和資源。

- 任務屬性
- 任務文件和目標
- 推展、排程和取消條件
- 逾時和重試條件

自訂和 AWS 受管範本

根據您要執行的遠端動作，您可以建立自訂任務範本或使用 AWS 受管範本。使用自訂任務範本來提供您自己的自訂任務文件，並建立可重複使用的任務以部署到您的裝置。 AWS 受管範本是由 AWS IoT 任務為常用動作提供的任務範本。這些範本具有適用於某些遠端動作的預先定義任務文件，無需您建立自己的任務文件。受管範本會協助您建立可重複使用的任務，以便更快速地啟動至裝置。

主題

- 使用 AWS 受管範本部署常見的遠端操作
- 建立自訂任務範本

使用 AWS 受管範本部署常見的遠端操作

AWS 受管範本是由 提供的任務範本 AWS。它們用於經常執行的遠端動作，例如重新啟動、下載檔案或在裝置上安裝應用程式。這些範本具有適用於每個遠端動作的預先定義任務文件，無需您建立自己的任務文件。

您可以從一組預先定義的組態中進行選擇，並使用這些範本建立任務，而無需撰寫任何其他程式碼。使用受管範本，您可以檢視部署至機群的任務文件。您可以使用這些範本來建立任務，並建立可重複使用於遠端操作的自訂任務範本。

受管範本包含哪些內容？

每個 AWS 受管範本都包含：

- 在任務文件中執行命令的環境。
- 指定操作名稱及其參數的任務文件。例如，如果使用 Download file (下載檔案) 範本，操作名稱為 Download file 並且參數可以是：
  - 您要下載至裝置的檔案 URL。這可以是網際網路資源，也可以是公有或預先簽章的 Amazon Simple Storage Service (Amazon S3) URL。
  - 裝置上用來存放下載檔案的本機檔案路徑。

  如需有關任務文件及其參數的詳細資訊，請參閱 受管範本遠端動作和任務文件。

先決條件

若要讓裝置執行受管範本任務文件指定的遠端操作，必須：

- 在裝置上安裝特定軟體

  使用您自己的裝置軟體和任務處理常式，或 AWS IoT 裝置用戶端。根據您的業務案例，您也可以同時執行它們，以便它們執行不同的功能。

  - 使用自有裝置軟體和任務處理常式

    您可以使用 AWS IoT Device SDK 及其支援遠端操作的處理常式程式庫，為裝置撰寫自己的程式碼。若要部署和執行任務，請驗證裝置代理程式程式庫是否已正確安裝，且正在裝置上執行。

    您也可以選擇使用自己的處理常式來支援遠端操作。如需詳細資訊，請參閱 AWS IoT Device Client GitHub 儲存庫中的範例任務處理常式。

  - 使用 AWS IoT 裝置用戶端

    或者，您可以在裝置上安裝和執行 AWS IoT 裝置用戶端，因為它預設支援直接從主控台使用所有受管範本。

    裝置用戶端是以 C++ 撰寫的開放原始碼軟體，可供您在內嵌 Linux 型 IoT 裝置上編譯並安裝。裝置用戶端具有基礎用戶端和分散的用戶端功能。基礎用戶端 AWS IoT 會透過 MQTT 通訊協定建立與 的連線，並可連接不同的用戶端功能。

    若要在裝置上執行遠端操作，請使用裝置用戶端的用戶端任務功能。此功能包含剖析器，可接收實作任務文件中指定之遠端動作的任務文件和任務處理常式。如需有關裝置用戶端及其功能的詳細資訊，請參閱 AWS IoT 裝置用戶端。

在裝置上執行時，裝置用戶端會接收任務文件，並具有用於在文件中執行命令的平台特定實作。如需有關設定裝置用戶端和使用任務功能的詳細資訊，請參閱 AWS IoT 教學課程。

- 使用受支援的環境

針對每個受管範本，您會找到可用來執行遠端動作之環境的相關資訊。建議依照範本所述，使用支援 Linux 環境的範本。使用 AWS IoT 裝置用戶端執行受管範本遠端動作，因為它支援常見的微處理器和 Linux 環境，例如 Debian 和 Ubuntu。

受管範本遠端動作和任務文件

下一節列出適用於 AWS IoT 任務的不同 AWS 受管範本，並說明可在裝置上執行的遠端動作。下節具有任務文件的相關資訊，以及每個遠端動作之任務文件參數的描述。您的裝置端軟體使用範本名稱和參數執行遠端動作。

AWS 受管範本接受輸入參數，您可以在使用範本建立任務時為其指定值。所有受管範本都有兩個共同的選用輸入參數：`runAsUser` 和 `pathToHandler`。除了 `AWS-Reboot` 範本之外，範本需要額外的輸入參數，您必須在使用範本建立任務時為其指定值。這些所需輸入參數取決於您選擇的範本。例如，如果您選擇 `AWS-Download-File` 範本，您必須指定要安裝的套件清單，以及要從中下載檔案的 URL。

使用 AWS IoT 主控台或 AWS Command Line Interface (AWS CLI) 建立使用受管範本的任務時，指定輸入參數的值。使用 CLI 時，請使用 `document-parameters` 物件提供這些值。如需詳細資訊，請參閱 documentParameters。

> **Note**
>
> 只在從 AWS 受管範本建立任務時，才會使用 `document-parameters`。這個參數不能與自訂任務範本一起使用，也不能用於從自訂任務範本建立任務。

以下為常見選用輸入參數的敘述。您會在下節中看到每個受管範本所需之其他輸入參數的描述。

`runAsUser`

此參數指定是否以其他使用者身分執行任務處理常式。如果未在建立任務期間指定，則任務處理常式會以與裝置用戶端相同的使用者身分執行。在以其他使用者身分執行任務處理常式時，請指定不超過 256 個字元的字串值。

`pathToHandler`

此為在裝置上執行的任務處理常式的路徑。如果未在任務建立期間指定，則裝置用戶端會使用目前的工作目錄。

下文顯示不同的遠端動作、其任務文件及其接受的參數。這些範本都支援 Linux 環境，以便在裝置上執行遠端操作。

AWS–Download–File

範本名稱

`AWS-Download-File`

範本描述

提供 AWS 用於下載檔案的受管範本。

輸入參數

此範本具有下列必要參數。您也可以指定選用參數 `runAsUser` 和 `pathToHandler`。

`downloadUrl`

要從中下載檔案的 URL。這可以是網際網路資源、Amazon S3 中可以公開存取的物件，或者 Amazon S3 中只能由您裝置使用預先簽章 URL 存取的物件。如需有關使用預先簽章的 URL 和授予許可的詳細資訊，請參閱 預先簽章的 URL。

`filePath`

本機檔案路徑，其中會顯示裝置中存放下載檔案的位置。

裝置行為

裝置會從指定的位置下載檔案，確認下載已完成，並將其存放在本機。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (`download-file.sh`)；必須執行任務處理常式，才能下載檔案。並說明必要參數 `downloadUrl` 和 `filePath`。

```
{
```

```
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Download-File",
        "type": "runHandler",
        "input": {
          "handler": "download-file.sh",
          "args": [
            "${aws:iot:parameter:downloadUrl}",
            "${aws:iot:parameter:filePath}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Install–Application

範本名稱

AWS-Install-Application

範本描述

提供的受管範本 AWS ，用於安裝一或多個應用程式。

輸入參數

此範本具有下列必要參數 packages。您也可以指定選用參數 runAsUser 和 pathToHandler。

packages

此為要安裝的一或多個應用程式的空格分隔清單。

裝置行為

裝置會依照任務文件所述安裝應用程式。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (install-packages.sh)；必須執行任務處理常式，才能下載檔案。其也會展示必要參數 packages。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Install-Application",
        "type": "runHandler",
        "input": {
          "handler": "install-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Reboot

範本名稱

AWS-Reboot

範本描述

提供的受管範本 AWS ，用於重新啟動您的裝置。

輸入參數

此範本沒有必要參數。您可以指定選用參數 runAsUser 和 pathToHandler。

裝置行為

裝置重新啟動成功。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (reboot.sh)；必須執行任務處理常式，才能重新啟動裝置。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Remove–Application

範本名稱

AWS-Remove-Application

範本描述

提供的受管範本 AWS ，用於解除安裝一或多個應用程式。

輸入參數

此範本具有下列必要參數 packages。您也可以指定選用參數 runAsUser 和 pathToHandler。

packages

此為要解除安裝的一或多個應用程式的空格分隔清單。

裝置行為

裝置會依照任務文件所述解除安裝應用程式。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (remove-packages.sh)；必須執行任務處理常式，才能下載檔案。其也會展示必要參數 packages。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Remove-Application",
        "type": "runHandler",
        "input": {
          "handler": "remove-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Restart–Application

範本名稱

AWS-Restart-Application

範本描述

提供的受管範本 AWS ，用於停止和重新啟動一或多個 服務。

輸入參數

此範本具有下列必要參數 services。您也可以指定選用參數 runAsUser 和 pathToHandler。

服務

此為要重新啟動的一或多個應用程式的空格分隔清單。

裝置行為

指定應用程式會停止，然後在裝置上重新啟動。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (`restart-services.sh`)；必須執行任務處理常式，才能重新啟動系統服務。其也會展示必要參數 `services`。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Restart-Application",
        "type": "runHandler",
        "input": {
          "handler": "restart-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Start–Application

範本名稱

`AWS-Start-Application`

範本描述

提供的受管範本 AWS ，用於啟動一或多個 服務。

輸入參數

此範本具有下列必要參數 `services`。您也可以指定選用參數 `runAsUser` 和 `pathToHandler`。

`services`

此為要啟動的一或多個應用程式的空格分隔清單。

裝置行為

指定應用程式會在裝置上開始執行。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (start-services.sh)；必須執行任務處理常式，才能啟動系統服務。其也會展示必要參數 services。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Start-Application",
        "type": "runHandler",
        "input": {
          "handler": "start-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Stop–Application

範本名稱

AWS-Stop-Application

範本描述

提供的受管範本 AWS ，用於停止一或多個 服務。

輸入參數

此範本具有下列必要參數 services。您也可以指定選用參數 runAsUser 和 pathToHandler。

services

此為要停止的一或多個應用程式的空格分隔清單。

裝置行為

指定應用程式會在裝置上停止執行。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務處理常式和 Shell 指令碼的路徑 (stop-services.sh)；必須執行任務處理常式，才能停止系統服務。其也會展示必要參數 services。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Stop-Application",
        "type": "runHandler",
        "input": {
          "handler": "stop-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Run–Command

範本名稱

AWS-Run-Command

範本描述

提供的受管範本， AWS 用於執行 shell 命令。

輸入參數

此範本具有下列必要參數 command。您也可以指定選用參數 runAsUser。

command

以逗號分隔的命令字串。命令本身中包含的任何逗號都必須逸出。

裝置行為

裝置會依照任務文件指示執行 Shell 命令。

任務文件

下文顯示任務文件及其最新版本。範本會顯示任務命令路徑，以及您所提供將由裝置執行的命令。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Run-Command",
        "type": "runCommand",
        "input": {
          "command": "${aws:iot:parameter:command}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

主題

- 使用 從 AWS 受管範本建立任務 AWS 管理主控台
- 使用 從 AWS 受管範本建立任務 AWS CLI

使用 從 AWS 受管範本建立任務 AWS 管理主控台

使用 AWS 管理主控台 取得 AWS 受管範本的相關資訊，並使用這些範本建立任務。然後可以將建立的任務儲存為自己的自訂範本。

取得受管範本的詳細資訊

您可以取得可從 AWS IoT 主控台使用之不同受管範本的相關資訊。

1. 若要查看可用的受管範本，請前往 AWS IoT 主控台的任務範本中樞，然後選擇受管範本索引標籤。
2. 若要檢視詳細資訊，請選擇受管範本。

詳細資訊頁面包含以下資訊：

- 受管範本的名稱、說明和 Amazon 資源名稱 (ARN)。
- 可執行遠端操作的環境，如 Linux。
- JSON 任務文件，指定任務處理常式的路徑以及要在裝置上執行的命令。例如，下文顯示 AWS-Reboot 範本的範例任務文件。範本會顯示任務處理常式和 Shell 指令碼的路徑 (`reboot.sh`)；必須執行任務處理常式，才能重新啟動裝置。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

  如需有關任務文件以及各種遠端操作參數的詳細資訊，請參閱 受管範本遠端動作和任務文件。
- 任務文件的最新版本。

使用受管範本建立任務

您可以使用 AWS Management 主控台選擇要用來建立任務的 AWS 受管範本。本節將告訴您如何做到。

您也可以開始任務建立工作流程，然後選擇要在建立任務時使用的 AWS 受管範本。如需有關此工作流程的詳細資訊，請參閱 使用 建立和管理任務 AWS 管理主控台。

1. 選擇您的 AWS 受管範本

   前往 AWS IoT 主控台的任務範本中樞，選擇受管範本索引標籤，然後選擇您的範本。

2. 使用受管範本建立任務

   1. 在範本的詳細資訊頁面中，選擇 Create job (建立任務)。

      主控台會切換至已新增範本組態的 Create job (建立任務) 工作流程的 Custom job properties (自訂任務屬性) 步驟。

   2. 輸入唯一的英數字元任務名稱，以及選用說明和標籤，然後選擇 Next (下一步)。
   3. 選擇要在此任務中執行的物件或物件群組作為任務目標。
   4. 在 Job document (任務文件) 區段中，範本會顯示其組態設定和輸入參數。輸入所選範本的輸入參數值。例如，如果您選擇 AWS-Download-File 範本：
      - downloadUrl 中，輸入要下載的檔案 URL，例如：`https://example.com/index.html`。
      - filePath 中，輸入裝置上存放下載檔案的路徑，例如：*path/to/file*。

      您也可以選擇輸入參數 `runAsUser` 和 `pathToHandler` 的值。如需有關各個範本輸入參數的詳細資訊，請參閱 受管範本遠端動作和任務文件。
   5. 在 Job configuration (任務組態) 頁面上，選擇任務類型為連續或快照任務。快照任務在目標裝置和群組上完成執行後即完成。連續任務適用於物件群組，並在新增至所指定目標群組的任何裝置上執行。
   6. 繼續為任務新增任何其他組態，然後檢閱並建立任務。如需其他組態的資訊，請參閱：
      - 任務推展、排程和中止組態
      - 任務執行逾時和重試組態

從受管範本建立自訂任務範本

您可以使用 AWS 受管範本和自訂任務作為建立自訂任務範本的起點。若要建立自訂任務範本，請先從受 AWS 管範本建立任務，如上一節所述。

然後將自訂任務另存為範本，建立您的自訂任務範本。若要另存為範本：

1. 前往 AWS IoT 主控台的任務中樞，然後選擇包含受管範本的任務。

2. 選擇 Save as a job template (另存為任務範本)，然後建立自訂任務範本。如需有關建立自訂任務範本的詳細資訊，請參閱 從現有任務建立任務範本。

使用 從 AWS 受管範本建立任務 AWS CLI

使用 AWS CLI 取得 AWS 受管範本的相關資訊，並使用這些範本建立任務。您便可以將任務另存為範本，然後建立自己的自訂範本。

列出受管範本

list-managed-job-templates AWS CLI 命令會列出 中的所有任務範本 AWS 帳戶。

```
aws iot list-managed-job-templates
```

根據預設，執行此命令會顯示所有可用的 AWS 受管範本及其詳細資訊。

```
{
    "managedJobTemplates": [
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Reboot:1.0",
            "templateName": "AWS-Reboot",
            "description": "A managed job template for rebooting the device.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Remove-
Application:1.0",
            "templateName": "AWS-Remove-Application",
            "description": "A managed job template for uninstalling one or more
 applications.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
```

```
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Stop-Application:1.0",
            "templateName": "AWS-Stop-Application",
            "description": "A managed job template for stopping one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },

        ...

        {
            "templateArn": "arn:aws:iot:us-east-1::jobtemplate/AWS-Restart-
Application:1.0",
            "templateName": "AWS-Restart-Application",
            "description": "A managed job template for restarting one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        }
    ]
}
```

如需詳細資訊，請參閱 ListManagedJobTemplates。

取得受管範本的詳細資訊

describe-managed-job-template AWS CLI 命令會取得指定任務範本的詳細資訊。指定任務範本名稱和選用範本版本。若未指定範本版本，則會傳回預先定義的預設版本。以下說明一個範例，執行命令以取得有關 AWS-Download-File 範本的詳細資訊。

```
aws iot describe-managed-job-template \
    --template-name AWS-Download-File
```

這個命令顯示範本詳細資訊和 ARN、其任務文件以及 documentParameters 參數，該參數為範本輸入參數的鍵值對清單。如需有關不同範本和輸入參數的詳細資訊，請參閱 受管範本遠端動作和任務文件。

> **Note**
>
> 只有在從 AWS 受管範本建立任務時，才能使用您使用此 API 時傳回的documentParameters物件。這個物件不能用於自訂任務範本。如需示範如何使用這個參數的範例，請參閱 使用受管範本建立任務。

```
{
    "templateName": "AWS-Download-File",
    "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0",
    "description": "A managed job template for downloading a file.",
    "templateVersion": "1.0",
    "environments": [
        "LINUX"
    ],
    "documentParameters": [
        {
            "key": "downloadUrl",
            "description": "URL of file to download.",
            "regex": "(.*?)",
            "example": "http://www.example.com/index.html",
            "optional": false
        },
        {
            "key": "filePath",
            "description": "Path on the device where downloaded file is written.",
            "regex": "(.*?)",
            "example": "/path/to/file",
            "optional": false
        },
        {
            "key": "runAsUser",
            "description": "Execute handler as another user. If not specified, then
 handler is executed as the same user as device client.",
            "regex": "(.){0,256}",
            "example": "user1",
            "optional": true
        },
        {
            "key": "pathToHandler",
            "description": "Path to handler on the device. If not specified, then
 device client will use the current working directory.",
```

```
            "regex": "(.){0,4096}",
            "example": "/path/to/handler/script",
            "optional": true
        }
    ],
    "document": "{\"version\":\"1.0\",\"steps\":[{\"action\":{\"name
\":\"Download-File\",\"type\":\"runHandler\",\"input\":{\"handler\":
\"download-file.sh\",\"args\":[\"${aws:iot:parameter:downloadUrl}\",
\"${aws:iot:parameter:filePath}\"],\"path\":\"${aws:iot:parameter:pathToHandler}\"},
\"runAsUser\":\"${aws:iot:parameter:runAsUser}\"}}]}"
}
```

如需詳細資訊，請參閱 DescribeManagedJobTemplate。

使用受管範本建立任務

create-job AWS CLI 命令可用來從任務範本建立任務。這會將名為 thingOne 的裝置設為目標，並指定受管範本的 Amazon 資源名稱 (ARN)，以用作任務的基礎。您可以覆寫進階組態，例如逾時和取消組態，方法是傳遞 create-job 命令的相關聯參數。

這個範例說明如何使用 AWS-Download-File 範本建立任務。它也說明了如何使用 document-parameters 參數指定範本的輸入參數。

> **Note**
>
> 僅搭配 AWS 受管範本使用 document-parameters 物件。這個對象不能與自訂任務範本搭配使用。

```
aws iot create-job \
   --targets arn:aws:iot:region:account-id:thing/thingOne \
   --job-id "new-managed-template-job" \
   --job-template-arn arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0 \
   --document-parameters downloadUrl=https://example.com/index.html,filePath=path/to/
file
```

其中：

- *region* 是 AWS 區域。
- *account-id* 是唯一的 AWS 帳戶 數字。

- *thingOne* 是任務所針對的 IoT 物件名稱。
- *AWS-Download-File:1.0* 是受管範本的名稱。
- https://*example.com/index*.html 是要下載檔案的 URL。
- https://*pathto/file/index* 是裝置上用來存放下載檔案的路徑。

執行以下命令來建立範本 *AWS-Download-File* 的任務。

```
{
    "jobArn": "arn:aws:iot:region:account-id:job/new-managed-template-job",
    "jobId": "new-managed-template-job",
    "description": "A managed job template for downloading a file."
}
```

從受管範本建立自訂任務範本

1. 使用受管範本建立任務，如前一節中所述。
2. 使用您建立的任務的 ARN 建立自訂任務範本。如需詳細資訊，請參閱 從現有任務建立任務範本。

建立自訂任務範本

您可以使用 AWS CLI 和 AWS IoT 主控台來建立任務範本。您也可以使用 AWS CLI、 AWS IoT 主控台和 Fleet Hub for AWS IoT Device Management Web 應用程式，從任務範本建立任務。如需有關在 Fleet Hub 應用程式中使用任務範本的詳細資訊，請參閱在 Fleet Hub for AWS IoT Device Management 中使用任務範本。

> **Note**
>
> 任務文件中替代模式總數應小於或等於 10。

主題

- 使用 建立自訂任務範本 AWS 管理主控台
- 使用 建立自訂任務範本 AWS CLI

使用 建立自訂任務範本 AWS 管理主控台

本主題說明如何使用 AWS IoT 主控台建立、刪除和檢視任務範本的詳細資訊。

建立自訂任務範本

可以建立原始自訂任務範本或從現有任務建立任務範本。您也可以從使用 AWS 受管範本建立的現有任務建立自訂任務範本。如需詳細資訊，請參閱從受管範本建立自訂任務範本。

建立原始任務範本

1. 開始建立任務範本

   1. 前往 AWS IoT 主控台的任務範本中樞，然後選擇自訂範本索引標籤。
   2. 選擇 Create job template (建立任務範本)。

   > **Note**
   >
   > 您也可以從 Fleet Hub (機群中樞) 下的 Related services (相關服務) 頁面導覽至 Job templates (任務範本) 頁面。

2. 指定任務範本屬性

   在 Create job template (建立任務範本) 頁面中，輸入任務名稱的英數字元識別碼和英數字元說明，提供範本的其他詳細資訊。

   > **Note**
   >
   > 不建議在任務 ID 或描述中使用個人識別資訊。

3. 提供任務文件

   提供存放在 S3 儲存貯體中或作為任務內指定的內嵌任務文件的 JSON 任務檔案。使用此範本建立任務時，此任務檔案會成為任務文件。

   若任務檔案存放於 S3 儲存貯體中，請輸入 S3 URL 或選擇 Browse S3 (瀏覽 S3)，然後導覽至您的任務文件並加以選取。

   > **Note**
   >
   > 您只能選取目前區域中的 S3 儲存貯體。

4. 繼續為任務新增任何其他組態，然後檢閱並建立任務。如需其他選用組態的詳細資訊，請參閱下列項目：

- 任務推展、排程和中止組態
- 任務執行逾時和重試組態

從現有任務建立任務範本

1. 選擇您的任務。

   1. 前往 AWS IoT 主控台的任務中樞，然後選擇您要用作任務範本基礎的任務。
   2. 選擇 Save as a job template (另存為任務範本)。

   > **Note**
   >
   > 也可以選擇不同的任務文件或編輯原始任務中的進階組態，然後選擇 Create job template (建立任務範本)。您的新任務範本會出現在 Job templates (任務範本) 頁面上。

2. 指定任務範本屬性

   在 Create job template (建立任務範本) 頁面中，輸入任務名稱的英數字元識別碼和英數字元說明，提供範本的其他詳細資訊。

   > **Note**
   >
   > 任務文件是您在建立範本時指定的任務檔案。如果在任務內而非 S3 位置指定任務文件，則可在此任務的詳細資訊頁面中看到任務文件。

3. 繼續為任務新增任何其他組態，然後檢閱並建立任務。如需其他組態的資訊，請參閱：

   - 任務推展、排程和中止組態
   - 任務執行逾時和重試組態

從自訂任務範本建立任務

您可以前往任務範本的詳細資訊頁面，從自訂任務範本建立任務，如本主題所述。您也可以建立任務，或透過選擇執行任務建立工作流程時要使用的任務範本來建立任務。如需詳細資訊，請參閱使用 建立和管理任務 AWS 管理主控台。

本主題顯示如何從自訂任務範本的詳細資訊頁面建立任務。您也可以從 AWS 受管範本建立任務。如需詳細資訊，請參閱使用受管範本建立任務。

1. 選擇自訂任務範本

   前往 AWS IoT 主控台的任務範本中樞，然後選擇自訂範本索引標籤，然後選擇您的範本。

2. 使用自訂範本建立任務

   若要建立任務：

   1. 在範本的詳細資訊頁面中，選擇 Create job (建立任務)。

      主控台會切換至已新增範本組態的 Create job (建立任務) 工作流程的 Custom job properties (自訂任務屬性) 步驟。

   2. 輸入唯一的英數字元任務名稱，以及選用說明和標籤，然後選擇 Next (下一步)。

   3. 選擇要在此任務中執行的物件或物件群組作為任務目標。

      在 Job document (任務文件) 區段中，範本會顯示其組態設定。如果您想要使用不同的任務文件，請選擇 Browse (瀏覽)，然後選取不同的儲存貯體和文件。選擇 Next (下一步)。

   4. 在 Job configuration (任務組態) 頁面上，選擇任務類型為連續或快照任務。快照任務在目標裝置和群組上完成執行後即完成。連續任務適用於物件群組，並在新增至所指定目標群組的任何裝置上執行。

   5. 繼續為任務新增任何其他組態，然後檢閱並建立任務。如需其他組態的資訊，請參閱：

      - 任務推展、排程和中止組態
      - 任務執行逾時和重試組態

> **Note**
>
> 當從任務範本建立的任務更新任務範本提供的現有參數時，這些更新的參數將會覆寫該任務之任務範本所提供的現有參數。

您也可以使用 Fleet Hub Web 應用程式，從任務範本建立任務。如需有關在 Fleet Hub 中建立任務的資訊，請參閱在 Fleet Hub for AWS IoT Device Management 中使用任務範本。

刪除任務範本

若要刪除任務範本，請先前往 AWS IoT 主控台的任務範本中樞，然後選擇自訂範本索引標籤。然後，選擇要刪除的任務範本，並選擇刪除。

> **Note**
>
> 此為永久性刪除，且任務範本不會再顯示在 Custom templates (自訂範本) 索引標籤上。

使用 建立自訂任務範本 AWS CLI

本主題說明如何使用 AWS CLI建立、刪除和擷取任務範本的詳細資訊。

從頭建立任務範本

下列 AWS CLI 命令顯示如何使用存放在 S3 儲存貯體中的任務文件 (*job-document.json*) 和具有從 Amazon S3 (*S3DownloadRole*) 下載檔案許可的角色來建立任務。

```
aws iot create-job-template  \
      --job-template-id 010  \
      --description "My custom job template for updating the device firmware"
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\":
 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\":
 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

選用的 `timeout-config` 參數會指定每個裝置必須完成任務執行的時間量。任務執行狀態設定為 `IN_PROGRESS` 時，計時器即會開始。在時間過期之前，如果任務執行狀態未設定為其他終止狀態，就會將其設定為 `TIMED_OUT`。

進行中的計時器無法更新，並會套用到任務的所有任務啟動。每當任務啟動保持 IN_PROGRESS 狀態超過此間隔時，任務啟動就會失敗並切換到終端機TIMED_OUT狀態。 AWS IoT 也會發佈 MQTT 通知。

如需有關建立任務推展和中止組態的詳細資訊，請參閱任務推展和中止組態。

> **Note**
>
> 指定為 Amazon S3 檔案的任務文件會在您建立任務時擷取。如果您在建立任務後變更作為任務文件來源的 Amazon S3 檔案的內容，則傳送到任務目標的內容不會變更。

從現有任務建立任務範本

下列 AWS CLI 命令會透過指定現有任務的 Amazon Resource Name (ARN) 來建立任務範本。新的任務範本會使用任務中指定的所有組態。您可以選擇性地使用任何選用參數，來變更現有任務中的任何組態。

```
aws iot create-job-template  \
      --job-arn arn:aws:iot:region:123456789012:job/job-name  \
      --timeout-config inProgressTimeoutInMinutes=100
```

取得任務範本的詳細資訊

下列 AWS CLI 命令會取得指定任務範本的詳細資訊。

```
aws iot describe-job-template \
      --job-template-id template-id
```

該命令會顯示下列輸出。

```
{
   "abortConfig": {
      "criteriaList": [
         {
```

```
            "action": "string",
            "failureType": "string",
            "minNumberOfExecutedThings": number,
            "thresholdPercentage": number
         }
      ]
   },
   "createdAt": number,
   "description": "string",
   "document": "string",
   "documentSource": "string",
   "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": number,
         "incrementFactor": number,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": number,
            "numberOfSucceededThings": number
         }
      },
      "maximumPerMinute": number
   },
   "jobTemplateArn": "string",
   "jobTemplateId": "string",
   "presignedUrlConfig": {
      "expiresInSec": number,
      "roleArn": "string"
   },
   "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
}
```

列出任務範本

下列 AWS CLI 命令會列出 中的所有任務範本 AWS 帳戶。

```
aws iot list-job-templates
```

該命令會顯示下列輸出。

```
{
   "jobTemplates": [
      {
         "createdAt": number,
         "description": "string",
         "jobTemplateArn": "string",
         "jobTemplateId": "string"
      }
   ],
   "nextToken": "string"
}
```

若要擷取其他結果頁面，請使用 nextToken 欄位的值。

刪除任務範本

下列 AWS CLI 命令會刪除指定的任務範本。

```
aws iot delete-job-template \
      --job-template-id template-id
```

此命令不會顯示輸出。

從自訂任務範本建立任務

下列 AWS CLI 命令會從自訂任務範本建立任務。它會將名為 thingOne 的裝置設為目標，並指定任務範本的 Amazon 資源名稱 (ARN)，以用作任務的基礎。您可以覆寫進階組態，例如逾時和取消組態，方法是傳遞 create-job 命令的相關聯參數。

> **Warning**
>
> 從 AWS 受管範本建立任務時，document-parameters 物件只能與 create-job 命令搭配使用。這個對象不能與自訂任務範本搭配使用。如需示範如何使用這個參數建立任務的範例，請參閱 使用受管範本建立任務。

```
aws iot create-job \
```

```
    --targets arn:aws:iot:region:123456789012:thing/thingOne  \
    --job-template-arn arn:aws:iot:region:123456789012:jobtemplate/template-id
```

任務 組態

對於部署至指定目標的每個任務，您可以擁有下列其他組態。

- 推展：定義每分鐘接收任務文件的裝置數量。
- 排程：除了使用週期性維護時段之外，還排程在未來日期和時間執行任務。
- 中止：在某些裝置未接收到任務通知或您的裝置報告其任務執行失敗等情況下取消任務。
- 逾時：如果在其任務執行開始後的特定期間內未接收到任務目標的回應，則任務可能會失敗。
- 重試：如果您的裝置在嘗試完成任務執行時報告失敗，或如果任務執行逾時，重試任務執行。

透過使用這些組態，您可以監控任務執行的狀態，並避免將錯誤的更新傳送至整個機群。

主題

- 任務組態的運作方式
- 指定其他組態

任務組態的運作方式

在部署任務時使用推展和中止組態，在任務執行時則使用逾時和重試組態。下列各節說明有關這些組態運作方式的詳細資訊。

主題

- 任務推展、排程和中止組態
- 任務執行逾時和重試組態

任務推展、排程和中止組態

您可以使用任務推展、排程和中止組態，來定義接收任務文件的裝置數量、排程任務推展，以及決定取消任務的條件。

任務推展組態

您可以指定何時通知目標有一項待執行的任務。您亦可建立一個分階段推展的任務，來管理更新、重新啟動以及其他操作。若要指定通知目標的方式，請使用任務推展率。

任務推展率

您可以使用恆定推展率或指數推展率來建立推展組態。若要指定每分鐘通知任務目標的數量上限，請使用恆定推展率。

AWS IoT 當符合各種條件和閾值時，任務可以使用指數推展率進行部署。如果失敗任務數量與您指定的一組條件相符，則可取消任務推展。您可以使用 `JobExecutionsRolloutConfig` 物件，在建立任務時設定任務推展率條件。您還可以使用 `AbortConfig` 物件在建立任務時設定任務中止條件。

下列範例說明推展率的運作方式。例如，基本速率為每分鐘 50 個、增量因數為 2、已通知和成功裝置的數量各為 1000 的任務推展率，將依以下方式運作：任務將以每分鐘 50 個任務執行的速率開始，並以該速率繼續，直到 1000 個物件均收到任務執行通知或發生 1000 次成功的任務執行。

下表說明推展如何在前四個增量中繼續進行。

每分鐘推展率	50	100	200	400
滿足速率提高的已通知裝置或成功執行任務的數量	1,000	2,000	3,000	4,000

> **Note**
>
> 如果最大並行任務限制為 500 個任務 (`isConcurrent = True`)，則所有作用中的任務將保持狀態 `IN-PROGRESS` 且不會推展任何新的任務執行作業，直到並行任務數量達到 499 以下 (`isConcurrent = False`)。此原則適用於連續任務和快照任務。
> 如果 `isConcurrent = True`,表示此任務目前正在將任務執行推展到目標群組中的所有裝置。如果 `isConcurrent = False`，任務已完成將所有任務執行推展至目標群組中的所有裝置。它會在目標群組中的所有裝置一旦達到終端狀態或目標群組的臨界值百分比後，就會更新其狀態 (如果您選取了任務中止組態)。`isConcurrent = True` 與 `isConcurrent = False` 的任務層級狀態皆為 `IN_PROGRESS`。
> 如需作用中任務與並行任務限制的詳細資訊，請參閱 作用中和並行任務限制。

使用動態物件群組進行連續任務的任務推展率

當您使用連續任務在機群上推展遠端操作時， AWS IoT Jobs 會推展目標物件群組中裝置的任務執行。對於新增至動態物件群組中的新裝置，這些任務執行會繼續推展到那些裝置，即使建立了任務之後也是如此。

建立任務之前，推展組態只能控制加入群組之裝置的推展率。建立任務後，對於任何新裝置，只要裝置加入目標群組，就會以近乎即時的速度建立任務執行。

任務排程組態

您可以使用預定的開始時間、結束時間以及各項任務執行作業達到結束時間後所應發生的結束行為，藉此預先排定最多 1 年以後的連續或快照任務。此外，您還可以為連續任務建立具有彈性頻率、開始時間和持續時間的選用週期性維護時段，以便將任務文件推展到目標群組中的所有裝置。

任務排程組態

開始時間

排程任務的開始時間是任務開始向目標群組中所有裝置推展任務文件的未來日期和時間。排程任務的開始時間會套用至連續任務和快照任務。最初建立排程任務時，它會維持為 SCHEDULED 的狀態。到達您選取的 `startTime` 後，它會更新為 IN_PROGRESS 並開始任務文件推展。`startTime` 必須少於或等於從您建立排程任務初始日期和時間起算的一年。

如需使用 API 命令或 `startTime`時 語法的詳細資訊 AWS CLI，請參閱時間戳記。

對於在遵守日光節約時間 (DST) 的位置中的週期性維護時段期間進行選用排程組態的任務，從 DST 切換到標準時間以及從標準時間切換到 DST 時，時間會變動一小時。

> **Note**
>
> 中顯示的時區 AWS 管理主控台 是您目前的系統時區。但是，這些時區將在系統中轉換為 UTC。

結束時間

排程任務的結束時間是任務停止向目標群組中所有裝置推展任務文件的未來日期和時間。排程任務的結束時間會套用至連續任務和快照任務。在排程任務達到選取的 `endTime` 且所有任務執行都達到了終端狀態之後，它會將其狀態從 IN_PROGRESS 更新為 COMPLETED。`endTime` 必須少於或等於從您建立排程任務初始日期和時間起算的二年。`startTime` 與 `endTime` 之間的最短持續時間為 30 分鐘。繼續嘗試任務執行重試，直到任務達到 `endTime` 為止，然後 `endBehavior` 將指示如何繼續進行。

如需使用 API 命令或 endTime時 語法的詳細資訊 AWS CLI，請參閱時間戳記。

對於在遵守日光節約時間 (DST) 的位置中的週期性維護時段期間進行選用排程組態的任務，從 DST 切換到標準時間以及從標準時間切換到 DST 時，時間會變動一小時。

> **Note**
>
> 中顯示的時區 AWS 管理主控台 是您目前的系統時區。但是，這些時區將在系統中轉換為 UTC。

結束行為

排程任務的結束行為會決定任務到達選定的 endTime 時，該任務和所有未完成的任務執行項目應有的狀態。

以下列出建立任務或任務範本時可供選取的結束行為：

- STOP_ROLLOUT
  - STOP_ROLLOUT 會停止將任務文件推展至任務目標群組中的所有剩餘裝置。此外，所有 QUEUED 和 IN_PROGRESS 任務執行都會繼續進行，直到達到終端狀態為止。除非您選取 CANCEL 或 FORCE_CANCEL，否則這是預設的結束行為。
- CANCEL
  - CANCEL 會停止將任務文件推展至任務目標群組中的所有剩餘裝置。此外，所有 QUEUED 任務執行項目都會取消，而所有 IN_PROGRESS 任務執行項目會繼續進行，直到達到終端狀態為止。
- FORCE_CANCEL
  - FORCE_CANCEL 會停止將任務文件推展至任務目標群組中的所有剩餘裝置。此外，所有 QUEUED 和 IN_PROGRESS 任務執行項目都會取消。

> **Note**
>
> 若要選取 endbehavior，您必須選取 endtime

最長持續期間

不論 startTime 和 endTime 為何，排程任務的最長持續時間都必須小於或等於兩年。

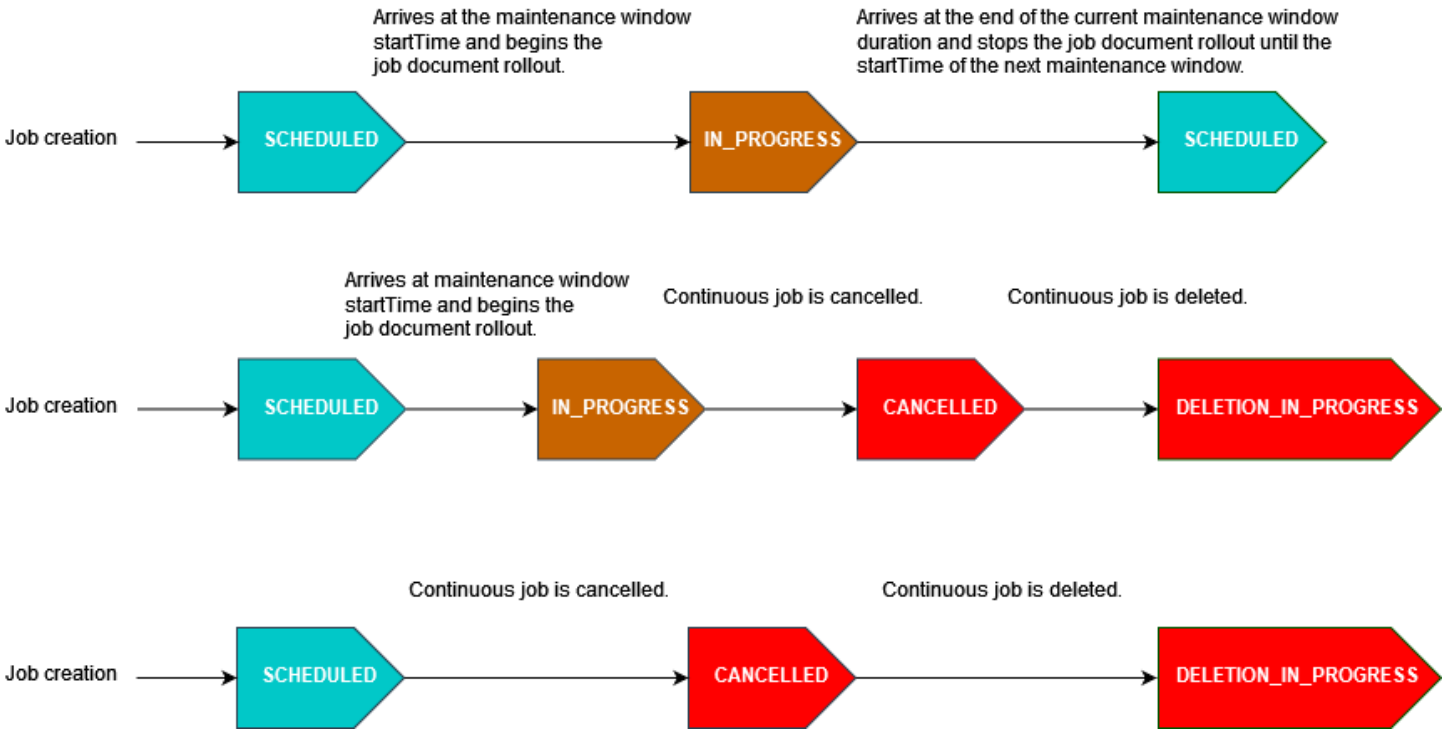
下表列出排程任務持續時間的常見情境：

排程任務範例編號	startTime	endTime	最長持續期間
1	在初始任務建立後立即執行。	初始任務建立一年後。	一年
2	初始任務建立後一個月。	初始建立任務後的 13 個月。	一年
3	初始任務建立一年後。	初始任務建立兩年後。	一年
4	在初始任務建立後立即執行。	初始任務建立兩年後。	兩年

週期性維護時段

維護時段是 排程組態內的選用組態， AWS 管理主控台 以及 `CreateJob`和 `CreateJobTemplate` APIs`SchedulingConfig`內的選用組態。您可以使用預先決定的開始時間、持續時間和頻率 (每日、每週或每月) 來設定週期性維護時段。維護時段僅適用於連續任務。週期性維護時段的最長持續時間為 23 小時 50 分鐘。

下圖說明具有選用維護時段之各種排程任務情境的任務狀態：



如需任務狀態的詳細資訊，請參閱 任務和任務執行狀態。

> **Note**
>
> 如果任務在維護時段到達 `endTime`，它將從 `IN_PROGRESS` 更新為 `COMPLETED`。此外，任何剩餘的任務執行都會遵循任務的 `endBehavior`。

Cron 表達式

對於在維護時段期間以自訂頻率推出任務文件的排定任務，自訂頻率會使用 cron 表達式輸入。Cron 表達式有六個必要欄位，以空格隔開。

語法

```
cron(fields)
```

欄位	Values (數值)	萬用字元
分鐘	0-59	, - * /
小時	0-23	, - * /

欄位	Values (數值)	萬用字元
月中的日	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
週中的日	1-7 或 SUN-SAT	, - * ? L #
年	1970-2199	, - * /

萬用字元

- , (逗號) 萬用字元包含額外的值。在 Month (月) 欄位，JAN、FEB、MAR 包括 January (一月)、February (二月) 與 March (三月)。
- - (破折號) 萬用字元用於指定範圍。在 Day (日) 欄位，1-15 包含指定月份的 1 至 15 號。
- * (星號) 包含欄位中所有的值。在 Hours (小時) 欄位，* 包含每個小時。您無法在月中的特定一天和週中的特定一天兩個欄位同時使用 *。若您在其中一個欄位使用它，您必須在另一個欄位使用 ?。
- / (斜線) 萬用字元用於指定增量。在 Minutes (分鐘) 欄位，您可以輸入 1/10 指定每十分鐘的間隔，從小時的第一分鐘開始 (例如第 11、第 21、第 31 分鐘等)。
- ? (問號) 萬用字元用於表示不限定任何一個。在當月幾號欄位中，您可以輸入 7，且如果您不在意當月的 7 號是星期幾，就可以在星期幾欄位中輸入 ?。
- L 萬用字元在 Day-of-month (月中的日) 或 Day-of-week (週中的日) 欄位可指定月份或週的最後一天。
- **W** 萬用字元在 Day-of-month (月中的日) 欄位可指定工作日。在 Day-of-month (月中的日) 欄位，**3W** 指定的是月份中最接近第三個工作日的日子。
- # 萬用字元在 Day-of-week (週中的日) 欄位可指定某個月中某週特定日子的特定執行個體。例如，3#2 代表則該月的第二個星期二：3 是指星期二，因為它是每週的第三天，2 指的是一個月內該類型的第二天。

> **Note**
>
> 如果您使用 '#' 字元，則只能在星期幾欄位中定義一個表達式。例如："3#1,6#3" 無效，因為它被轉譯為兩個表達式。

限制

- 您無法在同一個 cron 表達式中指定 Day-of-month (月中的日) 和 Day-of-week (週中的日) 欄位。如果您在其中一個欄位指定了一值 (或 *)，則必須在另一個欄位中使用?。

範例

針對週期性維護時段的 `startTime` 使用 cron 表達式時，請參閱下列範例 cron 字串。

分鐘	小時	月中的日	月	週中的日	年	意義
0	10	*	*	?	*	在每天上午 10:00 (UTC) 執行
15	12	*	*	?	*	在每天下午 12:15 (UTC) 執行
0	18	?	*	MON-FRI	*	在每週一至週五下午 6:00 (UTC) 執行
0	8	1	*	?	*	在每個月第一天上午 8:00 (UTC) 執行

週期性維護時段持續時間結束邏輯

當維護時段期間的任務推展到達目前維護時段結束時，將會發生下列動作：

- 任務將停止向目標群組中的所有剩餘裝置推展任務文件。它將在下一個維護時段的 `startTime` 繼續。
- 狀態為 QUEUED 的所有工作執行都會保留在 QUEUED 中，直到下一個維護時段的 `startTime` 發生為止。在下一個時段中，它們可以在裝置準備好開始執行任務文件中指定的操作時切換至 IN_PROGRESS。

- 所有 IN_PROGRESS 狀態為的任務執行都會繼續執行任務文件中指定的動作，直到達到結束狀態為止。JobExecutionsRetryConfig 中指定的任何重試嘗試都會在下一個維護時段的 startTime 開始進行。

任務中止組態

達到閾值百分比的裝置滿足該條件時，即可使用此組態來建立取消任務的條件。例如，在以下情況下，您可以使用此組態來取消任務：

- 在有達到閾值百分比的裝置未接收到任務執行通知時，例如您的裝置與空中 (OTA) 更新不相容時。在這種情況下，您的裝置可報告 REJECTED 狀態。
- 在有達到閾值百分比的裝置報告其任務執行失敗時，例如當您的裝置在嘗試從 Amazon S3 URL 下載任務文件時發生連線中斷時。在這種情況下，您的裝置必須以程式設計方式向 AWS IoT報告 FAILURE 狀態。
- 在任務執行開始後有達到閾值百分比的裝置任務執行逾時，因此報告 TIMED_OUT 狀態時。
- 出現多次重試失敗時。新增重試組態時，每次重試嘗試均會對您的 AWS 帳戶產生額外費用。在這種情況下，取消任務可以取消已排入佇列的任務執行，並避免這些執行的重試嘗試。如需重試組態以及將其與中止組態一起使用的詳細資訊，請參閱 任務執行逾時和重試組態。

您可以使用 AWS IoT 主控台或 任務 API 來設定 AWS IoT 任務中止條件。

任務執行逾時和重試組態

當任務執行的進行時間超過設定的持續時間時，使用任務執行逾時組態向您傳送 任務通知。使用任務執行重試組態在任務失敗或逾時時重試執行。

任務執行逾時組態

使用任務執行逾時組態，當任務執行長時間非預期卡在 IN_PROGRESS 狀態時即通知您。當任務為 IN_PROGRESS 時，您可以監控任務執行的進度。

任務逾時的計時器

計時器有兩種類型：進行中計時器和步驟計時器。

進行中計時器

建立任務或任務範本時，您可為進行中計時器指定一個介於 1 分鐘到 7 天之間的值。您可以更新此計時器的值，直到任務執行開始。計時器啟動後即無法更新，計時器值會套用於任務的所有任務執行。每

當任務執行保持 IN_PROGRESS 狀態超過此間隔時，任務執行就會失敗並切換到終端機TIMED_OUT狀態。 AWS IoT 也會發佈 MQTT 通知。

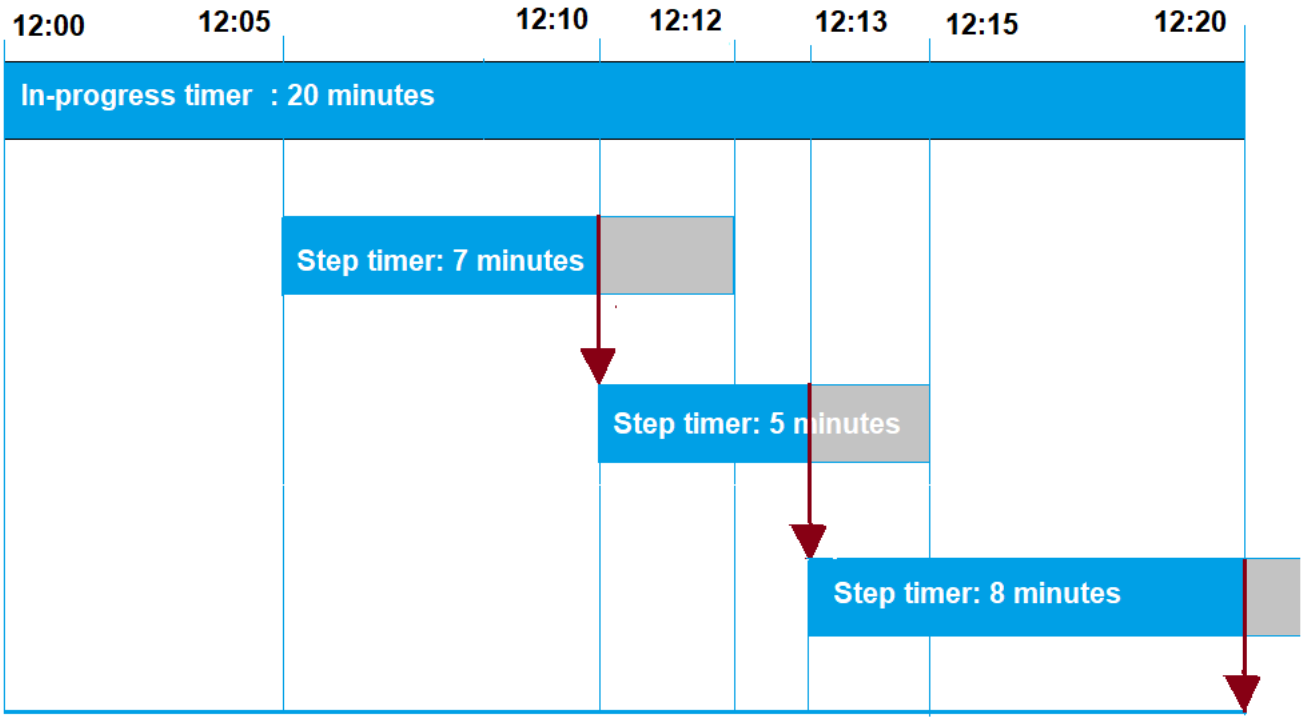
步驟計時器

您還可以設定僅適用於要更新的任務執行的步驟計時器。此計時器對進行中計時器沒有任何影響。每次更新任務執行時，您可以為步驟計時器設定新值。在啟動物件的下一個待處理任務執行時，您也可以建立新的步驟計時器。如果任務執行維持在 IN_PROGRESS 狀態的時間超過步驟計時器隔時，任務執行就會失敗，並切換到終止的 TIMED_OUT 狀態。

> **Note**
>
> 您可以使用 AWS IoT 主控台或 AWS IoT 任務 API 來設定進行中計時器。若要指定步驟計時器，請使用 API。

計時器如何處理任務逾時

下圖說明在 20 分鐘逾時時間內進行中逾時和步驟逾時相互互動方式。



以下顯示不同步驟：

1. 12：00

建立新任務，並在建立任務時啟動 20 分鐘的進行中計時器。進行中計時器開始執行，且任務執行切換為 IN_PROGRESS 狀態。

2. 12:05 PM

   建立一個值為 7 分鐘的新步驟計時器。任務執行現在將在 12:12 PM 逾時。

3. 12:10 PM

   建立一個值為 5 分鐘的新步驟計時器。當建立新步驟計時器時，會捨棄前一個步驟計時器，且任務執行現在將會在 12:15 PM 逾時。

4. 12:13 PM

   建立一個值為 9 分鐘的新步驟計時器。捨棄前一個步驟計時器，且任務執行現在將在 12:20 PM 逾時，因為進行中計時器會在 12:20 PM 逾時。步驟計時器無法超過進行中計時器的絕對界限。

任務執行重試組態

當滿足某一組條件時，您可以使用重試組態來重試任務執行。當任務逾時或裝置失敗時，可以嘗試重試。若要因逾時失敗而重試執行，必須啟用逾時組態。

如何使用重試組態

使用下列步驟重試組態：

1. 確定是否針對 FAILED、TIMED_OUT 或兩者的失敗條件使用重試組態。針對 TIMED_OUT 狀態，報告狀態之後， AWS IoT Jobs 會自動重試裝置的任務執行。
2. 若為 FAILED 狀態，請檢查您的任務執行失敗是否可以重試。如果裝置可重試，請將您的裝置以程式設計方式向 AWS IoT報告 FAILURE 狀態。以下部分介紹了有關可重試和不可重試失敗的詳細資訊。
3. 使用上述資訊來指定要用於每種失敗類型的重試次數。針對單一裝置，您可為兩種失敗類型組合指定最多 10 次重試。當執行成功或達到指定的嘗試次數時，重試嘗試即會自動停止。
4. 新增中止組態以在重複重試失敗的情況下取消任務，進而避免在發生大量重試嘗試時產生額外費用。

> **Note**
>
> 當任務到達週期性維護時段結束時，所有 IN_PROGRESS 任務執行會繼續執行任務文件中識別的動作，直到達到結束狀態為止。如果任務執行在維護時段之外到達 FAILED 或 TIMED_OUT

的終端狀態，若重試次數未用完，則會在下一個維護時段進行重試。在下一次維護時段的 `startTime` 時，會建立新的任務執行，並進入 QUEUED 狀態，直到裝置準備好開始為止。

重試和中止組態

每次重試嘗試都會對您的 產生額外費用 AWS 帳戶。為避免重複重試失敗而產生額外費用，我們建議新增中止組態。如需定價的詳細資訊，請參閱 AWS IoT Device Management 定價。

在裝置的高閾值百分比逾時或報告失敗時，您可能會遇到多次重試失敗。在這種情況下，您可以使用中止組態來取消任務，並避免任何已排入佇列的任務執行或進一步重試嘗試。

> **Note**
>
> 當滿足取消任務執行的中止條件時，僅會取消 QUEUED 任務。將不會嘗試對裝置進行任何佇列重試。但是，系統不會取消目前處於 IN_PROGRESS 狀態的任務執行。

在重試失敗的任務執行之前，我們亦建議您檢查任務執行失敗是否可重試，如下段所述。

重試失敗類型 **FAILED**

若要嘗試重試失敗類型 FAILED，則裝置必須以程式設計方式向 AWS IoT報告任務執行失敗的 FAILURE 狀態。使用重試 FAILED 任務執行的條件來設定重試組態，並指定要執行的重試次數。當 AWS IoT 任務偵測到 FAILURE 狀態時，它會自動嘗試重試裝置的任務執行。重試將繼續進行，直到任務執行成功或達到最大重試次數。

您可以追蹤每次重試作業的嘗試情形，以及在這些裝置上執行的任務。透過追蹤執行狀態，在嘗試指定的重試次數後，您可以使用裝置報告失敗並啟動另一次重試嘗試。

可重試和不可重試的失敗

您的任務執行失敗可能是可重試或不可重試的失敗。每次重試嘗試都會對您的 AWS 帳戶產生費用。為避免多次重試嘗試產生額外費用，請先考慮檢查任務執行失敗是否可重試。可重試失敗的範例包括在嘗試從 Amazon S3 URL 下載任務文件時您的裝置所遇到的連線錯誤。如果您的任務執行失敗可重試，請使用程式將您的裝置設計為若任務執行失敗，則報告 FAILURE。然後，設定重試組態以重試 FAILED 執行。

如果執行無法重試，則若要避免重試和可能會對您的帳戶產生額外費用，我們建議使用程式將裝置設計為向 AWS IoT報告 REJECTED 狀態。不可重試失敗的範例包括：裝置與接收任務更新不相容，或者在

執行任務時遇到記憶體錯誤。在這些情況下， AWS IoT Jobs 不會重試任務執行，因為它只會在偵測到 FAILED或 TIMED_OUT 狀態時重試任務執行。

確定可重試任務執行失敗後，如果重試嘗試仍失敗，請考慮檢查裝置日誌。

> **Note**
>
> 在具有選用排程組態的任務達到其 endTime 時，選取的 endBehavior 會停止將任務文件推展至目標群組中所有剩餘的裝置，並指定如何繼續其餘的任務執行。如果透過重試組態選取，則會重試嘗試。

重試失敗類型 **TIMEOUT**

如果您在建立任務時啟用逾時，則當狀態從 變更為 IN_PROGRESS 時， AWS IoT 任務會嘗試重試裝置的任務執行TIMED_OUT。當進行中計時器逾時，或者您指定的步驟計時器為 IN_PROGRESS 然後逾時，即可能會發生此狀態變更。重試會繼續進行，直到任務執行成功或達到此失敗類型的重試次數上限。

連續的任務和物件群組成員資格更新

對於任務狀態為 IN_PROGRESS 的連續任務，則當物件的群組成員資格更新時，重試嘗試次數將重設為零。例如，假定您指定了五次重試嘗試，並且已執行了三次重試。如果現已從物件群組中移除物件，然後重新連結群組 (例如動態物件群組)，重試嘗試次數將重設為零。現在，您可以對物件群組執行五次重試嘗試，而不是剩餘的兩次嘗試。此外，當物件從物件群組中移除時，系統將取消其他重試嘗試。

指定其他組態

當您建立任務或任務範本時，您可以指定這些其他組態。下文顯示您何時可以指定這些組態。

- 建立自訂任務範本時。從範本建立任務時，系統將會儲存您指定的其他組態設定。
- 使用任務檔案建立自訂任務時。任務檔案可以是上傳至 S3 儲存貯體中的 JSON 檔案。
- 使用自訂任務範本建立自訂任務時。如果範本已指定了這些設定，則可以重新使用這些設定，或是指定新的組態設定覆寫這些設定。
- 使用 AWS 受管範本建立自訂任務時。

主題

- 使用 指定任務組態 AWS 管理主控台

- 使用任務 API 指定 AWS IoT 任務組態

使用 指定任務組態 AWS 管理主控台

您可以使用 AWS IoT 主控台為您的任務新增不同的組態。建立任務後，您可以在任務詳細資訊頁面上看到任務組態的狀態詳細資訊。如需不同組態及其運作方式的詳細資訊，請參閱 任務組態的運作方式。

建立任務或任務範本時新增任務組態。

建立自訂任務範本時

建立自訂任務範本時指定推展組態

1. 前往 AWS IoT 主控台的任務範本中樞，然後選擇建立任務範本。
2. 指定任務範本屬性，提供任務文件，展開要新增的組態，然後指定組態參數。

建立自訂任務時

建立自訂任務時指定推展組態

1. 前往 AWS IoT 主控台的任務中樞，然後選擇建立任務。
2. 選擇 Create a custom job (建立自訂任務) 並指定任務屬性、目標，以及是否針對任務文件使用任務檔案或任務範本。您可以使用自訂範本或 AWS 受管範本。
3. 選擇任務組態，然後展開 Rollout configuration (推展組態) 指定是否使用 Constant rate (恆定速率) 或 Exponential rate (指數速率)。然後，指定組態參數。

下一節將顯示您可以為每個組態指定的參數。

推展組態

您可以指定使用恆定推展率或指數推展率。

- 設定恆定推展率

  若要設定任務執行的恆定速率，請選擇恆定速率，然後針對速率的上限指定每分鐘最大值。此值為選用值，範圍從 1 到 1000。如果未設定，則將使用 1000 作為預設值。
- 設定指數推展率

若要設定指數速率，請選擇 Exponential rate (指數速率)，然後指定這些參數：

- 每分鐘的基本速率

  在達到速率提高標準的已通知的裝置數量或成功的裝置數量閾值之前的任務執行速率。

- 增量因數

  在 Number of notified devices (已通知裝置數量) 或 Number of succeeded devices (成功裝置數量) 閾值滿足 Rate increase criteria (速率提高標準) 後推展率增加的指數因數。

- 速率提高標準

  Number of notified devices (已通知的裝置數量) 或 Number of succeeded devices (成功的裝置數量) 的閾值。

中止組態

選擇 Add new configuration (新增組態) 並為每個組態指定下列參數：

- 失敗類型

  指定啟動任務中止的失敗類型。其中包括 FAILED (失敗)、REJECTED (拒絕)、TIMED_OUT (逾時) 或 ALL (所有)。

- 增量因數

  指定在符合任務中止條件之前，必須發生的任務執行完成數量。

- 閾值百分比

  指定啟動任務中止的已執行物件總數。

Scheduling configuration (排程組態)

每項任務都可以在初始建立時立即開始、或排定在之後的日期和時間開始，或者在週期性維護時段進行。

選擇 Add new configuration (新增組態) 並為每個組態指定下列參數：

- 任務開始

  指定任務開始的日期和時間。

- 週期性維護時段

  週期性維護時段用於定義任務可將任務文件部署到任務中目標裝置的特定日期和時間。維護時段可以在每天、每週、每月或自訂日期和時間重複。

- 任務結束

  指定任務開始的日期和時間。

- 任務結束行為

  為所有未完成的任務執行項目選取當任務完成時應有的結束行為。

> **Note**
>
> 在具有選用排程組態和所選結束時間的任務達到結束時間時，任務會停止推展至目標群組中所有剩餘的裝置。它也會利用選取的結束行為，此行為有關如何根據重試組態繼續進行剩餘的工作執行及其重試嘗試。

逾時組態

根據預設不會逾時，且系統已取消或刪除您的任務執行。若要使用逾時，請選擇啟用逾時，然後指定一個介於 1 分鐘到 7 天之間的逾時值。

重試組態

> **Note**
>
> 建立任務後，即無法更新重試次數。您只能移除所有失敗類型的重試組態。建立任務時，請考慮用於組態的適當重試次數。為避免因潛在的重試失敗而產生超額成本，請新增中止組態。

選擇 Add new configuration (新增組態) 並為每個組態指定下列參數：

- 失敗類型

  指定應觸發任務執行重試的失敗類型。其中包括 Failed (失敗)、Timeout (逾時) 及 All (全部)。

- 重試次數

指定選定的 Failure type (失敗類型) 的重試次數。對於兩種失敗類型的組合，最多可嘗試 10 次重試。

使用任務 API 指定 AWS IoT 任務組態

您可以使用 CreateJob 或 CreateJobTemplate API 來指定不同的任務組態。下列各節描述如何新增這些組態。新增組態後，您可以使用 JobExecutionSummary 和 JobExecutionSummaryForJob 來查看其狀態。

如需不同組態及其運作方式的詳細資訊，請參閱 任務組態的運作方式。

推展組態

您可以為推展組態指定恆定推展率或指數推展率。

- 設定恆定推展率

  若要設定恆定推展率，請使用 JobExecutionsRolloutConfig 物件將 maximumPerMinute 參數新增至 CreateJob 請求。此參數指定可以發生任務執行的速率上限。此值為選用值，範圍從 1 到 1000。如果未設定該值，則將使用 1000 作為預設值。

```
    "jobExecutionsRolloutConfig": {
        "maximumPerMinute": 1000
    }
```

- 設定指數推展率

  若要設定可變任務推展率，請使用 JobExecutionsRolloutConfig 物件。您可以在執行 CreateJob API 操作時設定 ExponentialRolloutRate 屬性。以下範例使用 exponentialRate 參數來設定指數推展率。如需這些參數的詳細資訊，請參閱 ExponentialRolloutRate。

```
{
...
  "jobExecutionsRolloutConfig": {
    "exponentialRate": {
      "baseRatePerMinute": 50,
      "incrementFactor": 2,
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": 1000,
```

```
        "numberOfSucceededThings": 1000
      },
      "maximumPerMinute": 1000
    }
  }
...
}
```

其中參數：

baseRatePerMinute

指定達到 numberOfNotifiedThings 或 numberOfSucceededThings 閾值之前的任務執行速率。

incrementFactor

指定推展率在達到 numberOfNotifiedThings 或 numberOfSucceededThings 閾值之後提高的指數因數。

rateIncreaseCriteria

指定 numberOfNotifiedThings 或 numberOfSucceededThings 閾值。

中止組態

若要使用 API 新增此組態，請在執行 CreateJob 或 CreateJobTemplate API 操作時指定 AbortConfig 參數。以下範例顯示任務推展的中止組態，該組態經歷了多次失敗的執行，如 CreateJob API 操作所指定。

> **Note**
>
> 刪除任務執行會影響已完成執行總數的計算值。當任務中止時，服務會建立自動化的 comment 和 reasonCode，以區分使用者驅動的取消和任務中止的取消。

```
  "abortConfig": {
     "criteriaList": [
        {
           "action": "CANCEL",
```

```
            "failureType": "FAILED",
            "minNumberOfExecutedThings": 100,
            "thresholdPercentage": 20
        },
        {
            "action": "CANCEL",
            "failureType": "TIMED_OUT",
            "minNumberOfExecutedThings": 200,
            "thresholdPercentage": 50
        }
    ]
}
```

其中參數：

動作

指定符合中止條件時所要採取的動作。此為必要參數，且 CANCEL 是唯一有效的值。

failureType

指定應啟動任務中止的失敗類型。有效值為 FAILED、REJECTED、TIMED_OUT、ALL。

minNumberOfExecutedThings

指定在符合任務中止條件之前，必須發生的任務執行完成數量。在此範例中， AWS IoT 不會檢查是否在至少 100 個裝置完成任務執行之前發生任務中止。

thresholdPercentage

針對可以啟動任務中止的已執行任務指定物件總數。在此範例中， 會依序 AWS IoT 檢查，並在達到閾值百分比時啟動任務中止。如果在 100 次執行完成後至少有 20% 的完成執行失敗，則會取消任務推展。如果不符合此條件， AWS IoT 則 會檢查至少 50% 的已完成執行是否在 200 個執行完成後逾時。如果發生情況，即會取消任務推展。

Scheduling configuration (排程組態)

若要使用 API 新增此組態，請在執行 CreateJob 或 CreateJobTemplate API 操作時指定選用的 SchedulingConfig。

```
    "SchedulingConfig": {
      "endBehavior": string
```

```
    "endTime": string
    "maintenanceWindows": string
    "startTime": string
  }
```

其中參數：

startTime

指定任務開始的日期和時間。

endTime

指定任務開始的日期和時間。

maintenanceWindows

指定是否為排程任務選取了選用維護時段，以便將任務文件推展至目標群組中的所有裝置。`maintenanceWindow` 的字串格式是日期為 YYYY/MM/DD，而時間則為 hh:mm。

endBehavior

指定排程任務到達 `endTime` 時的任務行為。

> **Note**
>
> 可在 `DescribeJob` 和 `DescribeJobTemplate` API 中檢視任務的選擇性 `SchedulingConfig`。

逾時組態

若要使用 API 新增此組態，請在執行 `CreateJob` 或 `CreateJobTemplate` API 操作時指定 `TimeoutConfig` 參數。

使用逾時組態

1. 若要在建立任務或任務範本時設定進行中計時器，請為選用 TimeoutConfig 物件的 `inProgressTimeoutInMinutes` 屬性設定值。

```
    "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
```

```
    }
```

2. 若要針對任務執行指定步驟計時器，您可在呼叫 UpdateJobExecution 時設定 stepTimeoutInMinutes 的值。步驟計時器僅適用於您更新的任務執行。您可以在每次更新任務執行時，為此計時器設定新值。

> **Note**
>
> UpdateJobExecution 可以透過建立值為 -1 的新步驟計時器，來捨棄已建立的步驟計時器。

```
{
   ...
    "statusDetails": {
      "string" : "string"
   },
   "stepTimeoutInMinutes": number
}
```

3. 若要建立新的步驟計時器，還可以呼叫 StartNextPendingJobExecution API 操作。

重試組態

> **Note**
>
> 建立任務時，請考慮用於組態的適當重試次數。為避免因潛在的重試失敗而產生超額成本，請新增中止組態。建立任務後，即無法更新重試次數。您僅可使用 UpdateJob API 操作將重試次數設定為 0。

若要使用 API 新增此組態，請在執行 CreateJob 或 CreateJobTemplate API 操作時指定 jobExecutionsRetryConfig 參數。

```
{
...
  "jobExecutionsRetryConfig": {
      "criteriaList": [
         {
            "failureType": "string",
```

```
            "numberOfRetries": number
         }
      ]
  }
...
}
```

其中 criteriaList 是一個陣列，用於指定條件清單，可確定任務的每種失敗類型所允許的重試次數。

裝置和任務

裝置可以使用 MQTT、HTTP Signature 第 4 版或 HTTP TLS 與 AWS IoT 任務通訊。若要判斷裝置與 AWS IoT Jobs 通訊時要使用的端點，請執行 DescribeEndpoint命令。例如，如果您執行此命令：

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

您會得到類似以下的結果：

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

使用 MQTT 通訊協定

裝置可以使用 MQTT 通訊協定與 AWS IoT 任務通訊。裝置訂閱 MQTT 主題以接收新任務的通知，以及接收來自 AWS IoT Jobs 服務的回應。裝置發佈於 MQTT 主題來查詢或更新任務啟動的狀態。每項裝置皆有其一般性 MQTT 主題。如需發佈與訂閱 MQTT 主題的詳細資訊，請參閱 裝置通訊協定。

透過此通訊方法，您的裝置會使用其裝置特定的憑證和私有金鑰來驗證 AWS IoT 任務。

裝置可以訂閱下列主題。`thing-name` 是與裝置關聯的事物名稱。

- **$aws/things/*thing-name*/jobs/notify**

  訂閱此主題，即可在待定任務啟動清單中新增或移除任務啟動時通知您。
- **$aws/things/*thing-name*/jobs/notify-next**

  訂閱此主題，即可在下一個待處理任務執行項目發生變更時接收通知。
- **$aws/things/*thing-name*/jobs/*request-name*/accepted**

AWS IoT Jobs 服務會在 MQTT 主題上發佈成功和失敗訊息。主題是透過將 `accepted` 或 `rejected` 附加到用於發出請求的主題來形成。在這裡， `request-name` 是請求的名稱，例如 `Get`，而主題可以是：`$aws/things/myThing/jobs/get`. AWS IoT Jobs 接著會在`$aws/things/myThing/jobs/get/accepted`主題上發佈成功訊息。

- **`$aws/things/`*`thing-name`*`/jobs/`*`request-name`*`/rejected`**

  在此例中，`request-name` 是請求的名稱，例如 `Get`。如果請求失敗， AWS IoT Jobs 會在`$aws/things/myThing/jobs/get/rejected`主題上發佈失敗訊息。

您也可以使用下列 HTTPS API 操作：

- 藉由呼叫 `UpdateJobExecution` API 來更新任務執行的狀態。
- 藉由呼叫 `DescribeJobExecution` API 來查詢任務執行的狀態。
- 藉由呼叫 `GetPendingJobExecutions` API 來擷取待定任務執行的清單。
- 呼叫 `DescribeJobExecution` API (其中 `jobId` 作為 `$next`) 來擷取下一個待定任務執行。
- 藉由呼叫 `StartNextPendingJobExecution` API 來取得並開始下一個待定任務執行。

使用 HTTP Signature 第 4 版

裝置可以在連接埠 443 上使用 HTTP Signature 第 4 版與 AWS IoT 任務通訊。此為 AWS SDK 與 CLI 使用的方法。如需這些工具的詳細資訊，請參閱 AWS CLI 命令參考：iot-jobs-data 或 AWS SDK 與工具，並參閱您慣用語言的 IotJobsDataPlane 章節。

透過此通訊方法，您的裝置會使用 IAM 登入資料向 AWS IoT 任務進行驗證。

使用此方法可運用下列命令：

- DescribeJobExecution

  ```
  aws iot-jobs-data describe-job-execution ...
  ```

- GetPendingJobExecutions

  ```
  aws iot-jobs-data get-pending-job-executions ...
  ```

- StartNextPendingJobExecution

  ```
  aws iot-jobs-data start-next-pending-job-execution ...
  ```

- UpdateJobExecution

```
aws iot-jobs-data update-job-execution ...
```

使用 HTTP TLS

裝置可以使用支援此通訊協定的第三方軟體用戶端，在連接埠 8443 上使用 HTTP TLS 與 AWS IoT 任務通訊。

若使用此方法，您的裝置會使用 X.509 憑證型身分驗證 (例如，其裝置專屬的憑證與私有金鑰)。

使用此方法可運用下列命令：

- DescribeJobExecution
- GetPendingJobExecutions
- StartNextPendingJobExecution
- UpdateJobExecution

設定裝置與任務合作

本節中的範例使用 MQTT 說明裝置與 AWS IoT 任務服務之間如何運作。或者，您可以使用對應的 API 或 CLI 命令。在這些範例中，我們會假設名為 MyThing 的裝置將訂閱下列 MQTT 主題：

- `$aws/things/MyThing/jobs/notify` (或 `$aws/things/MyThing/jobs/notify-next`)
- `$aws/things/MyThing/jobs/get/accepted`
- `$aws/things/MyThing/jobs/get/rejected`
- `$aws/things/MyThing/jobs/jobId/get/accepted`
- `$aws/things/MyThing/jobs/jobId/get/rejected`

如果您使用 的程式碼簽署 AWS IoT，您的裝置程式碼必須驗證程式碼檔案的簽章。該簽章是包含任務文件中的 `codesign` 屬性。如需驗證程式碼檔簽章的詳細資訊，請參閱裝置代理程式範例。

主題

- 裝置工作流程
- 任務工作流程

- 任務通知

裝置工作流程

裝置可以使用下列其中一種方式來處理其執行的任務。

- 取得下一個任務

  1. 裝置首次上線時，應訂閱裝置的 `notify-next` 主題。
  2. 以 jobId `$next` 呼叫 DescribeJobExecution MQTT API 來取得下一個任務、任務文件及其他詳細資訊，包括任何儲存於 `statusDetails` 的狀態。如果任務文件具有程式碼檔簽章，則必須先驗證簽章，然後再繼續處理任務請求。
  3. 呼叫 UpdateJobExecutionMQTT API 以更新工作狀態。或者，裝置可以呼叫 StartNextPendingJobExecution，將此步驟與前一步驟結合至同一個呼叫裡。
  4. (選用) 當您呼叫 UpdateJobExecution 或 StartNextPendingJobExecution 時，您可以透過設定 `stepTimeoutInMinutes` 的值來新增步驟計時器。
  5. 執行由使用 UpdateJobExecutionMQTT API 之工作文件所指定的動作，以回報工作進度。
  6. 使用此 jobId 來呼叫 DescribeJobExecution MQTT API，即可繼續監控任務執行。如果刪除任務執行，DescribeJobExecution 會傳回 `ResourceNotFoundException`。

     如果任務執行在裝置執行任務時遭到取消或刪除，裝置應可還原為有效狀態。
  7. 任務完成時呼叫 UpdateJobExecutionMQTT API 以更新任務狀態並回報成功或失敗。
  8. 由於此任務的執行狀態已變更為結束狀態，因此下一個可執行的任務 (若有) 將會變更。狀置會收到下一個待定任務執行已變更的通知。此時，裝置應繼續如步驟 2 所述運作。

  如果裝置仍在線上，它會繼續收到下一個待定任務執行的通知。這包括其任務執行資料、它何時完成任務或何時新增待定任務執行。在此情況下，裝置應繼續如步驟 2 所述運作。

- 選取可用的任務

  1. 裝置首次上線時，應訂閱物件的 `notify` 主題。
  2. 呼叫 GetPendingJobExecutions MQTT API 來取得待定任務執行清單。
  3. 如果清單包含了一或多個任務執行項目，請選取其中一項。
  4. 呼叫 DescribeJobExecution MQTT API 來取得任務文件及其他詳細資訊，包括儲存於 `statusDetails` 的任何狀態。

5. 呼叫 UpdateJobExecutionMQTT API 以更新工作狀態。如果 `includeJobDocument` 欄位在此命令中設定為 `true`，則裝置可以略過先前的步驟並擷取當前的任務文件。
6. 或者，當您呼叫 UpdateJobExecution 時，您可以透過設定 `stepTimeoutInMinutes` 的值來新增步驟計時器。
7. 執行由使用 UpdateJobExecutionMQTT API 之工作文件所指定的動作，以回報工作進度。
8. 使用此 jobId 來呼叫 DescribeJobExecution MQTT API，即可繼續監控任務執行。如果任務執行在裝置執行任務時遭取消或刪除，裝置應可還原為有效狀態。
9. 任務完成時呼叫 UpdateJobExecutionMQTT API 以更新任務狀態並回報成功或失敗。

如果裝置維持在線上，只要有新的待定任務執行出現，裝置就會收到所有待定任務執行的通知。在此情況下，裝置可繼續如步驟 2 所述運作。

如果裝置無法實施任務，則應呼叫 UpdateJobExecution MQTT API 將任務狀態更新為 `REJECTED`。

任務工作流程

以下顯示了任務工作流程中從開始新任務到報告任務執行完成狀態的各個步驟。

開始新任務

建立新任務時， AWS IoT Jobs 會在每個目標裝置的 `$aws/things/`*`thing-name`*`/jobs/notify`主題上發佈訊息。

訊息包含下列資訊：

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

當任務執行排入佇列時，裝置就會收到這個以 `'$aws/things/`*`thingName`*`/jobs/notify'` 為主題的訊息。

> **Note**
>
> 對於設有選擇性 SchedulingConfig 的任務，任務會維持 SCHEDULED 的初始狀態。當任務達到選定的 startTime 時，會發生下列情形：
>
> - 任務狀態將更新為 IN_PROGRESS。
> - 任務會開始將任務文件推展至目標群組中的所有裝置。

取得任務資訊

若需取得更多有關任務執行的資訊，該裝置會呼叫 DescribeJobExecution MQTT API，且 includeJobDocument 欄位將設為 true (此為預設)。

如果請求成功， AWS IoT 任務服務會發佈$aws/things/MyThing/jobs/0023/get/accepted有關主題的訊息：

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

如果請求失敗， AWS IoT 任務服務會在$aws/things/MyThing/jobs/0023/get/rejected主題上發佈訊息。

裝置現在具有可用於執行任務遠端操作的任務文件。如果任務文件包含 Amazon S3 預先簽章的 URL，則裝置可以使用此 URL 下載任務需要的所有檔案。

報告任務執行狀態

裝置執行工作時，可以呼叫 UpdateJobExecutionMQTT API 以更新工作執行的狀態。

例如，裝置可以藉由以 IN_PROGRESS 主題發佈下列訊息而將任務執行狀態更新為 $aws/things/MyThing/jobs/0023/update：

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

任務會透過將訊息發佈至 $aws/things/MyThing/jobs/0023/update/accepted 或 $aws/things/MyThing/jobs/0023/update/rejected 主題進行回應：

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

裝置可以透過呼叫 StartNextPendingJobExecution 來合併前兩個請求。這會取得並開始下一個待定任務執行，並能讓裝置更新任務執行狀態。此請求也會在任務執行待定時傳回任務文件。

如果任務包含 TimeoutConfig，則進行中計時器會開始執行。您也可以設定任務執行的步驟計時器，方法是在您呼叫 UpdateJobExecution 時設定 stepTimeoutInMinutes 的值。步驟計時器僅適用於您更新的任務執行。您可以在每次更新任務執行時，為此計時器設定新值。當您呼叫 StartNextPendingJobExecution 時，您也可以建立步驟計時器。如果任務執行維持在 IN_PROGRESS 狀態的時間超過步驟計時器隔時，任務執行就會失敗，並切換到終止的 TIMED_OUT 狀態。當您建立任務時，步驟計時器不會影響您設定的進行中計時器。

status 欄位可以設定為 IN_PROGRESS、SUCCEEDED 或 FAILED。若任務執行已經處於結束狀態，您就無法更新其狀態。

報告執行完成

裝置完成執行工作時，會呼叫 UpdateJobExecutionMQTT API。如果任務成功，則將 `status` 設定為 `SUCCEEDED`，並在訊息酬載的 `statusDetails` 中以名稱值對的方式加入任務的其他相關資訊。任務執行完成時，執行中與步驟計時器會結束。

例如：

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

如果任務不成功，則將 `status`設定為 `FAILED`，並在 `statusDetails` 加入所發生錯誤的資訊：

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

> **Note**
>
> `statusDetails` 屬性可以包含任意數量的名稱值對。

當 AWS IoT Jobs 服務收到此更新時，它會在`$aws/things/MyThing/jobs/notify`主題上發佈訊息，以指出任務執行已完成：

```
{
    "timestamp":1476290692776,
    "jobs":{}
}
```

額外任務

如果裝置還有其他待定的任務執行，則會包含在發佈至 $aws/things/MyThing/jobs/notify 的訊息中。

例如：

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

任務通知

當 AWS IoT 任務待定或清單中的第一個任務執行變更時，Jobs 服務會將 MQTT 訊息發佈至預留主題。裝置可訂閱這些主題以追蹤待定的任務。

任務通知類型

任務通知會作為 JSON 承載發佈到 MQTT 主題。通知有兩種：

ListNotification

ListNotification 包含不超過 15 個待定任務執行的清單。它們會依狀態排序 (IN_PROGRESS 任務執行先於 QUEUED 任務執行)，然後依其佇列的時間排序。

每當符合以下其中一個條件，就發佈 ListNotification。

- 新的任務執行會排入佇列或變更為非終結狀態 (IN_PROGRESS 或 QUEUED)。
- 舊任務執行會變更為終止狀態 (FAILED、SUCCEEDED、CANCELED、TIMED_OUT、REJECTED 或 REMOVED)。

如需使用和不使用排程組態之限制的詳細資訊，請參閱 任務執行限制。

NextNotification

- NextNotification 包含佇列中下一個任務執行的摘要資訊。

  當清單中第一個任務執行變更時，就發佈 NextNotification。
  - 新的任務執行會新增到清單中作為 QUEUED，而且會是清單中的第一個。
  - 清單中不是第一個的現有任務執行的狀態，從 QUEUED 變更為 IN_PROGRESS，並且成為清單中的第一個。(清單中沒有其他 IN_PROGRESS 任務執行時，或者其狀態從 QUEUED 變更為 IN_PROGRESS 的任務執行比清單中任何其他 IN_PROGRESS 任務執行較早排入佇列時，就會發生這種情況)。
  - 任務執行的狀態，即清單中第一個變更為終結狀態並從清單中移除。

如需發佈與訂閱 MQTT 主題的詳細資訊，請參閱 the section called “裝置通訊協定”。

> **Note**
>
> 當您使用 HTTP Signature 第 4 版或 HTTP TLS 來與任務通訊時，通知無法使用。

任務待定

當任務新增至物件的待定任務執行清單中或從清單中的第一個任務執行變更中移除時， AWS IoT 任務服務會在 MQTT 主題上發佈訊息：

- $aws/things/*thingName*/jobs/notify
- $aws/things/*thingName*/jobs/notify-next

訊息包含的承載範例如下：

$aws/things/*thingName*/jobs/notify:

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
```

```
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : [ {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0
    } ]
  }
}
```

如果名為 this-job 的任務執行項目源於已選取選用排程組態的任務，且任務文件推出排定在維護時段期間執行，則它只會顯示在週期性維護時段期間。在維護時段之外，名為 this-job 的任務會從待定任務執行清單中排除，如下列範例所示。

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : []
  }
}
```

$aws/things/*thingName*/jobs/notify-next:

```
{
  "timestamp" : 10011,
  "execution" : {
    "jobId" : "other-job",
    "status" : "IN_PROGRESS",
    "queuedAt" : 10009,
    "lastUpdatedAt" : 10009,
    "versionNumber" : 1,
```

```
    "executionNumber" : 1,
    "jobDocument" : {"c":"d"}
  }
}
```

如果名為 other-job 的任務執行項目源於已選取選用排程組態的任務，且任務文件推出排定在維護時段期間執行，則它只會顯示在週期性維護時段期間。在維護時段之外，名為 other-job 的任務不會列為下一個任務執行，如下列範例所示。

```
{} //No other pending jobs
```

```
{
  "timestamp" : 10011,
  "execution" : {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0,
      "jobDocument" : {"a":"b"}
  }
} // "this-job" is pending next to "other-job"
```

可能的任務執行狀態為
QUEUED、IN_PROGRESS、FAILED、SUCCEEDED、CANCELED、TIMED_OUT、REJECTED 及 REMOVED。

下列一連串範例顯示在建立任務執行，且其從一個狀態變為另一個狀態時，發佈到各主題的通知。

首先，建立一個任務，稱為 job1。此通知會發佈到 jobs/notify 主題：

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      }
```

```
    ]
  }
}
```

此通知會發佈到 jobs/notify-next 主題：

```
{
  "timestamp": 1517016948,
  "execution": {
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

建立另一個任務 (job2) 時，此通知會發佈至 jobs/notify 主題：

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
```

```
}
```

通知不會發佈至 jobs/notify-next 主題，因為佇列中的下一個任務 (job1) 尚未變更。當 job1 開始執行時，它的狀態變更為 IN_PROGRESS。不會發佈通知，因為任務清單與佇列中下一個任務尚未變更。

新增第三個任務 (job3) 時，此通知會發佈至 jobs/notify 主題：

```
{
  "timestamp": 1517017906,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517017472,
        "startedAt": 1517017472,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ],
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

通知不會發佈至 jobs/notify-next 主題，因為佇列中的下一個任務仍然是 (job1)。

當 job1 完成時，它的狀態會變更為 SUCCEEDED，而且此通知會發佈至 jobs/notify 主題：

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

此時，系統會從佇列中移除 job1，而下一個要執行的任務為 job2。此通知會發佈到 jobs/notify-next 主題：

```
{
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

如果 job3 必須在 job2 前開始執行 (不建議這麼做)，可將 job3 的狀態變更為 IN_PROGRESS。這種情況下，job2 將不會是佇列中的下一個任務，而此通知會發佈至 jobs/notify-next 主題：

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

不會將通知發佈到 jobs/notify 主題，因為未新增或移除任何任務。

如果裝置拒絕 job2 並將其狀態更新為 REJECTED，此通知會發佈至 jobs/notify 主題：

```
{
  "timestamp": 1517189392,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
        "startedAt": 1517186779,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

如果 job3 (仍在進行中) 是強制刪除，此通知會發佈到 jobs/notify 主題：

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

此時，佇列是空的。此通知會發佈到 jobs/notify-next 主題：

```
{
  "timestamp": 1517189551
}
```

AWS IoT 任務 API 操作

AWS IoT 任務 API 可用於下列任一類別：

- 管理任務，例如任務的管理和控制。這是控制平面。
- 實施這些任務的裝置。這是資料平面，可讓您傳送和接收資料。

任務管理與控制會使用 HTTPS 通訊協定的 API。裝置可以使用 MQTT 或 HTTPS 通訊協定的 API。控制平面 API 適用於少量呼叫，特別是在建立與追蹤任務時使用。其通常會為單一請求開啟連線，並在接收到回應之後關閉連線。資料平面 HTTPS 和 MQTT API 允許長輪詢。這些 API 操作適用於可擴展至數百萬個裝置的大量流量。

每個 AWS IoT Jobs HTTPS API 都有對應的命令，可讓您從 AWS Command Line Interface () 呼叫 API AWS CLI。命令為小寫字母，且在組成 API 名稱的單字之間有連字號。例如，您可以輸入以下內容來呼叫 CLI 上的 CreateJob API：

```
aws iot create-job ...
```

如果操作期間發生錯誤，您會收到包含錯誤相關資訊的錯誤回應。

ErrorResponse

包含在 AWS IoT 任務服務操作期間所發生的錯誤資訊。

以下範例顯示此操作的語法：

```
{
    "code": "ErrorCode",
    "message": "string",
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
```

```
}
```

以下是 ErrorResponse 的說明：

code

ErrorCode 可設定為：

InvalidTopic

請求已傳送至 AWS IoT 任務命名空間中未對應至任何 API 操作的主題。

InvalidJson

請求的內容無法解譯為有效的 UTF-8 編碼 JSON。

InvalidRequest

請求的內容無效。例如，當 UpdateJobExecution 請求包含了無效的狀態詳細資訊，此代碼就會傳回。訊息包含錯誤的詳細資訊。

InvalidStateTransition

已嘗試更新，將任務執行變更為由於任務執行目前狀態而無效的狀態。例如，嘗試將狀態為 SUCCEEDED 的請求變更為狀態 IN_PROGRESS。在此情況下，錯誤訊息的本文也會包含 executionState 欄位。

ResourceNotFound

請求主題指定的 JobExecution 不存在。

VersionMismatch

請求中指定的預期版本與 Jobs 服務中 AWS IoT 任務執行的版本不符。在此情況下，錯誤訊息的本文也會包含 executionState 欄位。

InternalError

處理請求時發生內部錯誤。

RequestThrottled

請求受到調節。

TerminalStateReached

在處於結束狀態的任務上執行描述任務的命令時發生。

`message`

錯誤訊息字串。

`clientToken`

用於將請求與回覆建立關聯的任意字串。

`timestamp`

Epoch 時間，以秒為單位。

`executionState`

`JobExecutionState` 物件。只有當 `code` 欄位有 `InvalidStateTransition` 或 `VersionMismatch` 值，此欄位才會包含在內。在這些情況下，就不必另外執行 `DescribeJobExecution` 請求以獲得任務執行狀態資料。

以下列出任務 API 操作和資料類型。

- 任務管理、控制 API 和控制資料類型
- 任務裝置 MQTT、HTTPS API 操作和資料類型

任務管理、控制 API 和控制資料類型

以下命令可透過 HTTPS 通訊協定在 CLI 中用於任務管理和控制。

- 任務管理和控制資料類型
- 任務管理和控制 API 操作

執行這個命令，為 CLI 命令確定 *`endpoint-url`* 參數。

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

此命令會傳回下列輸出。

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

> **Note**
>
> 任務端點不支援 ALPN `x-amzn-http-ca`。
> 如果您使用的是雙堆疊端點 (IPv6 和 IPv6)，請使用`iot:Data-ATS`端點。`iot:Jobs` 端點僅支援 IPv4。

任務管理和控制資料類型

管理和控制應用程式會使用下列資料類型來與 AWS IoT 任務通訊。

任務

`Job`物件包含了工作的詳細資訊。語法如下列範例所示。

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "forceCanceled": boolean,
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "comment": "string",
    "targets": ["string"],
    "description": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp,
    "jobProcessDetails": {
        "processingTargets": ["string"],
        "numberOfCanceledThings": long,
        "numberOfSucceededThings": long,
        "numberOfFailedThings": long,
        "numberOfRejectedThings": long,
        "numberOfQueuedThings": long,
        "numberOfInProgressThings": long,
        "numberOfRemovedThings": long,
        "numberOfTimedOutThings": long
    },
    "presignedUrlConfig": {
        "expiresInSec": number,
        "roleArn": "string"
    },
```

```
    "jobExecutionsRolloutConfig": {
        "exponentialRate": {
           "baseRatePerMinute": integer,
           "incrementFactor": integer,
           "rateIncreaseCriteria": {
              "numberOfNotifiedThings": integer, // Set one or the other
              "numberOfSucceededThings": integer // of these two values.
           },
           "maximumPerMinute": integer
      }
    },
    "abortConfig": {
       "criteriaList": [
          {
             "action": "string",
             "failureType": "string",
             "minNumberOfExecutedThings": integer,
             "thresholdPercentage": integer
          }
       ]
    },
    "SchedulingConfig": {
      "startTime": string
      "endTime": string
      "timeZone": string


      "endTimeBehavior": string

   },
    "timeoutConfig": {
        "inProgressTimeoutInMinutes": long
    }
}
```

如需詳細資訊，請參閱 Job 或 job 。

JobSummary

JobSummary物件包含了工作摘要。語法如下列範例所示。

```
{
    "jobArn": "string",
    "jobId": "string",
```

```
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED|SCHEDULED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
}
```

如需詳細資訊，請參閱 JobSummary 或 job-summary 。

JobExecution

JobExecution 物件代表在裝置上執行任務。語法如下列範例所示。

> **Note**
>
> 當您使用控制平面 API 操作時，JobExecution 資料類型不包含 JobDocument 欄位。若要取得此資訊，您可以使用 GetJobDocument API 操作或 get-job-document CLI 命令。

```
{
    "approximateSecondsBeforeTimedOut": 50,
    "executionNumber": 1234567890,
    "forceCanceled": true|false,
    "jobId": "string",
    "lastUpdatedAt": timestamp,
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "forceCanceled": boolean,
    "statusDetails": {
        "detailsMap": {
            "string": "string" ...
        },
        "status": "string"
    },
    "thingArn": "string",
    "versionNumber": 123
}
```

如需詳細資訊，請參閱 JobExecution 或 job-execution 。

JobExecutionSummary

JobExecutionSummary 物件包含了任務執行摘要的資訊。語法如下列範例所示。

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

如需詳細資訊，請參閱 JobExecutionSummary 或 job-execution-summary 。

JobExecutionSummaryForJob

JobExecutionSummaryForJob物件包含了關於特定工作之工作執行的資訊摘要。語法如下列範例所示。

```
{
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
        ...
    ]
}
```

如需詳細資訊，請參閱 JobExecutionSummaryForJob 或 job-execution-summary-for-job 。

JobExecutionSummaryForThing

JobExecutionSummaryForThing物件包含了關於特定物件之工作執行的資訊摘要。語法如下列範例所示：

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

如需詳細資訊，請參閱 JobExecutionSummaryForThing 或 job-execution-summary-for-thing 。

任務管理和控制 API 操作

使用下列 API 操作或 CLI 命令：

AssociateTargetsWithJob

將群組與持續性任務建立關聯。必須符合以下條件：

- 建立任務時必須將 targetSelection 欄位設定為「CONTINUOUS」。
- 任務狀態目前必須是「IN_PROGRESS」。
- 與工作關聯的目標總數不得超過 100 個。

HTTPS request

```
POST /jobs/jobId/targets

{
"targets": [ "string" ],
"comment": "string"
}
```

如需詳細資訊，請參閱AssociateTargetsWithJob。

CLI syntax

```
aws iot  associate-targets-with-job \
--targets <value> \
--job-id <value> \
[--comment <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

如需詳細資訊，請參閱associate-targets-with-job。

CancelJob

取消任務。

HTTPS request

```
PUT /jobs/jobId/cancel

{
"force": boolean,
"comment": "string",
"reasonCode": "string"
}
```

如需詳細資訊，請參閱CancelJob。

CLI syntax

```
aws iot cancel-job \
    --job-id <value> \
```

```
    [--force <value>]  \
    [--comment <value>]  \
    [--reasonCode <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
    "jobId": "string",
    "force": boolean,
    "comment": "string"
}
```

如需詳細資訊，請參閱cancel-job。

CancelJobExecution

取消裝置上的任務執行。

HTTPS request

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
    ...
}
}
```

如需詳細資訊，請參閱CancelJobExecution。

CLI syntax

```
aws iot cancel-job-execution \
--job-id <value> \
--thing-name <value> \
[--force | --no-force] \
[--expected-version <value>] \
```

```
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

如需詳細資訊，請參閱cancel-job-execution。

CreateJob

建立任務。您可以將 Amazon S3 儲存貯體 (documentSource 參數) 或請求本文 (document 參數) 中的檔案連結提供給任務文件。

您可以將選用 targetSelection 參數設定為 CONTINUOUS (預設為 SNAPSHOT) 以讓任務「持續」執行。持續任務可以用來在新增至群組時加入或升級裝置，因為它會持續執行，並在新增的項目上啟動。即使在建立任務時群組中的物件已完成任務之後，仍可能發生這種情況。

任務可以有一個選用 TimeoutConfig，其會設定進行中計時器的值。進行中計時器無法更新，並會套用到任務的所有執行。

以下驗證會在 CreateJobAPI 的引數上執行：

- targets引數必須是有效物件或物件群組 ARN 的清單。所有物件和物件群組都必須位於您的 中 AWS 帳戶。
- documentSource 引數必須是連結至任務文件的有效 Amazon S3 URL。Amazon S3 URL 的格式為：https://s3.amazonaws.com/*bucketName*/*objectName*。
- 由 documentSource引數指定存放文件的 URL 必須是 UTF-8 編碼的 JSON 文件。
- 由於 MQTT 訊息大小限制 (128 KB) 與加密因素，因此工作文件的大小限制為 32 KB。
- 在您的 中jobId必須是唯一的 AWS 帳戶。

HTTPS request

```
PUT /jobs/jobId

{
"targets": [ "string" ],
"document": "string",
"documentSource": "string",
"description": "string",
"jobTemplateArn": "string",
"presignedUrlConfigData": {
    "roleArn": "string",
    "expiresInSec": "integer"
},
"targetSelection": "CONTINUOUS|SNAPSHOT",
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
         "thresholdPercentage": integer
      }
   ]
},
"SchedulingConfig": {
    "startTime": string
    "endTime": string
    "timeZone": string


    "endTimeBehavior": string
```

```
   }
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
}
}
```

如需詳細資訊，請參閱CreateJob。

CLI syntax

```
aws iot create-job \
    --job-id <value> \
    --targets <value> \
    [--document-source <value>] \
    [--document <value>] \
    [--description <value>] \
    [--job-template-arn <value>] \
    [--presigned-url-config <value>] \
    [--target-selection <value>] \
    [--job-executions-rollout-config <value>] \
    [--abort-config <value>] \
    [--timeout-config <value>] \
    [--document-parameters <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
    "jobId": "string",
    "targets": [ "string" ],
    "documentSource": "string",
    "document": "string",
    "description": "string",
    "jobTemplateArn": "string",
    "presignedUrlConfig": {
        "roleArn": "string",
        "expiresInSec": long
     },
    "targetSelection": "string",
    "jobExecutionsRolloutConfig": {
          "exponentialRate": {
              "baseRatePerMinute": integer,
```

```
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                   "numberOfNotifiedThings": integer, // Set one or the other
                   "numberOfSucceededThings": integer // of these two values.
                },
      "maximumPerMinute": integer
      }
    },
    "abortConfig": {
    "criteriaList": [
        {
           "action": "string",
           "failureType": "string",
           "minNumberOfExecutedThings": integer,
           "thresholdPercentage": integer
         }
      ]
    },
    "timeoutConfig": {
          "inProgressTimeoutInMinutes": long
    },
    "documentParameters": {
    "string": "string"
    }
}
```

如需詳細資訊，請參閱create-job。

DeleteJob

刪除任務及其相關的任務執行。

刪除任務可能會需要時間，根據為任務與各種其他因素所建立的任務執行數量而定。雖然任務被刪除，任務狀態將顯示為「DELETION_IN_PROGRESS」。嘗試刪除或取消狀態已為「DELETION_IN_PROGRESS」的任務將會導致錯誤。

HTTPS request

```
DELETE /jobs/jobId?force=force
```

如需詳細資訊，請參閱DeleteJob。

CLI syntax

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"force": boolean
}
```

如需詳細資訊，請參閱delete-job。

DeleteJobExecution

刪除任務執行。

HTTPS request

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

如需詳細資訊，請參閱DeleteJobExecution。

CLI syntax

```
aws iot  delete-job-execution \
--job-id <value> \
--thing-name <value> \
--execution-number <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
```

```
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

如需詳細資訊，請參閱delete-job-execution。

DescribeJob

取得任務執行的詳細資訊。

HTTPS request

```
GET /jobs/jobId
```

如需詳細資訊，請參閱DescribeJob。

CLI syntax

```
aws iot describe-job \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 格式：

```
{
"jobId": "string"
}
```

如需詳細資訊，請參閱describe-job。

DescribeJobExecution

取得工作執行的詳細資訊。任務的執行狀態必須為 SUCCEEDED或 FAILED。

HTTPS request

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

如需詳細資訊，請參閱DescribeJobExecution。

CLI syntax

```
aws iot  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long
}
```

如需詳細資訊，請參閱describe-job-execution。

GetJobDocument

取得工作的工作文件。

> **Note**
>
> 在傳回的文件中，預留位置 URL 不會被預先簽章的 Amazon S3 URL 取代。只有在 AWS IoT Jobs 服務透過 MQTT 收到請求時，才會產生預先簽章URLs。

HTTPS request

```
GET /jobs/jobId/job-document
```

如需詳細資訊，請參閱GetJobDocument。

CLI syntax

```
aws iot get-job-document \
--job-id <value>  \
```

```
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string"
}
```

如需詳細資訊，請參閱get-job-document。

ListJobExecutionsForJob

取得工作的工作執行清單。

HTTPS request

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

如需詳細資訊，請參閱ListJobExecutionsForJob。

CLI syntax

```
aws iot  list-job-executions-for-job \
--job-id <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

如需詳細資訊，請參閱list-job-executions-for-job。

ListJobExecutionsForThing

取得物件的工作執行清單。

HTTPS request

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

如需詳細資訊，請參閱ListJobExecutionsForThing。

CLI syntax

```
aws iot list-job-executions-for-thing \
--thing-name <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

如需詳細資訊，請參閱list-job-executions-for-thing。

ListJobs

取得 中任務的清單 AWS 帳戶。

HTTPS request

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thi
```

如需詳細資訊，請參閱ListJobs。

CLI syntax

```
aws iot list-jobs \
[--status <value>] \
[--target-selection <value>] \
[--max-results <value>] \
[--next-token <value>] \
[--thing-group-name <value>] \
[--thing-group-id <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"status": "string",
"targetSelection": "string",
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

如需詳細資訊，請參閱list-jobs。

UpdateJob

更新指定任務的支援欄位。timeoutConfig 的更新值僅對新進行中的啟動生效。目前，進行中的啟動會繼續以先前的逾時組態啟動。

HTTPS request

```
PATCH /jobs/jobId
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
```

```
      "baseRatePerMinute": number,
      "incrementFactor": number,
      "rateIncreaseCriteria": {
         "numberOfNotifiedThings": number,
         "numberOfSucceededThings": number
      },
   "maximumPerMinute": number
   },
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": number,
         "thresholdPercentage": number
      }
   ]
},
"timeoutConfig": {
   "inProgressTimeoutInMinutes": number
}
}
```

如需詳細資訊，請參閱UpdateJob。

CLI syntax

```
aws iot  update-job \
--job-id <value> \
[--description <value>] \
[--presigned-url-config <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
[--timeout-config <value>] \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"description": "string",
"presignedUrlConfig": {
   "expiresInSec": number,
```

```
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     }
  },
  "maximumPerMinute": number
},
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

如需詳細資訊，請參閱[update-job](#)。

任務裝置 MQTT、HTTPS API 操作和資料類型

以下命令可透過 MQTT 和 HTTPS 通訊協定執行。在執行任務之裝置的資料平面上使用這些 API 操作。

任務裝置 MQTT 和 HTTPS 資料類型

下列資料類型用於透過 MQTT 和 HTTPS 通訊協定與 AWS IoT Jobs 服務通訊。

JobExecution

`JobExecution` 物件代表在裝置上執行任務。語法如下列範例所示。

> **Note**
>
> 當您使用 MQTT 和 HTTP 資料平面 API 操作時，JobExecution 資料類型會包含 JobDocument 欄位。您的裝置可以使用此資訊從任務執行擷取任務文件。

```
{
    "jobId" : "string",
    "thingName" : "string",
    "jobDocument" : "string",
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
    },
    "queuedAt" : "timestamp",
    "startedAt" : "timestamp",
    "lastUpdatedAt" : "timestamp",
    "versionNumber" : "number",
    "executionNumber": long
}
```

如需詳細資訊，請參閱 JobExecution 或 job-execution 。

JobExecutionState

JobExecutionState 包含有關任務執行狀態的資訊。語法如下列範例所示。

```
{
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

如需詳細資訊，請參閱 JobExecutionState 或 job-execution-state 。

JobExecutionSummary

包含關於工作執行的部分資訊。語法如下列範例所示。

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

如需詳細資訊，請參閱 JobExecutionSummary 或 job-execution-summary 。

請參閱以下部分，了解更多有關 MQTT 和 HTTPS API 操作的資訊：

- 任務裝置 MQTT API 操作
- 任務裝置 HTTP API

任務裝置 MQTT API 操作

您可以發出任務裝置命令，方法為將 MQTT 訊息發佈至用於任務命令的保留主題。

您的裝置端用戶端必須訂閱這些命令的回應訊息主題。如果您使用 AWS IoT 裝置用戶端，您的裝置會自動訂閱回應主題。這表示，無論您的用戶端是否已訂閱回應訊息主題，訊息代理程式都會將回應訊息主題發佈至發佈命令訊息的用戶端。這些回應訊息不會透過訊息代理程式傳遞，而且其他用戶端或規則無法訂閱這些訊息。

訂閱機群監控解決方案的任務和 jobExecution 事件主題時，首先啟用任務和任務執行事件來接收雲端的任何事件。透過訊息代理程式處理並可由 AWS IoT 規則使用的任務進度訊息會發佈為 任務事件。由於訊息代理程式會發佈回應訊息，即使沒有明確地訂閱它們，您的用戶端也必須設定為接收並識別其所接收的訊息。您的用戶端也必須確認，在用戶端對訊息採取動作之前，內送訊息主題中的 *thingName* 套用至用戶端的物件名稱。

> **Note**
>
> 為回應 MQTT Jobs API 命令訊息而 AWS IoT 傳送的訊息會向您的 帳戶收費，無論您是否明確訂閱。

以下說明 MQTT API 操作及其請求和回應語法。所有 MQTT API 操作均有下列參數：

clientToken

用來關聯請求和回應的選用用戶端字符。在此輸入任意值，就會在回應中反映出來。

timestamp

訊息傳送的 epoch 時間，以秒為單位。

GetPendingJobExecutions

針對特定物件，取得其所有未處於終止狀態之任務的清單。

欲呼叫此 API，請發佈訊息於 $aws/things/*thingName*/jobs/get。

請求酬載：

```
{ "clientToken": "string" }
```

訊息代理程式將發佈 $aws/things/*thingName*/jobs/get/accepted 和 $aws/things/*thingName*/jobs/get/rejected，即使沒有具體訂閱它們也一樣。不過，為了讓您的用戶端接收訊息，它必須正在接聽它們。如需詳細資訊，請參閱關於任務 API 訊息的注意事項。

回應酬載：

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
}
```

inProgressJobs 和 queuedJobs 回傳狀態為 IN_PROGRESS 或 QUEUED 的 JobExecutionSummary 物件清單。

StartNextPendingJobExecution

取得並啟動物件的下一個待定任務執行 (狀態 IN_PROGRESS 或 QUEUED)。

- 狀態為 IN_PROGRESS 的任務執行會先傳回。

- 任務執行會依照其排入佇列的順序傳回。當您任務的目標群組中新增或移除物件時，請確認所有新任務執行相較於現有任務執行的推展順序。
- 如果下一個待定任務執行為 QUEUED，則其狀態會變更為 IN_PROGRESS，而任務執行狀態詳細資訊也會依指定設定。
- 如果下一個待定任務執行已經為 IN_PROGRESS，則其狀態詳細資訊不會變更。
- 如果沒有工作執行為待定，則回應不會包括 execution 欄位。
- 您也可以選擇透過設定 stepTimeoutInMinutes 的屬性來新增步驟計時器。如果您未透過執行 UpdateJobExecution 來更新此屬性的值，則步驟計時器逾期時，任務執行將逾時。

欲呼叫此 API，請發佈訊息於 $aws/things/*thingName*/jobs/start-next。

請求酬載：

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

statusDetails

名稱值對的集合，能夠描述任務執行的狀態。如果未指定，則 statusDetails 不會改變。

stepTimeOutInMinutes

指定此裝置必須完成這項任務執行的時間。在此計時器到期或計時器重設之前 (藉由呼叫 UpdateJobExecution，將狀態設為 IN_PROGRESS，並在 stepTimeoutInMinutes 欄位中指定新的逾時值)，如果任務執行狀態未設定為結束狀態，任務執行狀態就會設為 TIMED_OUT。設定此逾時不會影響任務執行逾時，該任務執行逾時可能已在建立任務時加以指定 (使用欄位 timeoutConfig 的 CreateJob)。

此參數的有效值範圍是 1 到 10080 (1 分鐘到 7 天)。值 -1 也有效，且會取消目前的步驟計時器（由先前使用的 UpdateJobExecutionRequest 所建立）。

訊息代理程式將發佈 $aws/things/*thingName*/jobs/start-next/accepted 和 $aws/things/*thingName*/jobs/start-next/rejected，即使沒有具體訂閱它們也一樣。不過，為

了讓您的用戶端接收訊息，它必須正在接聽它們。如需詳細資訊，請參閱關於任務 API 訊息的注意事項。

回應酬載：

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
"clientToken" : "string"
}
```

`execution` 為 JobExecution 物件。例如：

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

DescribeJobExecution

取得工作執行的詳細資訊。

您可以將 `jobId` 設定為 `$next`，傳回物件 (狀態為 `IN_PROGRESS` 或 `QUEUED`) 的下一個待定任務執行。

欲呼叫此 API，請發佈訊息於 `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get`。

請求酬載：

```
{
"jobId" : "022",
"thingName" : "MyThing",
```

```
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

`thingName`

與裝置關聯的物件名稱。

`jobId`

此工作在建立時被指派的獨特識別符。

或者使用 `$next` 傳回物件 (狀態為 `IN_PROGRESS` 或 `QUEUED`) 的下一個待定任務執行。在此情況下，狀態為 `IN_PROGRESS` 的任務執行會先傳回。工作執行會依照其建立的順序傳回。

`executionNumber`

(選用) 可識別在裝置上之任務執行的編號。如果未指定，則會傳回最新的工作執行。

`includeJobDocument`

(選用) 除非設為 `false`，否則回應會包含任務文件。預設值為 `true`。

訊息代理程式將發佈 `$aws/things/thingName/jobs/jobId/get/accepted` 和 `$aws/things/thingName/jobs/jobId/get/rejected`，即使沒有具體訂閱它們也一樣。不過，為了讓您的用戶端接收訊息，它必須正在接聽它們。如需詳細資訊，請參閱關於任務 API 訊息的注意事項。

回應酬載：

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

`execution` 為 JobExecution 物件。

UpdateJobExecution

更新工作執行的狀態。您可以選擇透過設定 `stepTimeoutInMinutes` 的屬性來新增步驟計時器。如果您未透過再次執行 `UpdateJobExecution` 來更新此屬性的值，則步驟計時器逾期時，任務執行將逾時。

欲呼叫此 API，請發佈訊息於 $aws/things/*thingName*/jobs/*jobId*/update。

請求酬載：

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`status`

任務執行的新狀態 (IN_PROGRESS、FAILED、SUCCEEDED 或 REJECTED)。這必須在每次更新時指定。

`statusDetails`

名稱值對的集合，能夠描述任務執行的狀態。如果未指定，則 statusDetails 不會改變。

`expectedVersion`

預期的目前工作執行版本。每次您更新工作執行，其版本編號就會增加。如果存放在 Jobs 服務中的 AWS IoT 任務執行版本不相符，則會拒絕更新並顯示VersionMismatch錯誤。也會傳回 ErrorResponse，其中包含目前任務執行狀態資料。(這樣就不必另外執行 DescribeJobExecution 請求以獲得任務執行狀態資料。)

`executionNumber`

(選用) 可識別在裝置上之任務執行的編號。如果未指定，則會使用最新的工作執行。

`includeJobExecutionState`

(選用) 若被包括在內並設定為 true，則回應會包含 JobExecutionState 欄位。預設值為 false。

`includeJobDocument`

(選用) 若被包括在內並設定為 true，則回應會包含 JobDocument。預設值為 false。

`stepTimeoutInMinutes`

指定此裝置必須完成這項任務執行的時間。如果任務執行狀態未在此計時器逾期之前，或未在計時器重設之前設定為終止狀態，任務執行狀態會設定為 `TIMED_OUT`。設定或重設此逾時不會影響任務執行逾時，該任務執行逾時可能已在建立任務時加以指定。

訊息代理程式將發佈 `$aws/things/thingName/jobs/jobId/update/accepted` 和 `$aws/things/thingName/jobs/jobId/update/rejected`，即使沒有具體訂閱它們也一樣。不過，為了讓您的用戶端接收訊息，它必須正在接聽它們。如需詳細資訊，請參閱關於任務 API 訊息的注意事項。

回應酬載：

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

`executionState`

JobExecutionState 物件。

`jobDocument`

工作文件物件。

> **Note**
>
> 在 MQTT 回應中， `jobDocument` 欄位是 JSON 物件。在 HTTP 回應中，它是 JSON 物件的字串表示法。

`timestamp`

訊息傳送的 epoch 時間，以秒為單位。

`clientToken`

用於將請求和回應建立關聯的用戶端符記。

使用 MQTT 通訊協定時，也可以執行下列更新：

JobExecutionsChanged

每當物件的待定任務執行清單新增或移除任務執行時就會傳送。

使用主題：

$aws/things/*thingName*/jobs/notify

訊息酬載：

```
{
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary ... ]
         },
    "timestamp": timestamp
}
```

NextJobExecutionChanged

每當物件待定任務清單上的下一個任務執行變更就會傳送，亦即定義 DescribeJobExecution 時加上 jobId $next。當下一個任務的執行詳細資訊變更，此訊息並不會傳送，只有在加上 jobId $next 之 DescribeJobExecution 傳回的下一個任務變更時才會傳送。考量狀態為 QUEUED 的任務執行 J1 與 J2。J1 是下一個待定任務執行。如果 J2 的狀態變更為 IN_PROGRESS 而 J1 的狀態保持不變，則會傳送此通知並包含 J2 的詳細資訊。

使用主題：

$aws/things/*thingName*/jobs/notify-next

訊息酬載：

```
{
"execution" : JobExecution,
"timestamp": timestamp,
}
```

任務裝置 HTTP API

裝置可以在連接埠 443 上使用 HTTP Signature 第 4 版與 AWS IoT 任務通訊。這是 AWS SDKs和 CLI 使用的方法。如需這些工具詳細資訊，請參閱 AWS CLI 命令參考：iot-jobs-data 或 AWS SDK 與工具。

以下命令可用於執行任務的裝置。如需如何使用 API 操作搭配 MQTT 通訊協定的詳細資訊，請參閱 任務裝置 MQTT API 操作。

GetPendingJobExecutions

針對特定物件，取得其所有未處於終止狀態之任務的清單。

HTTPS request

```
GET /things/thingName/jobs
```

回應：

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

如需詳細資訊，請參閱GetPendingJobExecutions。

CLI syntax

```
aws iot-jobs-data get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string"
}
```

如需詳細資訊，請參閱get-pending-job-executions。

StartNextPendingJobExecution

取得並啟動物件的下一個待定任務執行 (狀態 IN_PROGRESS 或 QUEUED)。

- 狀態為 IN_PROGRESS 的任務執行會先傳回。
- 工作執行會依照其建立的順序傳回。
- 如果下一個待定任務執行為 QUEUED，則其狀態會變更為 IN_PROGRESS，而任務執行狀態詳細資訊也會依指定設定。
- 如果下一個待定任務執行已經為 IN_PROGRESS，則其狀態詳細資訊不會變更。
- 如果沒有工作執行為待定，則回應不會包括 execution 欄位。
- 您也可以選擇透過設定 stepTimeoutInMinutes 的屬性來新增步驟計時器。如果您未透過執行 UpdateJobExecution 來更新此屬性的值，則步驟計時器逾期時，任務執行將逾時。

HTTPS request

請求語法如下列範例所示：

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

如需詳細資訊，請參閱StartNextPendingJobExecution。

CLI syntax

概要：

```
aws iot-jobs-data  start-next-pending-job-execution \
--thing-name <value> \
{--step-timeout-in-minutes <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

如需詳細資訊，請參閱start-next-pending-job-execution。

DescribeJobExecution

取得工作執行的詳細資訊。

您可以將 jobId 設定為 $next，傳回物件的下一個待定任務執行。任務的執行狀態必須為 QUEUED或 IN_PROGRESS。

HTTPS request

要求:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

回應：

```
{
"execution" : JobExecution,
}
```

如需詳細資訊，請參閱DescribeJobExecution。

CLI syntax

概要：

```
aws iot-jobs-data  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
```

```
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

如需詳細資訊，請參閱describe-job-execution。

UpdateJobExecution

更新工作執行的狀態。您也可以選擇透過設定 stepTimeoutInMinutes 的屬性來新增步驟計時器。如果您未透過再次執行 UpdateJobExecution 來更新此屬性的值，則步驟計時器逾期時，任務執行將逾時。

HTTPS request

要求:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

如需詳細資訊，請參閱UpdateJobExecution。

CLI syntax

概要：

```
aws iot-jobs-data  update-job-execution \
--job-id <value> \
--thing-name <value> \
--status <value> \
[--status-details <value>] \
[--expected-version <value>] \
[--include-job-execution-state | --no-include-job-execution-state] \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--step-timeout-in-minutes <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

如需詳細資訊，請參閱update-job-execution。

使用 AWS IoT 任務保護使用者和裝置

若要授權使用者搭配其裝置使用 AWS IoT 任務，您必須使用 IAM 政策授予他們許可。然後，必須使用 AWS IoT Core 政策安全地連線 AWS IoT、接收任務執行和更新執行狀態，來授權裝置。

AWS IoT 任務的必要政策類型

下表說明您必須用於授權的不同政策類型。如需所需使用政策的詳細資訊，請參閱 Authorization。

所需政策類型

使用案例	通訊協定	身分驗證	控制平面/資料平面	身分類型	所需政策類型
授權管理員、操作員或雲端服務安全使用任務	HTTPS	AWS Signature 第 4 版身分驗證 （連接埠 443)	控制平面和資料平面兩者	Amazon Cognito 身分、IAM 或聯合身分使用者	IAM 政策
授權您的 IoT 裝置安全使用任務	MQTT/HTTPS	TCP 或 TLS 相互驗證 (連接埠 8883 或 443	資料平面	X.509 憑證	AWS IoT Core 政策

若要授權可在控制平面和資料平面上執行 AWS IoT 的任務操作，您必須使用 IAM 政策。身分必須已透過 AWS IoT 驗證才能執行這些作業，必須是 Amazon Cognito 身分 或 IAM 使用者、群組和角色。如需身分驗證的相關詳細資訊，請參閱 身分驗證。

現在必須使用 AWS IoT Core 政策安全地連線至裝置閘道，在資料平面上授權裝置。裝置閘道可讓裝置安全地與 通訊 AWS IoT、接收任務執行，以及更新任務執行狀態。裝置通訊透過使用加密 MQTT 或 HTTPS 發佈 通訊協定進行加密。這些通訊協定使用 X.509 用戶端憑證提供的 AWS IoT 來驗證裝置連線。

以下顯示如何授權使用者、c 大聲服務和裝置使用 AWS IoT 任務。如需控制平面和資料平面 API 操作的資訊，請參閱 AWS IoT 任務 API 操作。

主題

- 授權使用者和雲端服務使用 AWS IoT 任務
- 授權您的裝置在資料平面上安全地使用 AWS IoT 任務

授權使用者和雲端服務使用 AWS IoT 任務

若要授權使用者和雲端任務，您必須在控制平面和資料平面上都使用 IAM 政策。這些政策必須與 HTTPS 通訊協定一起使用，並且必須使用 AWS Signature 第 4 版身分驗證 (連接埠 443) 以驗證使用者。

> **Note**
>
> AWS IoT Core 政策不得用於控制平面。IAM 政策僅用於授權使用者或雲端服務。如需使用所需政策類型的詳細資訊，請參閱 AWS IoT 任務的必要政策類型。

IAM 政策是包含政策陳述式的 JSON 文件。政策聲明使用效果、動作以及資源元素來指定資源、允許或拒絕動作，以及在何種條件下允許或拒絕動作。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

> **Warning**
>
> 我們建議您不要使用萬用字元許可，例如`"Action": ["iot:*"]`在 IAM 政策中 AWS IoT Core 。使用萬用字元許可不是推薦的安全最佳實務。如需詳細資訊，請參閱AWS IoT 過度寬鬆的政策。

控制平面上的 IAM 政策

在控制平面上，IAM 政策使用 `iot:` 字首，以授權對應的任務 API 作業。例如 `iot:CreateJob` 政策動作會授予使用者許可使用 `CreateJob` API。

政策動作

下表顯示使用 API 動作的 IAM 政策動作和許可列表。如需資源類型的資訊，請參閱 定義的資源類型 AWS IoT。如需動作的詳細資訊 AWS IoT ，請參閱 定義的動作 AWS IoT。

控制平面上的 IAM 政策動作

政策動作	API 操作	資源類型	說明
`iot:AssociateTargetsWithJob`	`AssociateTargetsWithJob`	• job • 物件 • thinggroup	代表將群組與連續任務關聯的許可。每次發出關聯目標的要求時，就會檢查一次 `iot:AssociateTargetsWithJob` 許可。

政策動作	API 操作	資源類型	說明
iot:CancelJob	CancelJob	job	代表取消任務的許可。每次發出取消任務的要求時，就會檢查一次 iot:CancelJob 許可。
iot:CancelJobExecution	CancelJobExecution	• job • 物件	代表可取消任務執行的許可。每次發出取消任務執行的要求時，就會檢查一次 iot:CancelJobExecution 許可。
iot:CreateJob	CreateJob	• job • 物件 • thinggroup • jobtemplate • package	代表建立任務的許可。每次發出建立任務的要求時，就會檢查一次 iot: CreateJob 許可。
iot:CreateJobTemplate	CreateJobTemplate	• job • jobtemplate • package	代表建立任務範本的許可。每次發出建立任務範本的要求時，就會檢查一次 iot:CreateJobTemplate 許可。
iot:DeleteJob	DeleteJob	job	代表刪除任務的許可。每次發出刪除任務的要求時，就會檢查一次 iot: DeleteJob 許可。
iot:DeleteJobTemplate	DeleteJobTemplate	jobtemplate	代表刪除任務範本的許可。每次發出刪除任務範本的要求時，就會檢查一次 iot:CreateJobTemplate 許可。
iot:DeleteJobExecution	DeleteJobTemplate	• job • 物件	代表可刪除任務執行的許可。每次發出刪除任務執行的要求時，就會檢查一次 iot:DeleteJobExecution 許可。

政策動作	API 操作	資源類型	說明
iot:DescribeJob	DescribeJob	job	代表描述任務的許可。每次發出描述任務的要求時，就會檢查一次 iot: DescribeJob 許可。
iot:DescribeJobExecution	DescribeJobExecution	• job • 物件	代表描述任務執行的許可。每次發出描述任務執行的要求時，就會檢查一次 iot: DescribeJobExecution 許可。
iot:DescribeJobTemplate	DescribeJobTemplate	jobtemplate	代表描述任務範本的許可。每次發出描述任務範本的要求時，就會檢查一次 iot: DescribeJobTemplate 許可。
iot:DescribeManagedJobTemplate	DescribeManagedJobTemplate	jobtemplate	代表描述受管任務範本的許可。每次發出描述受管任務範本的要求時，就會檢查一次 iot: DescribeManagedJobTemplate 許可。
iot:GetJobDocument	GetJobDocument	job	代表取得任務之任務文件的許可。每次發出取得任務文件的要求時，就會檢查一次 iot:GetJobDocument 許可。
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	job	代表列出任務之任務執行的許可。每次發出列出任務之任務執行的要求時，就會檢查一次 iot:ListJobExecutionsForJob 許可。
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	物件	代表列出任務之任務執行的許可。每次發出列出物件之任務執行的要求時，就會檢查一次 iot:ListJobExecutionsForThing 許可。
iot:ListJobs	ListJobs	無	代表可列出任務的許可。每次發出列出任務的要求時，就會檢查一次 iot:ListJobs 許可。

政策動作	API 操作	資源類型	說明
iot:ListJobTemplates	ListJobTemplates	無	代表可列出任務範本的許可。每次發出列出任務範本的要求時，就會檢查一次 iot:ListJobTemplates 許可。
iot:ListManagedJobTemplates	ListManagedJobTemplates	無	代表可列出受管任務範本的許可。每次發出列出受管任務範本的要求時，就會檢查一次 iot:ListManagedJobTemplates 許可。
iot:UpdateJob	UpdateJob	job	代表可更新任務的許可。每次發出更新任務的要求時，就會檢查一次 iot:UpdateJob 許可。
iot:TagResource	TagResource	• job • jobtemplate • 物件	准許標記指定的資源。
iot:UntagResource	UntagResource	• job • jobtemplate • 物件	准許取消標記指定的資源。

基本 IAM 政策範例

以下範例顯示一個 IAM 政策，該政策允許使用者對您的 IoT 物件和物件群組執行以下動作。

在該範例中，替換：

- *##* AWS 區域，例如 us-east-1。
- *account-id* 與您的 AWS 帳戶 數字，例如 57EXAMPLE833。
- *thing-group-name*換成您目標任務的 IoT 物件群組名稱，例如 FirmwareUpdateGroup。
- *thing-name*換成您目標任務的 IoT 物件名稱，例如 MyIoTThing。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/thing-group-
name"
        },
        {
            "Action": [
                "iot:DescribeJob",
                "iot:CancelJob",
                "iot:DeleteJob"
            ],
            "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:job/*"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
                "arn:aws:iot:us-east-1:123456789012:job/*"
            ]
        }
    ]
}
```

以 IP 為基礎的授權的 IAM 政策範例

您可以限制 principals (主體) 從特定 IP 地址對您的控制平面端點進行 API 呼叫。若要指定允許的 IP 地址，請在 IAM 政策的 Condition 元素中使用 aws:SourceIp 全域條件金鑰。

使用此條件金鑰也可以拒絕代表您進行這些 API 呼叫存取其他 AWS 服務，例如 AWS CloudFormation。若要允許存取這些服務，請使用 aws:ViaAWSService 有 aws:SourceIp 金鑰的全域條件金鑰。這可確保來源 IP 地址存取限制僅適用於主體的直接請求。如需詳細資訊，請參閱AWS：AWS 根據來源 IP 拒絕對 的存取。

下列範例顯示如何僅允許可對控制平面端點進行 API 呼叫的特定 IP 地址。此 aws:ViaAWSService 金鑰為 true，允許其他服務代表您進行 API 呼叫。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                 "IpAddress": {
                     "aws:SourceIp": "123.45.167.89"
             },
                 "Bool": {
                     "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

資料平面上的 IAM 政策

資料平面上的 IAM 政策使用 iotjobsdata: 字首來授權使用者可以執行的任務 API 操作。在資料平面上，您可以使用 iotjobsdata:DescribeJobExecution 政策動作授予使用者使用 DescribeJobExecution API 的許可。

> **Warning**
>
> 鎖定裝置的 AWS IoT 任務時，不推薦使用在資料平面上使用 IAM 政策。建議您在控制平面上使用 IAM 政策，以便使用者建立和管理任務。在資料平面上，若要授權裝置擷取任務執行並更新執行狀態，請使用 AWS IoT Core HTTPS 通訊協定的 政策。

基本 IAM 政策範例

必須授權的 API 操作通常由您輸入 CLI 命令來執行。以下說明使用者執行 `DescribeJobExecution` 操作的範例。

在該範例中，替換：

- *##* AWS 區域，例如 `us-east-1`。
- *account-id* 與您的 AWS 帳戶 數字，例如 `57EXAMPLE833`。
- *thing-name*換成您目標任務的 IoT 物件名稱，例如 `myRegisteredThing`。
- *job-id* 是使用 API 鎖定之任務的唯一識別符。

```
aws iot-jobs-data describe-job-execution \
    --endpoint-url "https://account-id.jobs.iot.region.amazonaws.com" \
    --job-id jobID --thing-name thing-name
```

以下顯示授權這個動作的 IAM 政策範例：

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iotjobsdata:DescribeJobExecution"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
    }
}
```

以 IP 為基礎的授權的 IAM 政策範例

您可以限制 principal (主體) 從特定 IP 地址對資料平面端點進行 API 呼叫。若要指定允許的 IP 地址，請在 IAM 政策的 Condition 元素中使用 aws:SourceIp 全域條件金鑰。

使用此條件金鑰也可以拒絕代表您進行這些 API 呼叫存取其他 AWS 服務，例如 AWS CloudFormation。若要允許存取這些服務，請使用 aws:ViaAWSService 有 aws:SourceIp 條件金鑰的全域條件金鑰。這可確保 IP 地址存取限制僅適用於主體直接發出的要求。如需詳細資訊，請參閱AWS： AWS 根據來源 IP 拒絕對 的存取。

下列範例顯示如何僅允許可對資料平面端點進行 API 呼叫的特定 IP 地址。

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iotjobsdata:*"
        ],
        "Resource": [
            "*"
        ],
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "123.45.167.89"
        },
            "Bool": {
                "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

下列範例顯示如何限制特定 IP 地址或地址範圍，以便對資料平面端點進行 API 呼叫。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
```

```
            "Action": [
                "iotjobsdata:*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "123.45.167.89",
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
            }
        },
            "Resource": [
                "*"
            ]
        }
    ]
}
```

控制平面和資料平面的 IAM 政策範例

如果您同時在控制平面和資料平面上執行 API 操作，控制平面政策動作必須使用 `iot:` 字首，而且您的資料平面政策動作必須使用 `iotjobsdata:` 字首。

例如 `DescribeJobExecution` API 可以同時在控制平面和資料平面上使用。在控制平面上，DescribeJobExecution API 用於描述任務執行。在資料平面上，DescribeJobExecution API 用於取得任務執行的詳細資訊。

以下 IAM 政策授權使用者同時在控制平面和資料平面上使用 `DescribeJobExecution` API。

在該範例中，替換：

- *##* AWS 區域，例如 `us-east-1`。
- *account-id* 與您的 AWS 帳戶 數字，例如 `57EXAMPLE833`。
- *thing-name*換成您目標任務的 IoT 物件名稱，例如 `MyIoTThing`。

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Action": [
                "iotjobsdata:DescribeJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
    "arn:aws:iot:us-east-1:123456789012:job/*"
            ]
        }
    ]
}
```

授權對 IoT 資源進行標記

為了更妥善控制您可以建立、修改或使用的任務和任務範本，您可以將標籤連接到任務或任務範本。標籤也可以幫助您識別所有權，並將其放在計費群組中並附加標籤來指派和分配成本。

當使用者想要標記他們使用 或 建立的任務 AWS 管理主控台 或任務範本時 AWS CLI，您的 IAM 政策必須授予使用者標記它們的許可。要授予權限，您的 IAM 政策必須使用 `iot:TagResource` 動作。

> **Note**
>
> 如果您的 IAM 政策不包含 `iot:TagResource`動作，則`CreateJobTemplate`具有標籤的任何 `CreateJob`或 都會傳回`AccessDeniedException`錯誤。

當您想要標記使用 或 建立的任務 AWS 管理主控台 或任務範本時 AWS CLI，IAM 政策必須授予標記這些任務的許可。要授予權限，您的 IAM 政策必須使用 `iot:TagResource` 動作。

如需標記資源的相關資訊，請參閱 標記您的 AWS IoT 資源。

IAM 政策範例

請參閱下列授予標記許可的 IAM 政策範例：

範例 1

執行下列命令以建立任務並將其標記至特定環境的使用者。

在此範例中，取代：

- *##* AWS 區域，例如 us-east-1。
- *account-id* 與您的 AWS 帳戶 數字，例如 57EXAMPLE833。
- *thing-name*換成您目標任務的 IoT 物件名稱，例如 MyIoTThing。

```
aws iot create-job
    --job-id test_job
    --targets "arn:aws:iot:region:account-id:thing/thingOne"
    --document-source "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json"
    --description "test job description"
    --tags Key=environment,Value=beta
```

這個範例必須使用下列 IAM 政策：

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateJob",
            "iot:CreateJobTemplate",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": [
           "arn:aws:iot:us-east-1:123456789012:job/*",
           "arn:aws:iot:us-east-1:123456789012:jobtemplate/*"
        ]
    }
}
```

授權您的裝置在資料平面上安全地使用 AWS IoT 任務

若要授權您的裝置與資料平面上的 AWS IoT 任務安全地互動，您必須使用 AWS IoT Core policies. AWS IoT Core policies for job 是包含政策陳述式的 JSON 文件。這些政策也使用效果、動作以及資源元素，並遵循與 IAM 政策類似的慣例。如需這些元素的詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

這些政策可以與 MQTT 和 HTTPS 通訊協定搭配使用，並且必須使用 TCP 或 TLS 相互驗證對裝置進行身分驗證。以下介紹如何跨不同的通訊協定使用這些政策。

> **Warning**
>
> 我們建議您不要使用萬用字元許可，例如`"Action": ["iot:*"]`在 IAM 政策中 AWS IoT Core 。使用萬用字元許可不是推薦的安全最佳實務。如需詳細資訊，請參閱AWS IoT 過度寬鬆的政策。

AWS IoT Core MQTT 通訊協定的 政策

AWS IoT Core MQTT 通訊協定的 政策授予您使用任務裝置 MQTT API 動作的許可。MQTT API 操作用於處理為任務命令保留的 MQTT 主題。如需有關這些 API 操作的詳細資訊，請參閱 任務裝置 MQTT API 操作。

MQTT 政策使用政策動作 (例如 `iot:Connect`、`iot:Publish`、`iot:Subscribe` 以及 `iot:Receieve`) 以處理任務主題。這些政策可讓您連線至訊息代理程式、訂閱任務 MQTT 主題，以及在您的裝置和雲端之間傳送和接收 MQTT 訊息。如需這些動作的詳細資訊，請參閱《AWS IoT Core 政策動作》。

如需 AWS IoT 任務主題的詳細資訊，請參閱 任務主題。

基本 MQTT 政策範例

以下範例說明如何使用 `iot:Publish` 和 `iot:Subscribe` 發佈和訂閱任務和任務執行。

在該範例中，替換：

- *##* AWS 區域，例如 `us-east-1`。
- *account-id* 與您的 AWS 帳戶 數字，例如 `57EXAMPLE833`。
- *thing-name*換成您目標任務的 IoT 物件名稱，例如 `MyIoTThing`。

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/job/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/jobExecution/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thing-123/jobs/*"
            ]
        }
    ]
}
```

AWS IoT Core HTTPS 通訊協定的 政策

AWS IoT Core 資料平面上的 政策也可以使用 HTTPS 通訊協定搭配 TLS 身分驗證機制來授權您的裝置。資料平面上的政策使用 `iotjobsdata:` 字首來授權您的裝置可以執行的任務 API 操作。例如 `iotjobsdata:DescribeJobExecution` 政策動作會授予使用者許可使用 `DescribeJobExecution` API。

> **Note**
>
> 資料平面政策動作必須使用 `iotjobsdata:` 字首。在控制平面上，動作必須使用 `iot:` 字首。有關同時使用控制平面和資料平面政策動作時的 IAM 政策範例，請參閱 控制平面和資料平面的 IAM 政策範例。

政策動作

下表顯示授權裝置使用 API 動作 AWS IoT Core 的政策動作和許可清單。如需您可以在資料平面中執行的 API 操作清單，請參閱 任務裝置 HTTP API。

> **Note**
>
> 任務執行政策動作僅適用於 HTTP TLS 端點。如果您使用 MQTT 端點，必須使用前述所定義的 MQTT 政策動作。

AWS IoT Core 資料平面上的政策動作

政策動作	API 操作	資源類型	說明
iotjobsdata:DescribeJobExecution	DescribeJobExecution	• job • 物件	代表可擷取任務執行的許可。每次發出擷取任務執行的要求時，就會檢查一次 iotjobsdata:DescribeJobExecution 許可。
iotjobsdata:GetPendingJobExecutions	GetPendingJobExecutions	物件	代表可擷取任務 (對物件來說並非結束狀態) 之清單的許可。每次發出擷取清單的要求時，就會檢查一次 iotjobsdata:GetPendingJobExecutions 許可。
iotjobsdata:StartNextPendingJobExecution	StartNextPendingJobExecution	物件	代表可取得並啟動物件之下一個待定任務執行的許可。也就是將任務執行狀態由 QUEUED 更新為 IN_PROGRESS 。每次發出啟動下一個待定任務執行的要求時，就會檢查一次 iotjobsdata:StartNextPendingJobExecution 許可。
iotjobsdata:UpdateJobExecution	UpdateJobExecution	物件	代表可更新任務執行的許可。每次發出更新任務執行狀態的要求時，就會檢查一次 iotjobsdata:UpdateJobExecution 許可。

基本政策範例

以下顯示 AWS IoT Core 政策的範例，該政策授予對任何資源的資料平面 API 操作執行動作的許可。您可以將政策範圍限定為特定資源，例如 IoT 物件。在您的範例中，替換：

- *##*， AWS 區域 例如 us-east-1。
- *account-id* 與您的 AWS 帳戶 數字，例如 57EXAMPLE833。
- *thing-name* 換成 IoT 物件名稱，例如 MyIoTthing。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        }
    ]
}
```

對於必須使用這些政策的時機，一個範例是當 IoT 裝置使用 AWS IoT Core 政策來存取這些 API 操作的其中之一時，例如以下的 DescribeJobExecution API 範例：

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument&namespaceId=namespaceId
 HTTP/1.1
```

AWS IoT 任務限制

AWS IoT 任務具有服務配額或限制，對應於 的服務資源或操作數目上限 AWS 帳戶。

主題

- 任務執行限制

- 作用中和並行任務限制

任務執行限制

本節提供有關 任務執行限制的資訊 AWS IoT Device Management。

> **Note**
>
> 這些限制不屬於您可以在 AWS IoT Device Management Service Quotas 文件中找到的服務配額。

若要取得待定任務執行數量的相關資訊，您可以使用 GetPendingJobExecutions API，或訂閱 AWS IoT 任務的 MQTT 預留主題並接收 任務通知類型。

您帳戶中待定任務執行的數量可能有所不同，取決於您是否已啟用排程組態並使用週期性維護時段。

待定任務執行的數量上限

API/通知名稱	說明	沒有排程組態	使用排程組態
ListNotification	每當舊任務執行進入終端狀態，或新任務執行排入佇列或變更為非終端狀態時，ListNotification 就會發佈。它最多可顯示 15 個 QUEUED或 的待定任務執行IN_PROGRESS 。	10	15（在維護時段ListNotification 期間，最多只會在 中顯示 5 個任務執行）。
GetPendingJobExecutions	當您叫用 GetPendingJobExecutions API 時，它會傳回尚未啟動的任務執行清單，而且可以在 API 呼叫之後啟動。API 最多可傳回 10 個待定任務執行。 • 在 10 個待定任務執行中，IN_PROGRESS 將從結果中篩選執行。 • 在 10 個待定任務執行中，如果任務處於 SCHEDULED 狀態，則會從結果中篩選它們。	10	15

作用中和並行任務限制

本區段將幫助您深入瞭解作用中和並行任務，以及適用於這些任務的限制。

作用中任務和作用中任務限制

當您使用 AWS IoT 主控台或 `CreateJob` API 建立任務時，任務狀態會變更為 `IN_PROGRESS`。所有進行中任務均為作用中任務，計入作用中任務限制。這包括正在推展新任務執行的任務，或正在等待裝置完成任務執行的任務。此限制適用於連續任務和快照任務。

並行任務和任務並行限制

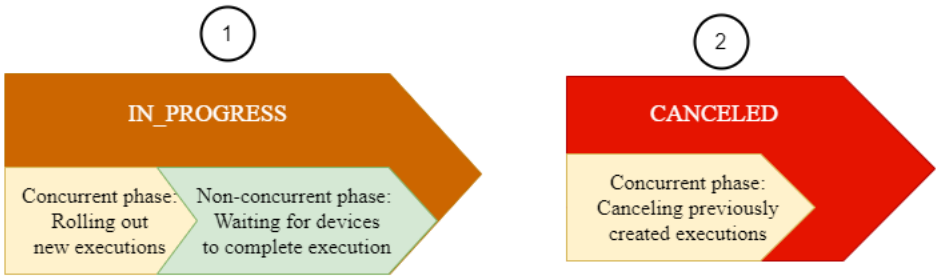
要推出新任務執行的進行中任務，或是取消先前建立任務執行的任務，都是並行任務並計入任務並行限制。 AWS IoT 任務可以每分鐘 1000 個裝置的速度快速推出和取消任務執行。每個任務為 `concurrent` 且只會短時間計入任務並行限制。在任務執行完成推展或取消後，任務不再是並行任務，不計入任務並行限制。您可以使用任務並行建立大量任務，同時等待裝置完成任務執行。

> **Note**
>
> 如果具有選用排程組態的任務和排定在維護時段期間執行的任務文件推展達到選取的 `startTime`，但您達到最大任務並行限制，則該排定的任務將會移至 `CANCELED` 的狀態。

若要判斷任務是否並行，您可以從 AWS IoT 主控台使用任務的 `IsConcurrent` 屬性，或使用 `DescribeJob`或 `ListJob` API。此限制適用於連續任務和快照任務。

若要檢視作用中任務和任務並行限制和其他任務 AWS IoT 配額， AWS 帳戶 以及請求提高限制，請參閱《》中的 AWS IoT 裝置管理端點和配額 AWS 一般參考。

下圖顯示如何將任務並行套用至進行中的任務以及取消中的任務。



> **Note**
>
> 具有選擇性 SchedulingConfig 的新任務將維持 SCHEDULED 的初始狀態，並在達到選取的 startTime 時更新為 IN_PROGRESS。在具有選擇性 SchedulingConfig 的新任務達到所選 startTime 並更新為 IN_PROGRESS 之後，它會計入作用中任務限制以及任務並行限制。狀態為 SCHEDULED 的任務將計入作用中任務限制，但不會計入任務並行限制。

下表顯示套用至作用中和並行任務的限制，以及任務狀態的並行和非並行階段。

作用中和並行任務限制

任務狀態	階段	作用中任務限制	任務並行限制
SCHEDULED	非並行階段： AWS IoT Jobs 會等待排定的startTime 任務開始對您的裝置發出任務執行通知。此階段中的任務僅計入作用中任務限制，且 IsConcurrent 屬性設置為 false。	適用	不適用
IN_PROGRESS	並行階段： AWS IoT Jobs 接受建立任務的請求，並開始將任務執行通知推展至您的裝置。此階段中的任務為並行任務，如同 IsConcurrent 屬性設置為 true 所示，並計入作用中任務和任務並行限制。	適用	適用
	非並行階段： AWS IoT Jobs 會等待裝置報告其任務執行的結果。此階段中的任務僅計入作用中任務限制，且 IsConcurrent 屬性設置為 false。	適用	不適用
Canceled	Concurrent phase： AWS IoT Jobs 接受取消任務的請求，並開始取消先前為您的裝置建立的任務執行。此階段中的任務為並行任務，且 IsConcurrent 屬性設置為 true。一旦取消任務和任務執行，任	不適用	適用

任務狀態	階段	作用中任務限制	任務並行限制
	務將不再是並行任務，且不計入任務並行限制。		

> **Note**
>
> 週期性維護時段的最長持續時間為 23 小時 50 分鐘。

AWS IoT Device Management 命令

> **Important**
>
> 本文件說明如何在 中使用命令功能 AWS IoT Device Management。如需針對 使用此功能的詳細資訊 AWS IoT FleetWise，請參閱遠端命令。
> 您全權負責以安全且符合相關法律的方式部署命令。如需 責任的詳細資訊，請參閱 AWSAWS IoT 服務條款。

使用 AWS IoT Device Management 命令將指示從雲端傳送至連線至 的裝置 AWS IoT。命令一次以一個裝置為目標，可用於低延遲、高輸送量的應用程式，例如擷取裝置端日誌或啟動裝置狀態變更。

命令是由 管理的可重複使用資源 AWS IoT Device Management。其中包含在發佈至裝置之前套用的組態。您可以針對特定使用案例預先定義一組命令，例如開啟燈泡或解鎖車輛門。

AWS IoT 命令功能可讓您：

- 使用靜態或動態承載建立可重複使用的命令範本，然後在特定裝置上執行這些範本。
- 目標裝置已註冊為 AWS IoT 實物或未註冊的 MQTT 用戶端。
- 在同一裝置上同時執行多個命令。
- 啟用事件通知，並在裝置處理命令時追蹤執行狀態。

下列主題說明如何建立命令、將命令傳送至您的裝置，以及擷取裝置報告的狀態。

主題

- 快速入門
- 命令概念和狀態
- 高階命令工作流程
- 建立和管理命令
- 開始和監控命令執行
- 取代指令資源

快速入門

完成以下步驟以傳送您的第一個命令：

1. 先決條件 - 確保您已加入 IoT Core、連線至 IoT Core 的裝置、 Commands API 操作的 IAM 許可，以及安裝在裝置上的 MQTT 用戶端程式庫。
2. 建立命令 - 定義具有靜態承載的命令，或指定裝置動作的承載範本。請參閱 建立命令資源。
3. 識別您的裝置 - 將您的裝置註冊為 IoT 物件或記下其 MQTT 用戶端 ID。請參閱 目標裝置考量事項。
4. 設定許可 - 設定 IAM 政策，允許您的裝置接收命令並發佈回應。
5. 訂閱主題 - 設定您的裝置以訂閱命令請求和回應主題。請參閱 為您的命令選擇目標裝置並訂閱 MQTT 主題。
6. 執行命令 - 在目標裝置上啟動命令執行。請參閱 啟動命令執行。

常見使用案例：

- 遠端診斷 - 擷取日誌、執行診斷、檢查裝置運作狀態
- 組態更新 - 變更設定、更新參數、切換功能
- 裝置控制 - 鎖定/解鎖、開啟/關閉電源、模式變更

命令概念和狀態

使用 AWS IoT 命令將指示從雲端傳送至連線的裝置。若要使用此功能：

1. 建立具有承載的命令，其中包含在裝置上執行 所需的組態。
2. 指定將接收承載並執行動作的目標裝置。
3. 在目標裝置上執行 命令並擷取狀態資訊。若要疑難排解問題，請參閱 CloudWatch 日誌。

如需有關此工作流程的詳細資訊，請參閱 高階命令工作流程。

主題

- 命令關鍵概念
- 命令狀態
- 命令執行狀態

命令關鍵概念

下列重要概念可協助您了解 命令功能。本文件中會持續使用術語：

- 命令 - 可重複使用的範本，定義裝置指示
- 執行 - 在裝置上執行之命令的執行個體
- 物件名稱 - 在 IoT 登錄檔中註冊之裝置的識別符
- 用戶端 ID - 未註冊裝置的 MQTT 識別符
- 承載 - 傳送至裝置的指令資料
- 主題 - 命令通訊的 MQTT 頻道

命令

命令是以 MQTT 訊息形式從雲端傳送至 IoT 裝置的指示。收到承載後，裝置會處理指示並採取對應的動作，例如修改組態設定、傳輸感應器讀數或上傳日誌。裝置接著會將結果傳回至雲端，以啟用遠端監控和控制。

命名空間

建立命令時，請指定其命名空間。對於 AWS IoT Device Management 命令，請使用預設`AWS-IoT`命名空間，並提供承載或 payloadTemplate。對於 AWS IoT FleetWise 命令，請使用 `AWS-IoT-FleetWise` 命名空間。如需詳細資訊，請參閱《 AWS IoT FleetWise 開發人員指南》中的遠端命令。

承載

建立命令時，請提供定義裝置必須執行之動作的靜態承載。承載可以使用任何支援的格式。為了確保裝置正確解譯承載，我們建議您指定承載格式類型。使用 MQTT5 通訊協定的裝置可以遵循 MQTT 標準來識別格式。JSON 或 CBOR 的格式指標可在命令請求主題中使用。

承載範本

承載範本會根據您提供的參數值，使用在執行時間產生不同承載的預留位置來定義命令承載。例如，不要為不同的溫度值建立單獨的承載，而是建立一個具有溫度預留位置的範本，並在執行期間指定該值。這可消除維護多個類似的承載。

目標裝置

若要執行命令，請使用物件名稱（適用於向 註冊的裝置 AWS IoT) 或 MQTT 用戶端 ID（適用於未註冊的裝置）指定目標裝置。用戶端 ID 是在用來連接裝置的MQTT通訊協定中定義的唯一識別符 AWS IoT。如需詳細資訊，請參閱目標裝置考量事項。

命令主題

在執行命令之前，裝置必須訂閱命令請求主題。當您執行命令時，承載會傳送至本主題上的裝置。在執行之後，裝置會將結果和狀態發佈至命令回應主題。如需詳細資訊，請參閱命令主題。

命令執行

執行是在目標裝置上執行之命令的執行個體。當您開始執行時，承載會交付至裝置，並產生唯一的執行 ID。裝置會執行 命令並報告進度 AWS IoT。裝置端邏輯會決定保留主題的執行行為和狀態報告。

參數值條件

使用承載範本建立命令時，請定義值條件，以在執行之前驗證參數值。值條件可確保參數符合要求，防止無效執行。

CommandParameterValue 類型支援的運算子

數值類型 (INTEGER、LONG、DOUBLE、UnSIGNEDLONG)

- EQUALS - 值必須等於指定的數字
- NOT_EQUALS - 值不得等於指定的數字
- GREATER_THAN - 值必須大於指定的數字
- GREATER_THAN_EQUALS - 值必須大於或等於指定的數字
- LESS_THAN - 值必須小於指定的數字
- LESS_THAN_EQUALS - 值必須小於或等於指定的數字
- IN_RANGE - 值必須在指定的範圍內 （包含）
- NOT_IN_RANGE - 值必須超出指定的範圍 （包含）
- IN_SET - 值必須符合其中一個指定的數字
- NOT_IN_SET - 值不得與任何指定的數字相符

字串類型 (STRING)

- EQUALS - 值必須等於指定的字串
- NOT_EQUALS - 值不得等於指定的字串
- IN_SET - 值必須符合其中一個指定的字串
- NOT_IN_SET - 值不得與任何指定的字串相符

布林值類型

- 不支援值條件

二進位類型

- 不支援值條件

範例：溫度控制命令

```
{
  "commandId": "SetTemperature",
  "namespace": "AWS-IoT",
  "payloadTemplate": "{\"temperature\":
 \"${aws:iot:commandexecution::parameter:temperature}\"}",
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {
            "numberRange": {
              "min": "60",
              "max": "80"
            }
          }
        }
      ]
    }
  ]
}
```

在此範例中，`temperature` 參數必須介於 60 到 80（含）之間。值超出此範圍的執行請求會失敗驗證。

> **Note**
>
> 值條件會在叫用 StartCommandExecution API 時評估。失敗的驗證會傳回錯誤，並防止建立執行。

參數值優先順序和評估

使用承載範本啟動命令執行時，會使用下列優先順序解析參數值：

1. 執行請求參數 - StartCommandExecution請求中提供的值具有最高優先順序
2. 命令預設值 - 如果執行請求中未提供參數，defaultValue則會使用 參數的
3. 無值 - 如果未提供，則執行會失敗，因為產生執行請求所需的參數

值條件會在建立執行之前的優先順序上，根據上述衍生的最終參數值進行評估。如果驗證失敗，執行請求會傳回錯誤。

範例：使用 的 SetTemperature 命令 **defaultValue**

```
{
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "defaultValue": {"I": 72},
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {"numberRange": {"min": "60", "max": "80"}}
        }
      ]
    }
  ]
}
```

開始執行時：

- 如果您在請求"temperature": {"I": 75}中提供 ，則會使用 75
- 如果您省略溫度參數，則會使用預設值 72
- 這兩個值都會根據範圍條件 【60，80】 進行驗證

命令狀態

您 中的命令 AWS 帳戶 可以處於三種狀態之一：可用、已棄用或待刪除。

可用性

成功建立後，命令會處於可用狀態，並且可以在裝置上執行。

已棄用

不再需要時，標記用於取代的命令。已棄用的命令無法啟動新的執行，但待定執行會繼續完成。若要啟用新的執行，請將 命令還原為可用狀態。

待刪除

當您將命令標記為刪除時，如果已取代超過最大逾時 （預設值：12 小時），則會自動將其刪除。此動作為永久性。如果超過逾時的時間未棄用或棄用，則命令會進入待定刪除狀態，並在逾時過期後移除。

命令執行狀態

當您在目標裝置上啟動執行時，它會進入 CREATED 狀態，並根據裝置報告轉換為其他狀態。您可以擷取狀態資訊和追蹤執行。

> **Note**
>
> 您可以在裝置上同時執行多個命令。使用並行控制來限制每個裝置的執行，並防止過載。如需每個裝置的並行執行上限，請參閱AWS IoT Device Management 命令配額。

下表顯示根據執行進度的執行狀態及其轉換。

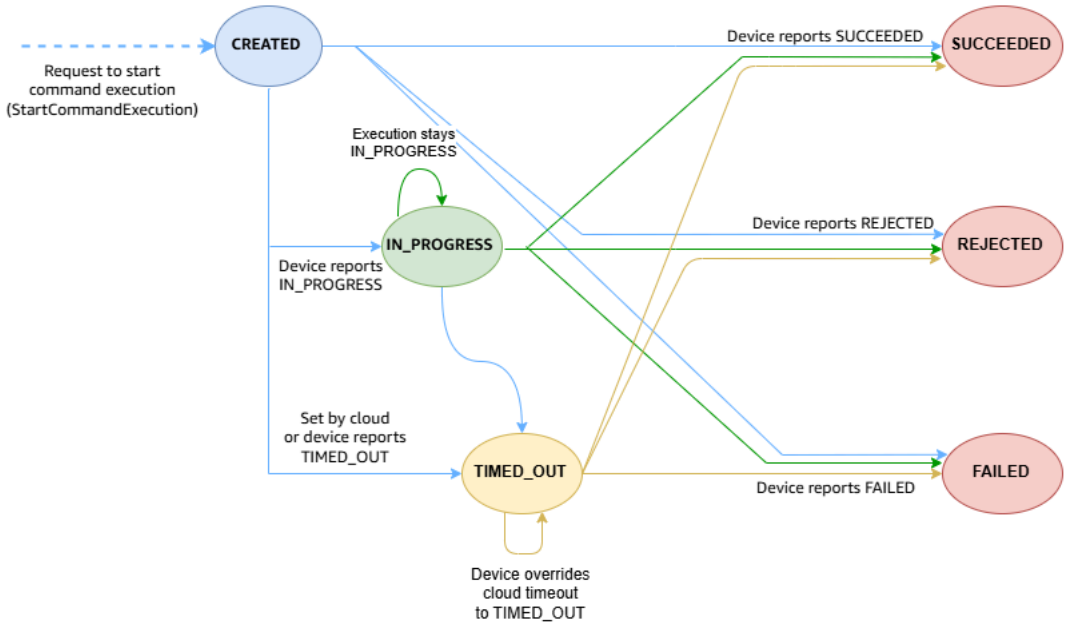
命令執行狀態和來源

命令執行狀態	由裝置/雲端啟動？	終端機執行？	允許的狀態轉換
CREATED	雲端	否	• IN_PROGRESS • SUCCEEDED (成功) • 失敗 • REJECTED • TIMED_OUT
IN_PROGRESS	裝置	否	• IN_PROGRESS

命令執行狀態	由裝置/雲端啟動？	終端機執行？	允許的狀態轉換
			• SUCCEEDED (成功) • 失敗 • REJECTED • TIMED_OUT
TIMED_OUT	裝置和雲端	否	• SUCCEEDED (成功) • 失敗 • REJECTED • TIMED_OUT
SUCCEEDED	裝置	是	不適用
FAILED	裝置	是	不適用
REJECTED	裝置	是	不適用

裝置可以隨時使用命令預留 MQTT 主題來發佈狀態和結果更新。為了提供額外的內容，裝置可以使用 `statusReason` 物件中的 `reasonCode`和 `reasonDescription` 欄位。

下圖顯示執行狀態轉換。



> **Note**
>
> 當 AWS IoT 偵測到逾時期間內沒有裝置回應時，它會將 TIMED_OUT設定為暫時狀態，以允許重試和狀態變更。如果您的裝置明確報告 TIMED_OUT，這會成為沒有進一步轉換的終端狀態。如需詳細資訊，請參閱非終端命令執行。

下列各節說明終端機和非終端機執行及其狀態。

主題

- 非終端命令執行
- 終端機命令執行

非終端命令執行

如果可以接受來自裝置的更新，則執行是非終端的。非終端執行會被視為作用中。下列狀態為非終端狀態：

- CREATED (已建立)

  當您從 AWS IoT 主控台或使用 StartCommandExecution API 開始執行時，成功的請求會將狀態變更為 CREATED。從此狀態，執行可以轉換為任何其他非終端或終端狀態。

- IN_PROGRESS

  收到承載後，裝置可以開始執行指示和執行指定的動作。執行時，裝置可以將回應發佈至命令回應主題，並將狀態更新為 IN_PROGRESS。從 IN_PROGRESS，執行可以轉換到 以外的任何終端機或非終端機狀態CREATED。

  > **Note**
  >
  > 您可以多次叫用 UpdateCommandExecution API 並顯示IN_PROGRESS狀態。使用 statusReason 物件指定其他執行詳細資訊。

- TIMED_OUT

  雲端和裝置都可以觸發此狀態。CREATED 或 IN_PROGRESS 狀態的執行可能會TIMED_OUT因為下列原因而變更為 ：

  - 傳送命令後，計時器會啟動。如果裝置未在指定的持續時間內回應，雲端會將狀態變更為 TIMED_OUT。在這種情況下，執行是非終端的。
  - 裝置可以將狀態覆寫為任何終端狀態，或報告逾時並將狀態設定為 TIMED_OUT。在此情況下，狀態會保持 TIMED_OUT，但StatusReason物件欄位會根據裝置資訊而變更。執行會變成終端機。

  如需詳細資訊，請參閱逾時值和TIMED_OUT執行狀態。

終端機命令執行

當執行不再接受來自裝置的更新時，就會變成終端機。下列狀態為終端機。執行可以從任何非終端狀態轉換為終端狀態：IN_PROGRESS、 CREATED或 TIMED_OUT。

- SUCCEEDED (成功)

  如果裝置成功完成執行，則可以將回應發佈至命令回應主題，並將狀態更新為 SUCCEEDED。

- 失敗

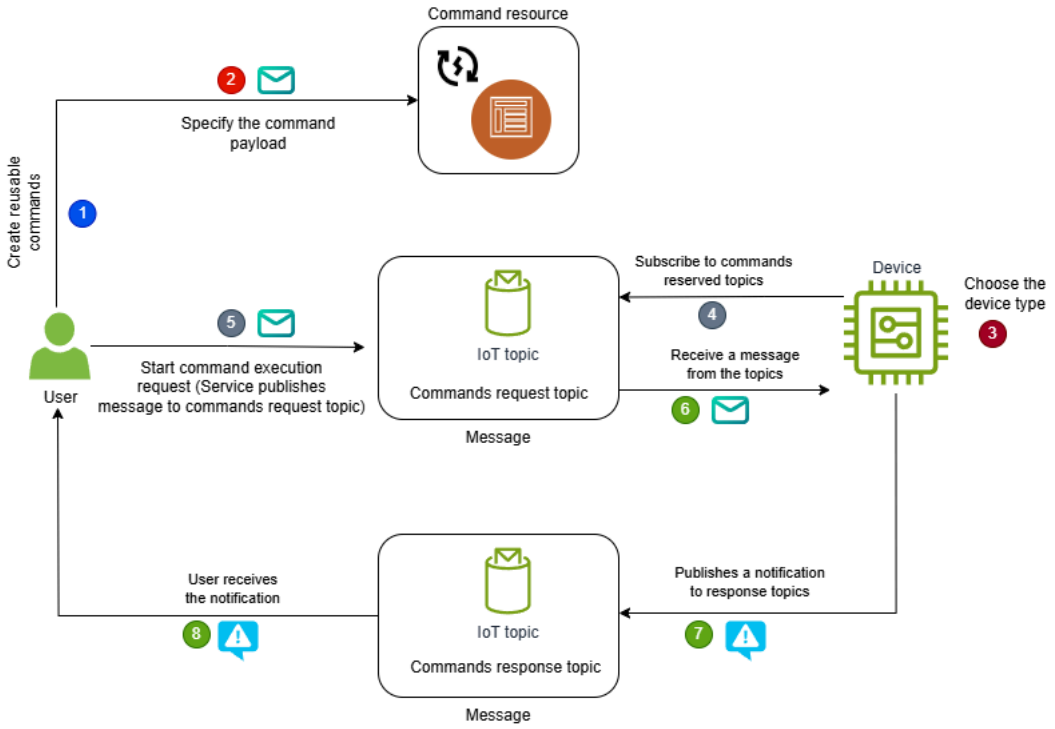
  當裝置無法完成執行時，可以發佈回應至命令回應主題，並將狀態更新至 FAILED。使用 statusReason 物件中的 reasonCode和 reasonDescription 欄位或 CloudWatch 日誌，對失敗進行故障診斷。

- REJECTED

當裝置收到無效或不相容的請求時，它可以調用狀態為 的 UpdateCommandExecution APIREJECTED。使用 statusReason 物件中的 reasonCode和 reasonDescription 欄位或 CloudWatch 日誌來疑難排解問題。

高階命令工作流程

此工作流程顯示裝置如何與 AWS IoT Device Management 命令互動。所有 HTTP API 請求都使用 Sigv4 登入資料進行簽署。



工作流程概觀

- 建立和管理命令
- 為您的命令選擇目標裝置並訂閱 MQTT 主題
- 啟動和監控目標裝置的命令執行
- （選用）啟用命令事件的通知

建立和管理命令

若要建立和管理裝置的命令，請執行下列步驟。

1. 建立命令資源

   從 Command Hub 或使用 API 建立命令。 `CreateCommand`

2. 指定承載

   以任何格式提供承載。指定內容類型，以確保正確的裝置解釋。

   對於具有承載範本的動態命令，最終承載會在執行時間使用您提供的參數產生。範本僅支援 JSON 格式，但產生的承載可以 JSON 或 CBOR 傳送。

3. （選用）管理建立的命令

   建立後更新顯示名稱和描述。不再需要時，將命令標記為已棄用，或將其永久刪除。若要修改承載資訊，請建立新的 命令。

為您的命令選擇目標裝置並訂閱 MQTT 主題

選擇您的目標裝置，並設定 MQTT 主題以接收命令和發佈回應。

1. 選擇命令的目標裝置

   選擇要接收和執行 命令的目標裝置。將物件名稱用於已註冊的裝置，或將用戶端 ID 用於未註冊的裝置。如需詳細資訊，請參閱目標裝置考量事項。

2. 設定 AWS IoT 裝置政策

   設定授予許可以接收執行和發佈更新的 IAM 政策。如需政策範例範例 IAM 政策，請參閱 。

3. 建立 MQTT 連線

   將裝置連接至訊息中介裝置，並訂閱請求和回應主題。裝置需要 `iot:Connect` 許可。使用 `DescribeEndpoint` API 或 CLI `describe-endpoint` 命令尋找您的資料平面端點：

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   執行此命令會傳回帳戶特定的資料平面端點，如下所示。

   ```
   account-specific-prefix.iot.region.amazonaws.com
   ```

4. 訂閱命令主題

   訂閱 命令請求主題。當您啟動執行時，訊息中介裝置會將承載發佈至此主題。您的裝置會接收並處理 命令。

   （選用）訂閱回應主題 (accepted 或 rejected) 以接收雲端服務是否接受或拒絕裝置回應的確認。

   在此範例中，取代：

   - *<device>* ，thing或client取決於您要鎖定的裝置是否已註冊為 IoT 物件，或指定為 MQTT 用戶端。
   - *<DeviceID>* 具有目標裝置的唯一識別符。此 ID 可以是唯一的 MQTT 用戶端 ID 或物件名稱。

   > **Note**
   >
   > 如果承載類型不是 JSON 或 CBOR，則命令請求主題中可能不存在 *<PayloadFormat>* 欄位。若要取得承載格式，建議您使用 MQTT5 從 MQTT 訊息標頭取得格式資訊。如需詳細資訊，請參閱命令主題。

   ```
   $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
   $aws/commands/<devices>/<DeviceID>/executions/+/response/accepted/<PayloadFormat>
   $aws/commands/<devices>/<DeviceID>/executions/+/response/rejected/<PayloadFormat>
   ```

啟動和監控目標裝置的命令執行

建立命令並指定命令的目標之後，您可以執行下列步驟，在目標裝置上開始執行。

1. 在目標裝置上啟動命令執行

   從 Command Hub 或使用 StartCommandExecution API 搭配您帳戶特定的端點來啟動執行。iot:Data-ATS 用於雙堆疊 (IPv4/IPv6) 或僅用於 iot:Jobs IPv4。

   API 會將承載發佈至 命令請求主題。

> **Note**
>
> 如果裝置離線並使用 MQTT 持久性工作階段， 命令會等待訊息中介裝置。當裝置在逾時前重新連線時，可以處理 命令並發佈結果。如果逾時過期，則執行逾時，承載將被捨棄。

2. 更新命令執行的結果

   裝置會收到承載、處理 命令、執行指定的動作，以及使用 UpdateCommandExecution MQTT API 將結果發佈至 命令回應主題。如果訂閱接受和拒絕的主題，裝置會收到雲端服務是否接受或拒絕回應的確認。

   根據您在請求主題中指定的內容，*<devices>* 可以是實物或用戶端，而 *<DeviceID>* 可以是您的 AWS IoT 實物名稱或 MQTT 用戶端 ID。

   > **Note**
   >
   > *<PayloadFormat>* 只能是命令回應主題中的 JSON 或 CBOR。

   ```
   $aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/
   response/<PayloadFormat>
   ```

3. （選用）擷取命令執行結果

   從 AWS IoT 主控台或使用 擷取執行結果GetCommandExecution。裝置必須將結果發佈至 Commands 回應主題，以取得最新資訊。檢視其他詳細資訊，包括上次更新時間、結果和完成時間。

（選用）啟用命令事件的通知

訂閱 命令事件，以在執行狀態變更時收到通知。如需命令執行事件的詳細資訊，包括事件訊息格式和承載屬性，請參閱 命令執行事件。

1. 建立主題規則

   訂閱 Commands 事件主題以取得狀態變更通知。使用 AWS IoT 主控台或 建立主題規則，將裝置資料路由到其他服務 AWS Lambda， AWS IoT 例如 Amazon SQS 和 AWS Step Functions建立 AWS IoT 規則。

在此範例中，將 *<CommandID>*取代為您要接收通知之命令的識別符，並將 *<CommandExecutionStatus>*取代為命令執行的狀態。

```
$aws/events/commandExecution/<CommandID>/<CommandExecutionStatus>
```

> **Note**
>
> 若要接收所有命令和命令執行狀態的通知，您可以使用萬用字元並訂閱下列主題。

```
$aws/events/commandExecution/+/#
```

2. 接收和處理命令事件

   管理命令推送通知，並使用訂閱的事件建置應用程式。

下列程式碼顯示您將收到的命令事件通知的範例承載。

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status":"FAILED",
    "statusReason": {
         "reasonCode": "DEVICE_TOO_BUSY",
         "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn":"arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn":"arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp":1717708862107
}
```

建立和管理命令

使用 AWS IoT Device Management 命令來設定可重複使用的遠端動作，或傳送立即指示至裝置。從 AWS IoT 主控台或使用 建立和管理命令 AWS CLI。

建立和管理命令操作

- 建立命令資源
- 擷取命令的相關資訊
- 在 中列出命令 AWS 帳戶
- 更新命令資源
- 棄用或還原命令資源
- 刪除命令資源

建立命令資源

在建立命令時提供以下資訊：

- 一般資訊

  提供唯一的命令 ID，以在目標裝置上執行命令時識別命令。選擇性地指定要管理的顯示名稱、描述和標籤。

- 承載

  對於靜態命令，請提供承載定義裝置動作。指定用於正確裝置解譯的承載格式類型。

  如需動態命令，請參閱承載範本屬性。

- 承載範本

  對於動態命令，請提供具有預留位置和參數的 payloadTemplate。選擇性地提供 `defaultValue`和條件。 AWS IoT Device Management 命令會在執行時間取代預留位置。缺少參數會使用其 defaultValue。所有值都必須滿足定義的條件。

  支援以下區分大小寫的預留位置類型：

  - `${aws:iot:commandexecution::parameter:`*`parameter1`*`}` – 名稱為 之參數值的預留位置`parameter1`。
  - `${aws:iot:commandexecution::executionTimeoutSec}` – 執行時間提供的命令執行逾時參數的預留位置。

承載和命令主題

命令預留主題使用基於承載格式類型的格式。

- 對於 application/json或 application/cbor內容類型，請使用此請求主題：

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

- 對於其他內容類型或未指定的格式，請使用此請求主題。承載格式會出現在 MQTT 訊息標頭中。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

命令回應主題使用 json或 cbor 格式，無論承載類型為何。*<PayloadFormat>* 必須是 json或 cbor：

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

建立命令資源 （ 主控台 ）

下列各節說明承載格式考量，以及從 主控台建立命令。

主題

- 靜態命令承載格式
- 動態命令承載格式
- 如何建立命令 （ 主控台 ）

靜態命令承載格式

承載支援任何最多 32 KB 的格式。指定承載格式類型，以進行安全且正確的裝置解譯。

使用格式 （ 例如 application/json或 application/cbor) 指定承載type/subtype格式類型。預設：application/octet-stream。如需支援的格式，請參閱常見 MIME 類型。

動態命令承載格式

payloadTemplate 必須是具有至少一個預留位置的有效 JSON，最多 32 KB。

對於AwsJsonSubstitution預處理器， AWS IoT Device Management Commands 會根據預處理器組態，以 JSON 或 CBOR 格式傳送通知。

如何建立命令 （ 主控台 ）

若要從主控台建立命令，請前往 Command Hub 並遵循下列步驟：

1. 選擇建立命令。
2. 指定唯一的命令 ID。
3. （選用）指定顯示名稱、描述和標籤。
4. 上傳包含裝置動作的承載檔案。指定用於正確裝置解譯的承載格式類型。
5. （選用）對於具有預留位置的 JSON 承載範本，參數會預先填入內嵌資料表中以供編輯。
6. （選用）設定參數值類型（必要）、預設值和條件。
7. 選擇建立命令。

建立命令資源 (CLI)

使用 `CreateCommand` API 或 `create-command` CLI 命令來建立命令。

主題

- 命令承載
- 範例 IAM 政策
- 靜態命令建立範例
- 動態命令建立範例

命令承載

提供靜態承載或承載範本。靜態承載是 base64 編碼。對於承載範本，最終承載會在執行時間使用參數值產生。裝置會處理承載並執行指定的動作。指定用於正確裝置接收的承載內容類型。

> **Note**
>
> 建立命令後無法修改承載。建立新的命令來修改承載。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示 IAM 政策，允許使用者執行 `CreateCommand`動作的許可。

在此範例中，取代：

- *`region`* 搭配您的 AWS 區域，例如 *`us-east-1`*。
- *`account-id`* 您的 AWS 帳戶 號碼，例如 *`123456789012`*。

- *command-id* 具有 AWS IoT 命令 ID 的唯一識別符，例如 *LockDoor*。如果您想要傳送多個命令，您可以在 IAM 政策的資源區段下指定這些命令。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:CreateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

靜態命令建立範例

下列範例示範如何建立靜態命令。根據您的應用程式，取代：

- *<command-id>* 具有命令的唯一識別符。例如，若要鎖定房屋的門，您可以指定 *LockDoor*。我們建議您使用 UUID。您也可以使用英數字元、"-" 和 "_"。
- （選用）*<description>* *<display-name>*和 ，這些是選用欄位，可用來為命令提供易記的名稱和有意義的描述，例如 *Lock the doors of my home*。
- namespace，您可以使用它來指定命令的命名空間。它必須是 AWS-IoT。如需針對 使用此功能的詳細資訊 AWS IoT FleetWise，請參閱遠端命令
- payload 包含您在執行 命令時要使用之承載及其內容類型的相關資訊。

```
aws iot create-command \
    --command-id <command-id> \
    --display-name <display-name> \
    --description <description> \
    --namespace AWS-IoT \
    --payload
 '{"content":"eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=","contentType":"application/json"}'
```

執行此命令會產生回應，其中包含命令的 ID 和 ARN (Amazon 資源名稱）。例如，如果您在建立期間指定 *LockDoor*命令，則以下顯示執行命令的範例輸出。

```
{
```

```
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor"
}
```

動態命令建立範例

下列範例示範如何建立動態命令。根據您的應用程式，取代：

- *<command-id>* 具有命令的唯一識別符。例如，若要設定光源狀態，您可以指定 *Light_Power_Status*。我們建議您使用 UUID。您也可以使用英數字元、"-" 和 "_"。
- （選用）*<description>* *<display-name>*和 ，這些是選用欄位，可用來為命令提供易記的名稱和有意義的描述，例如 *Turn a light ON or OFF*。
- namespace，您可以使用它來指定命令的命名空間。它必須是 AWS-IoT。如需針對 使用此功能的詳細資訊 AWS IoT FleetWise，請參閱遠端命令
- payloadTemplate 包含具有 placehodlers 的 JSON 格式 Playoad 範本。
- preprocessor 包含預處理器的組態，決定如何處理 payloadTemplate。
- mandatoryParameter 包含對應於 payloadTemplate 中預留位置的參數、其類型、預設值和條件。

```
aws iot create-command \
    --command-id Light_Power_Status \
    --description "Turn a light ON or OFF" \
    --namespace AWS-IoT \
    --payload-
template '{"powerStatus":"${aws:iot:commandexecution::parameter:powerStatus}"}' \
    --preprocessor awsJsonSubstitution={outputFormat=JSON} \
    --mandatory-parameters "name=powerStatus, defaultValue={S=OFF},
 valueConditions=[{comparisonOperator=IN_SET, operand={strings=['ON','OFF']}}]"
```

執行此命令會產生回應，其中包含命令的 ID 和 ARN (Amazon 資源名稱）。例如，如果您在建立期間指定 *Light_Power_Status*命令，則以下顯示執行命令的範例輸出。

```
{
    "commandId": "Light_Power_Status",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status"
}
```

擷取命令的相關資訊

建立命令之後，您可以從主控台並使用 擷取其 AWS IoT 相關資訊 AWS CLI。您可以取得下列資訊。

- 命令 ID、Amazon 資源名稱 (ARN)、您為命令指定的任何顯示名稱和描述。
- 命令狀態，指出命令是否可在目標裝置上執行，或是否已被取代或刪除。
- 您提供的承載或 payloadTemplate。
- 您提供的前置處理器。
- 您提供的 mandatoryParameters。
- 命令建立和上次更新的時間。

擷取命令資源 （ 主控台 ）

若要從主控台擷取命令，請前往 AWS IoT 主控台的 Command Hub，然後選擇您建立的命令以檢視其詳細資訊。

除了命令詳細資訊之外，您還可以查看命令歷史記錄，該歷史記錄提供有關在目標裝置上執行命令的資訊。在裝置上執行此命令後，您可以在此標籤上找到執行的相關資訊。

擷取命令資源 (CLI)

使用 `GetCommand` HTTP 控制平面 API 操作或 `get-command` AWS CLI 命令來擷取命令資源的相關資訊。您必須已經使用 `CreateCommand` API 請求或 CLI `create-command` 建立 命令。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示允許使用者執行 `GetCommand`動作的 IAM 政策。

在此範例中，取代：

- *`region`* 搭配您的 AWS 區域，例如 `us-east-1`。
- *`account-id`* 使用您的 AWS 帳戶 號碼，例如 *`123456789023`*。
- *`command-id`* 您的 AWS IoT 唯一命令識別符，例如 *`LockDoor`*或 *`Light_Power_Status`*。如果您想要擷取多個命令，您可以在 IAM 政策的資源區段下指定這些命令。

```
{
    "Version":"2012-10-17",
```

```
    "Statement": {
        "Action": "iot:GetCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

擷取命令範例 (AWS CLI)

下列範例說明如何使用 擷取命令的相關資訊get-command AWS CLI。根據您的應用程式，將*<command-id>*取代為您要擷取資訊的 命令的識別符。您可以從 CLI create-command 的回應取得此資訊。

```
aws iot get-command --command-id <command-id>
```

執行此命令會產生回應，其中包含命令、承載及其建立和上次更新時間的相關資訊。它也提供指出命令是否已棄用或正在刪除的資訊。

例如，下列程式碼顯示範例回應。

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:<region>:<account>:command/LockDoor",
    "namespace": "AWS-IoT",
    "payload":{
        "content": "eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-03-23T00:50:10.095000-07:00",
    "deprecated": false,
    "pendingDeletion": false
}
```

在 中列出命令 AWS 帳戶

建立命令之後，您可以檢視在帳戶中建立的命令。在清單中，您可以找到下列相關資訊：

- 命令 ID，以及您為命令指定的任何顯示名稱。
- 命令的 Amazon 資源名稱 (ARN)。

- 命令狀態，指出命令是否可在目標裝置上執行，或是否已棄用。

> **Note**
>
> 此清單不會顯示正在從您的帳戶中刪除的 。如果命令正在等待刪除，您仍然可以使用其命令 ID 檢視這些命令的詳細資訊。

- 命令建立和上次更新的時間。

列出您帳戶中的命令 （主控台）

在 AWS IoT 主控台中，您可以前往 Command Hub，找到您建立的命令清單及其詳細資訊。

列出您帳戶中的命令 (CLI)

若要列出您建立的命令，請使用 `ListCommands` API 操作或 `list-commands` CLI。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示允許使用者執行 `ListCommands`動作的 IAM 政策。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 `us-east-1`。
- *account-id* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:ListCommands",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/*"
    }
}
```

在您的帳戶範例中列出命令

下列命令說明如何列出您帳戶中的命令。

```
aws iot list-commands --namespace "AWS-IoT"
```

執行此命令會產生回應，其中包含您建立的命令清單、命令建立時間和上次更新的時間。它也提供命令狀態資訊，指出命令是否已棄用或可在目標裝置上執行。如需不同狀態和狀態原因的詳細資訊，請參閱命令執行狀態。

更新命令資源

建立命令之後，您可以更新命令的顯示名稱和描述。

> **Note**
>
> 命令的承載無法更新。若要更新此資訊或使用修改後的承載，您需要建立新的命令。

更新命令資源（主控台）

若要從主控台更新命令，請前往 AWS IoT 主控台的 Command Hub，並執行下列步驟。

1. 若要更新現有的命令資源，請選擇您要更新的命令，然後在動作下選擇編輯。
2. 指定您要使用的顯示名稱和描述，以及任何名稱/值對做為命令的標籤。
3. 選擇編輯以使用新設定儲存命令。

更新命令資源 (CLI)

使用UpdateCommand控制平面 API 操作或 update-command AWS CLI 來更新命令資源。使用此 API，您可以：

- 編輯您建立之命令的顯示名稱和描述。
- 棄用命令資源，或還原已棄用的命令。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示 IAM 政策，允許使用者執行 UpdateCommand動作的許可。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 us-east-1。

- *account-id* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。
- *command-id* 您的 AWS IoT 唯一命令識別符，例如 *LockDoor*。如果您想要擷取多個命令，您可以在 IAM 政策中的資源區段下指定這些命令。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:UpdateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

更新命令範例的相關資訊 (AWS CLI)

下列範例示範如何使用 命令更新命令的相關資訊update-command AWS CLI 。如需有關如何使用此 API 取代或還原命令資源的資訊，請參閱 更新命令資源 (CLI)。

此範例顯示如何更新命令的顯示名稱和描述。根據您的應用程式，將 *<command-id>*取代為您要擷取資訊的命令的識別符。

```
aws iot update-command \
    --command-id <command-id>
    --displayname <display-name> \
    --description <description>
```

執行此命令會產生回應，其中包含有關命令和上次更新時間的更新資訊。下列程式碼顯示更新關閉 AC 之命令的顯示名稱和描述的範例請求和回應。

```
aws iot update-command \
    --command-id <LockDoor> \
    --displayname <Secondary lock door> \
    --description <Locks doors to my home>
```

執行此命令會產生下列回應。

```
{
    "commandId": "LockDoor",
```

```
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "displayName": "Secondary lock door",
    "description": "Locks doors to my home",
    "lastUpdatedAt": "2024-05-09T23:15:53.899000-07:00"
}
```

棄用或還原命令資源

建立命令之後，如果不想再繼續使用命令，您可以將其標記為已棄用。當您棄用命令時，所有待定命令執行都會繼續在目標裝置上執行，直到達到終端機狀態為止。一旦命令已棄用，如果您想要使用 等命令，將新的命令執行傳送至目標裝置，則必須將其還原。

> **Note**
>
> 您無法編輯已棄用的命令，或對其執行任何新的執行。若要在裝置上執行新的命令，您必須將其還原，命令狀態才會變更為可用。

如需有關棄用和還原命令以及其考量事項的其他資訊，請參閱 取代指令資源。

刪除命令資源

如果您不想再使用命令，可以從您的帳戶永久移除該命令。如果刪除動作成功：

- 如果命令已棄用超過最大逾時 12 小時的持續時間，則會立即刪除命令。
- 如果命令未棄用，或已棄用超過最大逾時的持續時間，則命令將處於 `pending deletion` 狀態。最長逾時 12 小時後，系統會自動將其從您的帳戶中移除。

> **Note**
>
> 即使有任何待定的命令執行，也可能會刪除命令。命令將處於待定刪除狀態，並會自動從您的帳戶中移除。

刪除命令資源（主控台）

若要從主控台刪除命令，請前往 AWS IoT 主控台的 Command Hub，然後執行下列步驟。

1. 選擇您要刪除的命令，然後在動作下選擇刪除。

2. 確認您想要刪除命令，然後選擇刪除。

命令將標記為刪除，並在 12 小時後從您的帳戶中永久移除。

刪除命令資源 (CLI)

使用 DeleteCommand HTTP 控制平面 API 操作或 delete-command AWS CLI 命令來刪除命令資源。如果刪除動作成功，您會看到 HTTP statusCode為 204 或 202，且命令會在逾時持續時間上限為 12 小時後自動從您的帳戶刪除。在 204 狀態的情況下，表示命令已刪除。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示 IAM 政策，允許使用者執行 DeleteCommand動作的許可。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 us-east-1。
- *account-id* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。
- *command-id* 您的 AWS IoT 唯一命令識別符，例如 *LockDoor*。如果您想要擷取多個命令，您可以在 IAM 政策的資源區段下指定這些命令。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:DeleteCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

刪除命令範例 (AWS CLI)

下列範例示範如何使用 命令刪除delete-command AWS CLI 命令。根據您的應用程式，*<command-id>*將 取代為您要刪除之命令的識別符。

```
aws iot delete-command --command-id <command-id>
```

如果 API 請求成功，則命令會產生狀態碼 202 或 204。您可以使用 GetCommand API 來驗證 命令是否不再存在於您的帳戶中。

開始和監控命令執行

建立命令後，在目標裝置上啟動執行。裝置會更新結果並將狀態發佈至 MQTT 預留主題。從您的帳戶擷取和監控執行狀態。

使用 AWS IoT 主控台或 啟動和監控命令 AWS CLI。

啟動和監控命令操作

- 啟動命令執行
- 更新命令執行的結果
- 擷取命令執行
- 使用 MQTT 測試用戶端檢視命令更新
- 在 中列出命令執行 AWS 帳戶
- 刪除命令執行

啟動命令執行

> **Important**
>
> 您全權負責以安全且符合相關法律的方式部署命令。

開始執行之前，請確定：

- 您已在具有承載資訊的 AWS IoT 命名空間中建立 命令。開始執行時，裝置會處理承載指示並執行指定的動作。如需建立命令建立命令資源的詳細資訊，請參閱 。
- 您的裝置已訂閱 命令的 MQTT 預留主題。開始執行時，承載資訊會發佈至此預留 MQTT 請求主題：

  <devices> 可以是 Things 或 MQTT 用戶端。*<DeviceID>* 是物件名稱或用戶端 ID。支援的 *<PayloadFormat>* 值：JSON 和 CBOR。如需詳細資訊，請參閱命令主題。

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

對於非 JSON/CBOR *<PayloadFormat>*，請使用下列命令主題格式：

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

目標裝置考量事項

指定要接收和執行 命令的目標裝置。將物件名稱用於已註冊的裝置，或將用戶端 ID 用於未註冊的裝置。收到承載後，裝置會執行 命令並執行指定的動作。

AWS IoT 物件

目標裝置可以是在 AWS IoT 登錄檔中註冊的物件。實物可簡化裝置搜尋和管理。

從 Connect 裝置頁面或使用 將裝置註冊為實物CreateThing。從物件中樞或使用 尋找現有的物件DescribeThing。如需註冊詳細資訊，請參閱使用登錄檔管理物件。

用戶端 ID

對於未註冊的裝置，請使用 用戶端 ID。

用戶端 ID 是您指派給裝置的唯一識別符。在 MQTT 通訊協定中定義，其中包含英數字元、底線或破折號。連線至 的每個裝置 AWS IoT 都需要唯一的用戶端 ID。

> **Note**
>
> - 對於已註冊的物件，用戶端 ID 可以符合物件名稱。
> - 以特定用戶端 ID 為目標時，裝置必須使用 AWS IoT 該用戶端 ID 連線至 ，才能接收承載。

用戶端 ID 是連線至 時使用的 MQTT 用戶端 ID 裝置 AWS IoT Core。 AWS IoT 使用此 ID 來識別裝置和管理連線和訂閱。

命令執行逾時考量

逾時會指定裝置提供執行結果的持續時間 （ 以秒為單位 ）。

建立執行後，計時器會啟動。如果裝置離線或無法在逾時內報告結果，則執行會逾時，狀態為 TIMED_OUT。

預設：10 秒。上限：12 小時。

逾時值和**TIMED_OUT**執行狀態

雲端和裝置都可以報告逾時。

傳送 命令後，計時器會啟動。如果沒有裝置回應在逾時內到達，則雲端會將執行狀態設定為 ，TIMED_OUT原因碼為 $NO_RESPONSE_FROM_DEVICE。

發生下列情況時，會發生這種情況：

- 裝置在執行期間離線。
- 裝置無法在逾時內完成執行。
- 裝置無法在逾時內報告狀態。

在此執行個體中，當從雲端報告 TIMED_OUT的執行狀態時，命令執行是非終端的。您的裝置可以發佈回應，將狀態覆寫為任何終端狀態：FAILED、 SUCCEEDED或 REJECTED。命令執行接著會變成終端機，且不接受任何進一步的更新。

您的裝置也可以透過報告執行命令時發生逾時，來更新雲端啟動TIMED_OUT的狀態。在此情況下，命令執行狀態會保持在 TIMED_OUT，但會根據裝置報告的資訊更新statusReason物件。然後，命令執行會變成終端機，而且不接受進一步的更新。

使用 MQTT 持久性工作階段

您可以設定 MQTT 持久性工作階段，以搭配 AWS IoT Device Management 命令功能使用。此功能在例如當您的裝置離線，且您希望確保裝置在逾時持續時間之前恢復上線時仍能收到命令，並執行指定的指示等情況下特別有用。

MQTT 持續性工作階段到期時間預設為 60 分鐘。如果您的命令執行逾時設定為超過此持續時間的值，執行時間超過 60 分鐘的命令執行可能會被訊息代理程式拒絕並失敗。若要執行持續期間超過 60 分鐘的命令，您可以請求增加持續性工作階段到期時間。

> **Note**
>
> 為了確保您正確使用 MQTT 持久性工作階段功能，請將 Clean Start 旗標設為零。如需詳細資訊，請參閱 MQTT 持續性工作階段。

啟動命令執行 （主控台）

若要從主控台開始執行命令，請前往 AWS IoT 主控台的 Command Hub 頁面，然後執行下列步驟。

1. 若要執行您已建立的命令，請選擇執行命令。
2. 檢閱您所建立命令的相關資訊，包括 MQTT 預留主題和參數，如適用。

   針對動態命令，輸入參數值或保留預設值。對於沒有預設值的參數，您必須提供要在此執行中傳送的值。
3. 指定要接收和執行 命令的目標裝置。如果裝置已向 註冊，則可以指定為 AWS IoT 物件 AWS IoT；如果裝置尚未註冊，則可以使用用戶端 ID。如需詳細資訊，請參閱目標裝置考量事項
4. （選用）設定命令的逾時值，決定命令在逾時之前執行的持續時間。如果您的命令需要執行超過 60 分鐘，您可能需要增加 MQTT 持久性工作階段到期時間。如需詳細資訊，請參閱命令執行逾時考量。
5. 選擇執行命令。

啟動命令執行 (AWS CLI)

使用 StartCommandExecution HTTP 資料平面 API 操作來啟動命令執行。API 請求和回應與命令執行 ID 相關聯。裝置完成執行命令後，可以透過將訊息發佈至命令回應主題，向雲端報告狀態和執行結果。對於自訂回應碼，您擁有的應用程式碼可以處理回應訊息並將結果發佈到其中 AWS IoT。

如果您的裝置已訂閱命令請求主題，StartCommandExecutionAPI 會將承載訊息發佈至主題。承載可以使用您選擇的任何格式。如需詳細資訊，請參閱命令承載。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

如果承載格式不是 JSON 或 CBOR，則以下顯示命令請求主題的格式。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示 IAM 政策，允許使用者執行 StartCommandExecution動作的許可。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 us-east-1。
- *account-id* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。

- *command-id* 具有 AWS IoT 命令的唯一識別符，例如 *LockDoor*。如果您想要傳送多個命令，您可以在 IAM 政策中指定這些命令。
- *devices* 使用 thing或 ，client取決於您的裝置是否已註冊為 AWS IoT 實物，或指定為 MQTT 用戶端。
- *device-id* 您的 AWS IoT thing-name或 client-id。

```
{
  "Effect": "Allow",
  "Action": [
      "iot:StartCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

若要查看 支援的條件金鑰清單StartCommandExecution，請參閱《IAM 使用者指南》中的 的條件金鑰 AWS IoT。

取得帳戶特定的資料平面端點

執行 API 命令之前，您必須取得端點的帳戶特定端點 URL。如果您使用的是雙堆疊端點 (IPv4 和 IPv6)，請使用 iot:Data-ATS。iot:Jobs 端點僅適用於 IPv4。例如，如果您執行此命令：

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

它會傳回帳戶特定的端點 URL，如以下範例回應所示。

```
{
    "endpointAddress":
    "<account-specific-prefix>-ats.iot.<region>.api.com"
}
```

啟動命令執行範例 (AWS CLI)

下列範例顯示如何使用 命令開始執行start-command-execution AWS CLI 命令。

在此範例中，取代：

- *<command-arn>* 搭配您要執行之命令的 ARN。您可以從 CLI `create-command` 命令的回應取得此資訊。例如，如果您正在執行變更方向盤模式的命令，請使用 `arn:aws:iot:`*region*`:`*account-id*`:command/`*SetComfortSteeringMode*。
- *<target-arn>* 目標裝置的物件 ARN，可以是您要執行命令的 IoT 物件或 MQTT 用戶端。例如，如果您要執行目標裝置 的 命令`myRegisteredThing`，請使用 `arn:aws:iot:`*region*`:`*account-id*`:thing/`*myRegisteredThing*。
- *<endpoint-url>* 使用您在 中取得的帳戶特定端點取得帳戶特定的資料平面端點，字首為 `https://`。例如 `https://`*123456789012abcd*`.jobs.iot.`*us-east-1*`.amazonaws.com`。
- （選用）您也可以在執行 `StartCommandExecution` API 操作時指定其他參數 `executionTimeoutSeconds`。此選用欄位指定裝置必須完成執行命令的時間，以秒為單位。根據預設，值為 10 秒。當命令執行狀態為 時`CREATED`，計時器會啟動。如果在計時器過期之前未收到命令執行結果，則狀態會自動變更為 `TIMED_OUT`。
- ```
  aws iot-jobs-data start-command-execution \
      --command-arn <command-arn>  \
      --target-arn <target-arn> \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900
  ```
- （選用）對於動態命令，指定要用於替換的參數及其值。您必須為在命令建立時未設定 defaultValue 的參數提供值。如果參數具有 defaultValue，則此處提供的參數值優先。對於設定 valueConditions 的參數，此處提供的參數值必須符合條件。

  根據`Light_Power_Status`動態命令範例：
- ```
  aws iot-jobs-data start-command-execution \
      --command-arn arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status  \
      --target-arn arn:aws:iot:us-east-1:123456789012:thing/exampleThing \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900 \
      --parameters "powerStatus={S=ON}"
  ```

執行此命令會傳回命令執行 ID。您可以使用此 ID 來查詢命令執行狀態、詳細資訊和命令執行歷史記錄。

> **Note**
>
> 如果命令已棄用，則 StartCommandExecution API 請求會失敗並出現驗證例外狀況。若要修正此錯誤，請先使用 UpdateCommand API 還原命令，然後執行StartCommandExecution請求。

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
}
```

更新命令執行的結果

使用 UpdateCommandExecution MQTT 資料平面 API 操作來更新命令執行的狀態或結果。

> **Note**
>
> 使用此 API 之前：
>
> - 您的裝置必須已建立 MQTT 連線並訂閱命令請求和回應主題。如需詳細資訊，請參閱高階命令工作流程。
> - 您必須已使用 StartCommandExecution API 操作執行此命令。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您的裝置執行這些動作。以下顯示範例政策，授權您的裝置執行 動作。如需允許使用者執行 UpdateCommandExecution MQTT 動作之許可的其他範例 IAM 政策，請參閱 連線和發佈政策範例。

在此範例中，取代：

- *Region* ， AWS 區域例如 us-east-1。
- *AccountID* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。
- *ThingName* 您以命令執行為目標的 AWS IoT 物件名稱，例如 *myRegisteredThing*。
- *commands-request-topic* 和 *commands-response-topic* ，以及 AWS IoT 命令請求和回應主題的名稱。如需詳細資訊，請參閱高階命令工作流程。

MQTT 用戶端 ID 的 IAM 政策範例

下列程式碼顯示使用 MQTT 用戶端 ID 時的範例裝置政策。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted",
```

```
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

IoT 物件的 IAM 政策範例

下列程式碼顯示使用 AWS IoT 物件時的範例裝置政策。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected",
```

```
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

如何使用 **UpdateCommandExecution** API

在請求主題收到命令執行後，裝置會處理命令。然後，它會使用 UpdateCommandExecution API 將命令執行的狀態和結果更新為下列回應主題。

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

在此範例中，*<DeviceID>*是目標裝置的唯一識別符，而 *<execution-id>*是目標裝置上命令執行的識別符。*<PayloadFormat>* 可以是 JSON 或 CBOR。

> **Note**
>
> 如果您尚未向 註冊裝置 AWS IoT，您可以使用用戶端 ID 做為識別符，而不是物件名稱。

```
$aws/commands/clients/<ClientID>/executions/<ExecutionId>/response/<PayloadFormat>
```

裝置回報執行狀態的更新

您的裝置可以使用 API 向命令執行報告下列任何狀態更新。如需這些狀態的詳細資訊，請參閱 命令執行狀態。

- IN_PROGRESS：當裝置開始執行命令時，它可以將狀態更新為 IN_PROGRESS。
- SUCCEEDED：當裝置成功處理命令並完成執行時，裝置可以將訊息發佈至回應主題，做為 SUCCEEDED。
- FAILED：如果裝置無法執行命令，它可以將訊息發佈至回應主題，做為 FAILED。
- REJECTED：如果裝置無法接受 命令，它可以將訊息發佈到回應主題，做為 REJECTED。
- TIMED_OUT：TIMED_OUT由於下列任何原因，命令執行狀態可能會變更為 。
  - 未收到命令執行的結果。這可能是因為未在指定的持續時間內完成執行，或裝置無法將狀態資訊發佈至回應主題。
  - 裝置報告嘗試執行 命令時發生逾時。

如需 TIMED_OUT 狀態的詳細資訊，請參閱 逾時值和TIMED_OUT執行狀態。

使用 **UpdateCommandExecution** API 時的考量事項

以下是使用 UpdateCommandExecution API 時的一些重要考量。

- 您的裝置可以使用選用statusReason物件來提供有關執行的其他資訊。如果您的裝置提供此物件，則需要 物件reasonCode的欄位，但 reasonDescription 欄位是選用的。
- 當您的裝置使用 statusReason 物件時， reasonCode 必須使用 模式[A-Z0-9_-]+，長度不得超過 64 個字元。如果您提供 reasonDescription，請確保長度不超過 1，024 個字元。它可以使用任何字元，控制字元除外，例如換行。

- 您的裝置可以使用選用result物件來提供命令執行結果的相關資訊，例如遠端函數呼叫的傳回值。如果您提供 result，它必須至少需要一個項目。
- 在 result 欄位中，您將項目指定為鍵/值對。對於每個項目，您必須將資料類型資訊指定為字串、布林值或二進位。字串資料類型必須使用金鑰 s，布林值資料類型必須使用金鑰 b，而二進位資料類型必須使用金鑰 bin。請確定這些金鑰為小寫。
- 如果您在執行 UpdateCommandExecution API 時發生錯誤，您可以在 Amazon CloudWatch 的AWSIoTLogsV2日誌群組中檢視錯誤。如需啟用記錄和檢視日誌的資訊，請參閱 設定 AWS IoT 記錄。

UpdateCommandExecution API 範例

下列程式碼顯示您的裝置如何使用 UpdateCommandExecution API 來報告執行狀態的範例、提供狀態額外資訊statusReason的欄位，以及提供執行結果相關資訊的結果欄位，例如在此情況下的汽車電池百分比。

```
{
  "status": "IN_PROGRESS",
  "statusReason": {
    "reasonCode": "200",
    "reasonDescription": "Execution_in_progress"
  },
  "result": {
        "car_battery": {
            "s": "car battery at 50 percent"
        }
    }
}
```

> **Note**
>
> 當 UpdateCommandExecution MQTT 請求失敗時，服務會將錯誤回應發佈至/rejected主題。如需錯誤代碼和故障診斷指引的完整清單，請參閱 AWS IoT 命令故障診斷。

擷取命令執行

執行命令之後，您可以從 AWS IoT 主控台和使用 擷取命令執行的相關資訊 AWS CLI。您可以取得下列資訊。

> **Note**
>
> 若要擷取最新的命令執行狀態，您的裝置必須使用 UpdateCommandExecution MQTT API 將狀態資訊發佈至回應主題，如下所述。在裝置發佈至此主題之前，GetCommandExecutionAPI 會將狀態報告為 CREATED或 TIMED_OUT。

您建立的每個命令執行作業都會有：

- 執行 ID，這是命令執行作業的唯一識別碼。
- 命令執行的狀態。當您在目標裝置上執行命令時，命令執行作業會進入 CREATED 狀態。然後作業會轉換成其他命令執行狀態，如下所述。
- 命令執行的結果。
- 唯一的命令 ID 和已建立執行作業的目標裝置。
- 開始日期，顯示命令執行作業建立的時間。

擷取命令執行（主控台）

您可以使用下列其中一種方法，從主控台擷取命令執行。

- 從命令中樞頁面

  前往 AWS IoT 主控台的 Command Hub 頁面，並執行這些步驟。

  1. 選擇您在目標裝置上建立執行的命令。
  2. 在命令詳細資訊頁面的命令歷史記錄索引標籤中，您會看到您建立的執行。選擇您要擷取資訊的執行。
  3. 如果您的裝置使用 UpdateCommandExecution API 提供結果資訊，您可以在此頁面的結果索引標籤中找到此資訊。
- 從物件中樞頁面

  如果您在執行 命令時選擇 AWS IoT 物件做為目標裝置，您可以從物件中樞頁面檢視執行詳細資訊。

  1. 前往 AWS IoT 主控台中的物件中樞頁面，然後選擇您為其建立命令執行的物件。
  2. 在物件詳細資訊頁面的命令歷史記錄中，您會看到您建立的執行。選擇您要擷取資訊的執行。
  3. 如果您的裝置使用 UpdateCommandExecution API 提供結果資訊，您可以在此頁面的結果索引標籤中找到此資訊。

擷取命令執行 (CLI)

使用GetCommandExecution AWS IoT Core 控制平面 HTTP API 操作來擷取命令執行的相關資訊。您必須已使用 StartCommandExecution API 操作執行此命令。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示允許使用者執行 GetCommandExecution動作的 IAM 政策。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 us-east-1。
- *account-id* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。
- *command-id* 您的唯一 AWS IoT 命令識別符，例如 *LockDoor*。
- *devices* 使用 thing或 ，client取決於您的裝置是否已註冊為 AWS IoT 實物，或指定為 MQTT 用戶端。
- *device-id* 您的 AWS IoT thing-name或 client-id。

```
{
  "Effect": "Allow",
  "Action": [
      "iot:GetCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

擷取命令執行範例

下列範例說明如何擷取使用 命令執行之start-command-execution AWS CLI 命令的相關資訊。下列範例顯示如何擷取已執行之命令的相關資訊，以關閉方向盤模式。

在此範例中，取代：

- *<execution-id>* 具有您要擷取資訊的命令執行識別符。
- *<target-arn>* 您以執行為目標之裝置的 Amazon Resource Number (ARN)。您可以從 CLI start-command-execution 命令的回應取得此資訊。

- 或者，如果您的裝置使用 UpdateCommandExection API 來提供執行結果，您可以指定是否使用 GetCommandExecution API 將命令執行結果包含在 GetCommandExecution API 的回應中。

```
aws iot get-command-execution
    --execution-id <execution-id> \
    --target-arn <target-arn> \
    --include-result
```

執行此命令會產生回應，其中包含命令執行的 ARN、執行狀態，以及開始執行的時間和完成時間的相關資訊。它也提供 statusReason 物件，其中包含 狀態的其他資訊。如需不同狀態和狀態原因的詳細資訊，請參閱 命令執行狀態。

下列程式碼顯示來自 API 請求的範例回應。

> **Note**
>
> 執行回應中的 completedAt 欄位對應至裝置向雲端回報終端機狀態的時間。在TIMED_OUT狀態的情況下，只有在裝置報告逾時時，才會設定此欄位。狀態由雲端設定TIMED_OUT時，TIMED_OUT狀態不會更新。如需逾時行為的詳細資訊，請參閱 命令執行逾時考量。

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/myRegisteredThing",
    "status": "SUCCEEDED",
    "statusReason": {
        "reasonCode": "DEVICE_SUCCESSFULLY_EXECUTED",
        "reasonDescription": "SUCCESS"
    },
    "result": {
        "sn": { "s": "ABC-001" },
        "digital": { "b": true }
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "completedAt": "2024-03-23T00:50:10.095000-07:00"
}
```

使用 MQTT 測試用戶端檢視命令更新

您可以使用 MQTT 測試用戶端，在使用命令功能時透過 MQTT 檢視訊息交換。在您的裝置與 建立 MQTT 連線後 AWS IoT，您可以建立命令、指定承載，然後在裝置上執行它。當您執行 命令時，如果您的裝置訂閱命令的 MQTT 預留請求主題，它會看到發佈至此主題的承載訊息。

然後，裝置會收到承載指示，並在 AWS IoT 裝置上執行指定的操作。然後，它會使用 `UpdateCommandExecution` API 將命令執行結果和狀態資訊發佈至 command. AWS IoT Device Management listens 的 MQTT 預留回應主題，以更新回應主題，並將更新的資訊和發佈日誌至 AWS CloudTrail 和 Amazon CloudWatch。然後，您可以從主控台或使用 `GetCommandExecution` API 擷取最新的命令執行資訊。

下列步驟說明如何使用 MQTT 測試用戶端來觀察訊息。

1. 在 AWS IoT 主控台中開啟 MQTT 測試用戶端。
2. 在訂閱索引標籤上，輸入下列主題，然後選擇訂閱，其中 *<thingId>* 是您已註冊之裝置的物件名稱 AWS IoT。

   > **Note**
   >
   > 您可以從 AWS IoT 主控台的物件中樞頁面找到裝置的物件名稱。如果您尚未將裝置註冊為物件，您可以在 AWS IoT 從 Connect 裝置頁面連線至 時註冊裝置。

   ```
   $aws/commands/things/<thingId>/executions/+/request
   ```

3. （選用）在訂閱索引標籤上，您也可以輸入下列主題，然後選擇訂閱。

   ```
   $aws/commands/things/+/executions/+/response/accepted/json
   $aws/commands/things/+/executions/+/response/rejected/json
   ```

4. 當您啟動命令執行時，將使用裝置已訂閱的請求主題 ，將訊息承載傳送至裝置`$aws/commands/things/<thingId>/executions/+/request`。在 MQTT 測試用戶端中，您應該會看到命令承載，其中包含裝置處理命令的指示。
5. 裝置開始執行命令後，可以發佈狀態更新至下列命令的 MQTT 預留回應主題。

   ```
   $aws/commands/<devices>/<device-id>/executions/<executionId>/response/json
   ```

例如，請考慮您執行的命令，以開啟汽車的 AC，將溫度降至所需的值。下列 JSON 顯示 車輛發佈至回應主題的範例訊息，顯示無法執行 命令。

```
{
  "deviceId": "My_Car",
  "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
  "status": "FAILED",
  "statusReason": {
    "reasonCode": "CAR_LOW_ON_BATTERY",
    "reasonDescription": "Car battery is lower than 5 percent"
  }
}
```

在這種情況下，您可以為汽車的電池充電，然後再次執行 命令。

在 中列出命令執行 AWS 帳戶

執行命令之後，您可以從 AWS IoT 主控台和使用 擷取命令執行的相關資訊 AWS CLI。您可以取得下列資訊。

- 執行 ID，這是命令執行作業的唯一識別碼。
- 命令執行的狀態。當您在目標裝置上執行命令時，命令執行作業會進入 CREATED 狀態。然後作業會轉換成其他命令執行狀態，如下所述。
- 唯一的命令 ID 和已建立執行作業的目標裝置。
- 開始日期，顯示命令執行作業建立的時間。

列出您帳戶中的命令執行 （主控台）

您可以使用下列其中一種方法，從主控台查看所有命令執行。

- 從命令中樞頁面

  前往 AWS IoT 主控台的 Command Hub 頁面，並執行這些步驟。

  1. 選擇您在目標裝置上建立執行的命令。
  2. 在命令詳細資訊頁面中，前往命令歷史記錄索引標籤，您會看到您建立的執行清單。
- 從物件中樞頁面

如果您在執行 命令時選擇 AWS IoT 物件做為目標裝置，並為單一裝置建立多個命令執行，您可以從物件中樞頁面檢視裝置的執行。

1. 前往 AWS IoT 主控台中的物件中樞頁面，然後選擇您為其建立執行的物件。
2. 在物件詳細資訊頁面的命令歷史記錄中，您會看到為裝置建立的執行清單。

列出您帳戶中的命令執行 (CLI)

使用ListCommandExecutions AWS IoT Core 控制平面 HTTP API 操作列出您帳戶中的所有命令執行。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您在裝置上執行此動作。下列範例顯示 IAM 政策，允許使用者執行 ListCommandExecutions動作的許可。

在此範例中，取代：

- *region* 搭配您的 AWS 區域，例如 us-east-1。
- *account-id* 您的 AWS 帳戶 號碼，例如 *123456789012*。
- *command-id* 您的唯一 AWS IoT 命令識別符，例如 *LockDoor*。

```
{
  "Effect": "Allow",
  "Action": "iot:ListCommandExecutions",
  "Resource": *
}
```

列出命令執行範例

下列範例示範如何在 中列出命令執行 AWS 帳戶。

執行 命令時，您必須指定是否要篩選清單，以僅顯示使用 為特定裝置建立的命令執行targetArn，還是使用 為特定命令指定的執行commandArn。

在此範例中，取代：

- *<target-arn>* 搭配您以執行為目標之裝置的 Amazon Resource Number (ARN)，例如 arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f*。

- *<target-arn>* 搭配您以執行為目標之裝置的 Amazon Resource Number (ARN)，例如 arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f*。
- *<after>* 以及您想要列出建立之執行的時間，例如 2024-11-01T03:00。

```
aws iot list-command-executions \
--target-arn <target-arn> \
--started-time-filter '{after=<after>}' \
--sort-order "ASCENDING"
```

執行此命令會產生回應，其中包含您建立的命令執行清單、執行開始執行的時間，以及執行完成的時間。它也提供狀態資訊，以及包含狀態額外資訊的statusReason物件。

```
{
    "commandExecutions": [
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "b2b654ca-1a71-427f-9669-e74ae9d92d24",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "TIMED_OUT",
            "createdAt": "2024-11-24T14:39:25.791000-08:00",
            "startedAt": "2024-11-24T14:39:25.791000-08:00"
        },
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "34bf015f-ef0f-4453-acd0-9cca2d42a48f",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "IN_PROGRESS",
            "createdAt": "2024-11-24T14:05:36.021000-08:00",
            "startedAt": "2024-11-24T14:05:36.021000-08:00"
        }
    ]
}
```

如需不同狀態和狀態原因的詳細資訊，請參閱 命令執行狀態。

刪除命令執行

如果您不想再使用命令執行，可以從您的帳戶永久移除它。

> **Note**
>
> - 命令執行只有在進入終端狀態時才能刪除，例如 SUCCEEDED、 FAILED或 REJECTED。
> - 此操作只能使用 AWS IoT Core API 或 執行 AWS CLI。它不適用於 主控台。

範例 IAM 政策

使用此 API 操作之前，請確定您的 IAM 政策授權您的裝置執行這些動作。以下顯示範例政策，授權您的裝置執行 動作。

在此範例中，取代：

- *Region* ， AWS 區域例如 us-east-1。
- *AccountID* 使用您的 AWS 帳戶 號碼，例如 *123456789012*。
- *CommandID* 具有您要刪除執行之命令的識別符。
- *devices* 使用 thing或 ，client取決於您的裝置是否已註冊為 AWS IoT 實物，或指定為 MQTT 用戶端。
- *device-id* 您的 AWS IoT thing-name或 client-id。

```
{
  "Effect": "Allow",
  "Action": [
      "iot:DeleteCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

刪除命令執行範例

下列範例示範如何使用 命令刪除delete-command AWS CLI 命令。根據您的應用程式，將 *<execution-id>*取代為您要刪除之命令執行的識別符，並將 *<target-arn>*取代為您目標裝置的 ARN。

```
aws iot delete-command-execution \
```

```
--execution-id <execution-id> \
--target-arn <target-arn>
```

如果 API 請求成功，則命令執行會產生 200 狀態碼。您可以使用 GetCommandExecution API 來驗證 帳戶中不再存在命令執行。

取代指令資源

棄用命令以指出它們已過時且不應使用。例如，棄用不再主動維護的命令，或在建立具有相同 ID 但不同承載的較新命令時。

關鍵考量

取代命令時的重要考量事項：

- 棄用命令並不會將其刪除。您可以使用命令的 ID 擷取命令，並將其還原以重複使用。
- 嘗試在已棄用命令上啟動新執行會產生錯誤，防止使用過時的命令。
- 若要執行已棄用命令，請先將其還原。還原之後，命令即可在目標裝置上定期使用和執行。
- 如果您在執行進行時棄用命令，它們會繼續執行，直到完成為止。您仍然可以擷取執行狀態。

棄用命令資源 （主控台）

若要從主控台棄用命令，請前往 AWS IoT 主控台的 Command Hub 並執行下列步驟。

1. 選擇您要棄用的命令，然後在動作下選擇棄用。
2. 確認您要棄用命令，然後選擇棄用。

棄用命令資源 (CLI)

使用 CLI update-command 將命令標記為已棄用。您必須先棄用命令，才能刪除。若要使用已棄用的命令，請先將其還原。

```
aws iot update-command \
    --command-id <command-id> \
    --deprecated
```

例如，如果您已棄用在上述範例中更新的*ACSwitch*命令，下列程式碼會顯示執行命令的範例輸出。

```
{
    "commandId": "turnOffAc",
    "deprecated": true,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
}
```

檢查棄用時間和狀態

使用 GetCommand API 來判斷命令是否已棄用，以及上次棄用的時間。

```
aws iot get-command --command-id <turnOffAC>
```

此命令會產生包含命令資訊的回應，包括上次更新欄位的建立和取代時間戳記。這有助於判斷命令生命週期，以及是否要刪除或重複使用。以下顯示 *turnOffAc*命令的範例回應：

```
{
    "commandId": "turnOffAC",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/turnOffAC",
    "namespace": "AWS-IoT",
    "payload": {
        "content": "testPayload.json",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00",
    "deprecated": false
}
```

還原命令資源

若要使用或傳送ACSwitch命令至您的裝置，請先將其還原。

若要從主控台還原命令，請前往 AWS IoT 主控台的 Command Hub，選擇您要還原的命令，然後在動作下選擇還原。

若要使用 AWS IoT Core API 或 還原命令 AWS CLI，請使用 UpdateCommand API 操作或 update-command CLI。下列程式碼顯示範例請求和回應。

```
aws iot update-command \
    --command-id <command-id>
```

```
    --no-deprecated
```

下列程式碼顯示範例輸出。

```
{
    "commandId": "ACSwitch",
    "deprecated": false,
    "lastUpdatedAt": "2024-05-09T23:17:21.954000-07:00"
}
```

AWS IoT 安全通道

在遠端網站的受限制防火牆後面部署裝置時，您需要一種方法來存取這些裝置，進行疑難排解、組態更新及其他作業任務。使用安全通道，透過 管理的安全連線，與遠端裝置建立雙向通訊 AWS IoT。安全通道不需要更新現有的輸入防火牆規則，因此您可以在遠端網站保留防火牆規則所提供的相同安全層級。

例如，位於幾百英里外工廠的感應器裝置在測量工廠溫度時遇到麻煩。您可以使用安全通道，開啟並快速啟動該感應器裝置的工作階段。找出問題之後 (例如，不當的組態檔)，您可以重設檔案並透過相同的工作階段重新啟動感應器裝置。相較於傳統的故障診斷 (例如，派遣技術人員到工廠調查感應器裝置)，安全通道可減少事故回應和回復時間，以及操作成本。

什麼是安全通道？

使用安全通道存取部署在遠端站點連接埠受限防火牆後的裝置。您可使用 AWS 雲端，從作為來源裝置的筆記型電腦或桌上型電腦連線至目的地裝置。來源和目的地會使用在每個裝置上執行的開放原始碼本機代理進行通訊。本機代理 AWS 雲端 會使用防火牆允許的開放連接埠與 通訊，通常是 443。透過通道傳輸的資料會使用 Transport Layer Security (TLS) 加密。

主題

- 安全通道概念
- 安全通道運作方式
- 安全通道生命週期

安全通道概念

與遠端裝置建立通訊時，安全通道使用以下術語。如需安全通道運作方式的資訊，請參閱 安全通道運作方式。

用戶端存取字符 (CAT)

> 建立新通道時，由安全通道產生的一對字符。CAT 是由來源和目標裝置用來連線至安全通道服務。CAT 只能用於連接通道一次。若要重新連線至通道，請使用 RotateTunnelAccessToken API 操作或 rotate-tunnel-access-token CLI 命令輪換用戶端存取字符。

用戶端字符

AWS IoT 安全通道的用戶端所產生的唯一值，可用於所有後續重試連線到相同通道。此欄位為選用欄位。如果未提供用戶端字符，則用戶端存取字符 (CAT) 只能用於同一通道一次。使用相同 CAT 的後續連線嘗試會被拒絕。如需有關使用用戶端字符的詳細資訊，請參閱在 GitHub 中的本地代理參考實作。

目標應用程式

在目標裝置上執行的應用程式。例如，目標應用程序可以是一個 SSH 協助程式，用於使用安全通道建立 SSH 工作階段。

目標裝置

您要存取的遠端裝置。

裝置代理程式

連線至 AWS IoT 裝置閘道並透過 MQTT 接聽新通道通知的 IoT 應用程式。如需詳細資訊，請參閱IoT Agent Snippet。

本機代理

在來源和目標裝置上執行，並在安全通道與裝置應用程式之間轉送資料串流的軟體代理。本機代理可以在源模式或目標模式下執行。如需詳細資訊，請參閱本機代理。

來源裝置

此裝置通常是筆記型電腦或桌上型電腦，操作人員用來啟動目標裝置的工作階段。

通道

透過 的邏輯路徑 AWS IoT ，可啟用來源裝置與目的地裝置之間的雙向通訊。

安全通道運作方式

下面顯示了安全通道如何在來源裝置和目的地裝置之間建立連線。如需用戶端存取字符 (CAT) 等不同術語的資訊，請參閱 安全通道概念。

1. 開啟通道

   若要使用遠端目的地裝置開啟啟動工作階段的通道，您可以使用 AWS 管理主控台、AWS CLI Open-tunnel 命令或 OpenTunnel API。

2. 下載用戶端存取字符對

開啟通道後，您可以下載來源和目的地的用戶端存取字符 (CAT)，並將其儲存在來源裝置上。在啟動本機代理之前，必須擷取 CAT 並立即儲存。

3. 以目的地模式啟動本機代理

   已安裝並正在目的地裝置上執行的 IoT 代理將訂閱保留的 MQTT 主題 `$aws/things/thing-name/tunnels/notify` 並將收到 CAT。在這裡，`####`是您為目的地建立的 AWS IoT 實物名稱。如需詳細資訊，請參閱安全通道主題。

   然後，IoT 代理使用 CAT 在目的地模式下啟動本機代理，並在通道的目的地端設定連線。如需詳細資訊，請參閱IoT Agent Snippet。

4. 以來源模式啟動本機代理

   通道開啟後， AWS IoT Device Management 會提供來源的 CAT，您可以在來源裝置上下載。您可以使用 CAT 在來源模式下啟動本機代理，然後連線通道的來源端。如需本機代理的詳細資訊，請參閱 本機代理。

5. 開啟 SSH 工作階段

   當通道兩端都連線時，您可以使用來源端的本機代理啟動 SSH 工作階段。

如需如何使用 AWS 管理主控台 開啟通道並啟動 SSH 工作階段的詳細資訊，請參閱 開啟通道並啟動遠端裝置的 SSH 工作階段。

下列影片介紹了安全通道的運作方式，並引導您完成設定與 Raspberry Pi 裝置的 SSH 工作階段的程序。

安全通道生命週期

通道可以具有 `OPEN` 或 `CLOSED` 狀態。與通道的連線狀態可為 `CONNECTED` 或 `DISCONNECTED`。下面顯示了不同的通道和連線狀態的運作方式。

1. 當您開啟通道時，它的狀態為 `OPEN`。通道的來源和目標連線狀態會設定為 `DISCONNECTED`。
2. 當裝置 (來源或目標) 連線至通道時，對應的連線狀態會變更為 `CONNECTED`。
3. 當裝置與通道中斷連線而通道狀態維持為 `OPEN` 時，對應的連線狀態會變回 `DISCONNECTED`。只要通道仍然保持 `OPEN`，裝置就可以重複連線至通道，以及與通道中斷連線。

> **Note**
>
> 用戶端存取字符 (CAT) 只能用於連接通道一次。若要重新連線至通道，請使用 RotateTunnelAccessToken API 操作或 rotate-tunnel-access-token CLI 命令輪換用戶端存取字符。如需範例，請參閱 透過輪換用戶端存取字符解決 AWS IoT 安全通道連線問題。

4. 當您呼叫 `CloseTunnel` 或通道維持 `OPEN` 的時間超過 `MaxLifetimeTimeout` 值時，通道的狀態就會變成 `CLOSED`。您可以在呼叫 `OpenTunnel` 時設定 `MaxLifetimeTimeout`。如果您未指定一值，則 `MaxLifetimeTimeout` 會預設為 12 小時。

> **Note**
>
> 狀態為 `CLOSED` 時，無法重新開啟通道。

5. 通道顯示時，您可以呼叫 `DescribeTunnel` 和 `ListTunnels` 來檢視通道中繼資料。通道在刪除之前，可以在 AWS IoT 主控台中看到至少三個小時。

AWS IoT 安全通道教學課程

AWS IoT 安全通道協助客戶透過 管理的安全連線，與防火牆後方的遠端裝置建立雙向通訊 AWS IoT。

若要示範 AWS IoT 安全通道，請使用 AWS IoT GitHub 上的安全通道示範。

下列教學課程將會協助您了解如何開始及使用安全通道。您將學到如何：

1. 使用快速設定和手動設定方法建立安全通道以存取遠端裝置。
2. 在使用手動設定方法時設定本機代理，並連線至通道以存取目的地裝置。
3. 無需設定本機代理即可透過瀏覽器使用 SSH 進入遠端裝置。
4. 轉換使用 AWS CLI 或使用手動設定方法建立的通道，以使用快速設定方法。

本節中的教學課程

本節中的教學課程著重於使用 AWS 管理主控台 和 AWS IoT API 參考建立通道。在 AWS IoT 主控台中，您可以從通道中樞頁面或您所建立物件的詳細資訊頁面建立通道。如需詳細資訊，請參閱AWS IoT 主控台中的通道建立方法。

本節教學課程如下所示：

- 開啟通道並使用瀏覽器型 SSH 來存取遠端裝置

  本教學課程說明如何使用快速設定方法從 Tunnels hub (通道中樞) 頁面開啟通道。您也將了解如何使用瀏覽器型 SSH，在 AWS IoT 主控台中使用內容內命令列界面存取遠端裝置。

- 使用手動設定開啟通道並連線至遠端裝置

  本教學課程說明如何使用手動設定方法從 Tunnels hub (通道中樞) 頁面開啟通道。您也將了解如何從來源裝置中的終端設定和啟動本機代理並連接到通道。

- 開啟遠端裝置的通道並使用瀏覽器型 SSH

  本教學課程將介紹如何從您所建立物件的詳細資訊頁面開啟通道。您將了解如何建立新的通道和使用現有通道。現有通道會對應至為裝置建立的最新開放通道。您也可以使用瀏覽器型 SSH 來存取遠端裝置。

AWS IoT 安全通道教學課程

- 開啟通道並啟動遠端裝置的 SSH 工作階段
- 開啟遠端裝置的通道並使用瀏覽器型 SSH

開啟通道並啟動遠端裝置的 SSH 工作階段

在這些教學課程中，您將了解如何從遠端存取防火牆後方的裝置。您無法在裝置中啟動直接 SSH 工作階段，因為防火牆會封鎖所有傳入流量。教學課程會示範如何開啟通道，然後使用該通道啟動遠端裝置的 SSH 工作階段。

教學課程的先決條件

執行教學課程的先決條件可能不盡相同，取決於您是使用手動或是快速設定方法來開啟通道和存取遠端裝置。

> **Note**
>
> 對於這兩種設定方法，您都必須允許連接埠 443 的傳出流量。

- 如需了解快速設定方法自學課程之先決條件，請參閱 快速設定方法的先決條件。

- 如需了解手動設定方法自學課程之先決條件，請參閱 手動設定方法的先決條件。如果您使用此設定方法，則必須在來源裝置上設定本機代理。若要下載本機代理原始碼，請參閱 GitHub 上的本機代理參考實作。

通道設定方法

在這些教學課程中，您將了解如何以手動和快速設定方法開啟通道和連線至遠端裝置。下表顯示設定方法之間的差異。建立通道之後，您就可以使用瀏覽器內的命令列介面，透過 SSH 存取遠端裝置。如果誤置權杖或通道連線中斷，您可以傳送新的存取權杖以重新連線至通道。

快速和手動設定方法

條件	快速設定	手動設定
通道建立	使用預設、可編輯的組態來建立新的通道。若要存取遠端裝置，您只能使用 SSH 作為目的地服務。	透過手動指定通道組態來建立通道。您可以使用此方法，透過 SSH 以外的服務連線至遠端裝置。
存取權杖	若在建立通道時指定物件名稱，目的地存取權杖會自動傳遞至保留 MQTT 主題上的裝置。您無需透過來源裝置下載或管理權仗。	您必須透過來源裝置手動下載和管理權杖。若在建立通道時指定物件名稱，目的地存取權杖會自動傳遞至保留 MQTT 主題上的遠端裝置。
本機代理	系統會自動為您設定 Web 型本機代理，以便與裝置進行互動。您無需手動設定本機代理。	您必須手動設定並啟動本機代理。若要設定本機代理，您可以使用 AWS IoT 裝置用戶端，或在 GitHub 上下載本機代理參考實作。

AWS IoT 主控台中的通道建立方法

本節教學課程將會介紹如何使用 AWS 管理主控台 和 Openunnel API 來建立通道。如果您在建立通道時設定目的地， AWS IoT 安全通道會透過 MQTT 和預留 MQTT 主題 ，將目的地用戶端存取字符傳遞至遠端裝置`$aws/things/RemoteDeviceA/tunnels/notify`)。在收到 MQTT 訊息後，遠端裝置上的 IoT 代理程式會以目的地模式啟動本機代理。如需詳細資訊，請參閱預留主題。

> **Note**
>
> 如果您想要透過其他方法將目標用戶端存取字符交付給遠端裝置，則可以省略目標組態。如需詳細資訊，請參閱設定遠端裝置並使用 IoT 代理程式。

在 AWS IoT 主控台中，您可以使用下列其中一種方法建立通道。如需更多有關使用這些方法建立通道的輔助教學課程資訊，請參閱 本節中的教學課程。

- 通道中樞

  建立通道時，您可以指定要使用快速設定或手動設定方法來建立通道，並提供選擇性的通道組態詳細資訊。組態詳細資訊也包括目的地裝置的名稱，以及您要用來連線到裝置的服務。建立通道之後，您可以在瀏覽器內使用 SSH，或在 AWS IoT 主控台外開啟終端機以存取遠端裝置。
- 物件詳細資訊頁面

  建立通道時，除了選擇設定方法並提供任何選擇性的通道組態詳細資訊之外，您還可以指定是否要使用最新開放通道或為裝置建立新的通道。您無法編輯現有通道的組態詳細資訊。您可以使用快速設定方法輪換存取權杖，並在瀏覽器中使用 SSH 進入遠端裝置。若要使用此方法開啟通道，您必須在 AWS IoT 登錄檔中建立 IoT 物件 （例如 `RemoteDeviceA`)。如需詳細資訊，請參閱在 AWS IoT 登錄檔中註冊裝置。

本節中的教學課程

- 開啟通道並使用瀏覽器型 SSH 來存取遠端裝置
- 使用手動設定開啟通道並連線至遠端裝置

開啟通道並使用瀏覽器型 SSH 來存取遠端裝置

您可以使用快速設定或手動設定方法來建立通道。本教學課程說明如何使用快速設定方法開啟通道，並使用瀏覽器型 SSH 連線至遠端裝置。如需查看如何使用手動設定方法來開啟通道的範例，請參閱 使用手動設定開啟通道並連線至遠端裝置。

您可以透過快速設定方法，使用可編輯的預設組態建立新的通道。已為您設定 Web 型本機代理，並使用 MQTT 將存取權杖自動傳遞到您的遠端目的地裝置。建立通道後，您可以使用主控台內的命令列介面開始與遠端裝置互動。

若採用快速設定方法，您必須使用 SSH 作為目的地服務才能存取遠端裝置。如需不同設定方法的詳細資訊，請參閱 通道設定方法。

快速設定方法的先決條件

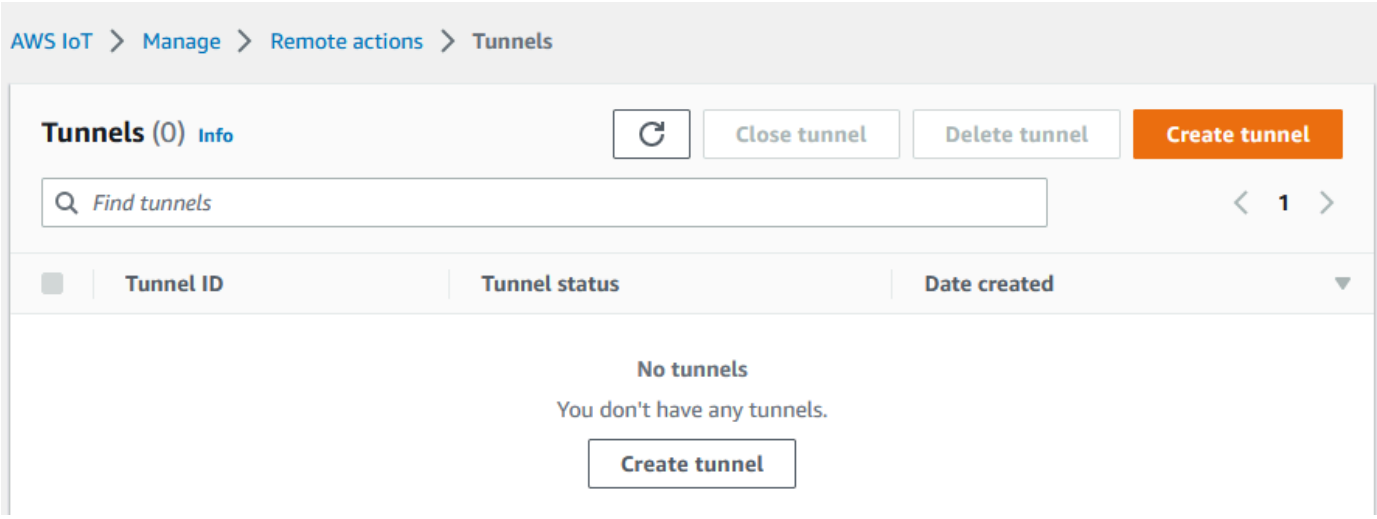
- 遠端裝置所在的防火牆必須允許連接埠 443 上的傳出流量。您建立的通道將使用此通訊埠連線至遠端裝置。
- 您有 IoT 裝置代理程式 （ 請參閱 IoT Agent Snippet) 在連線至 AWS IoT 裝置閘道的遠端裝置上執行，並使用 MQTT 主題訂閱進行設定。如需詳細資訊，請參閱將裝置連線至 AWS IoT 裝置閘道。
- 您必須具有一個在遠端裝置上執行的 SSH 常駐程式。

開啟通道

您可以使用 AWS 管理主控台、 AWS IoT API 參考或 來開啟安全通道 AWS CLI。您可以選擇設定目的地名稱，但這並非本教學課程的必要步驟。如果您設定目的地，安全通道會使用 MQTT 自動將存取權杖傳遞至遠端裝置。如需詳細資訊，請參閱AWS IoT 主控台中的通道建立方法。

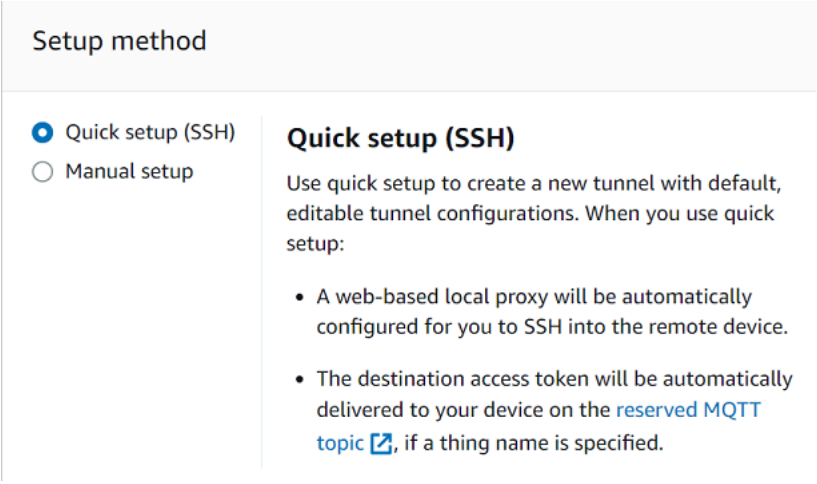
若要使用主控台中開啟通道

1. 前往 AWS IoT 主控台的 Tunnels hub (通道中樞)，然後選擇 Create job (建立通道)。



2. 在本教學課程中，請選擇 Quick setup (快速設定) 作為通道建立方法，然後選擇 Next (下一步)。

> **Note**
>
> 如果您從建立物件的詳細資訊頁面建立安全通道，您可以選擇要建立新通道或使用現有通道。如需詳細資訊，請參閱開啟遠端裝置的通道並使用瀏覽器型 SSH。

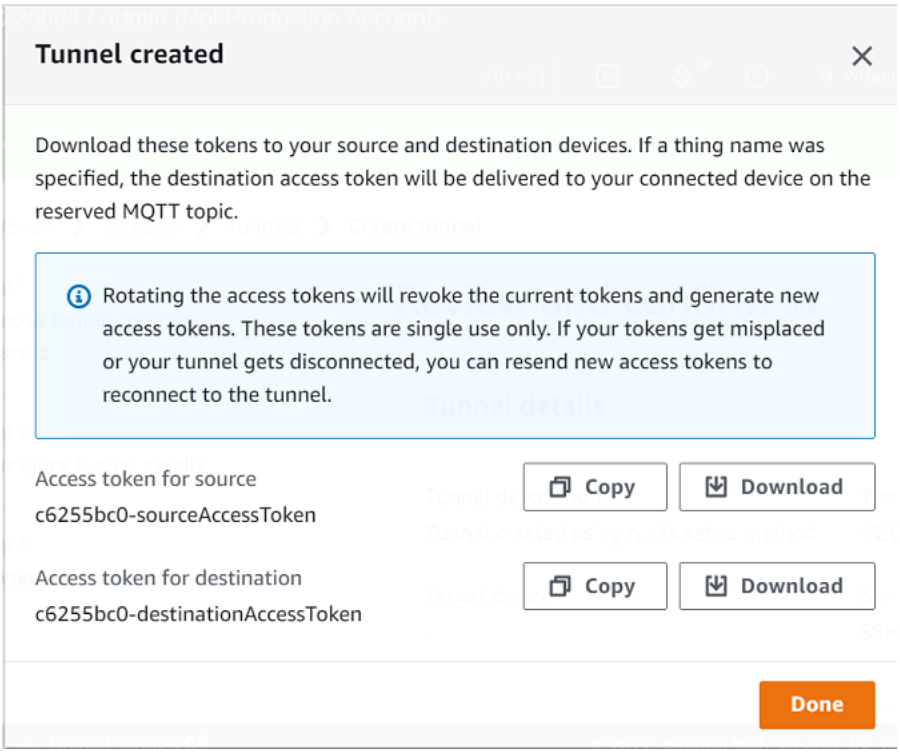


3. 檢閱並確認通道組態詳細資訊。若要建立通道，請選擇 Confirm and create (確認並建立)。如果您要編輯這些詳細資訊，請選擇 Previous (上一步) 返回上一頁，然後確認並建立通道。

> **Note**
>
> 使用快速設定時，將無法編輯服務名稱。您必須使用 SSH 作為服務。

4. 若要建立通道，請選擇 Done (完成)。

在本教學課程中，您無需下載來源或目的地存取權杖。這些權杖只能用於連接通道一次。如果您的通道中斷連線，您可以產生新權杖並將其傳送到遠端裝置，以重新連線至通道。如需詳細資訊，請參閱重新傳送通道存取權杖。



若要使用主控台中開啟通道

若要開放新的通道，您可以使用 OpenTunnel API 操作。

> **Note**
>
> 您只能從 AWS IoT 主控台使用快速設定方法建立通道。當您使用 AWS IoT API 參考 API 或 時 AWS CLI，它會使用手動設定方法。您可以開啟您所建立的現有通道，然後變更通道的設定方法，即可使用快速設定。如需詳細資訊，請參閱開啟現有的通道並使用瀏覽器型 SSH。

關於如何執行此 API 操作的範例如下所示。如果您要指定物件名稱和目的地服務，也可以選擇使用 `DestinationConfig` 參數。如需示範如何使用這個參數的範例，請參閱 為遠端裝置開啟新通道。

```
aws iotsecuretunneling open-tunnel
```

執行此命令會建立新的通道，並提供您來源和目的地存取權杖。

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
```

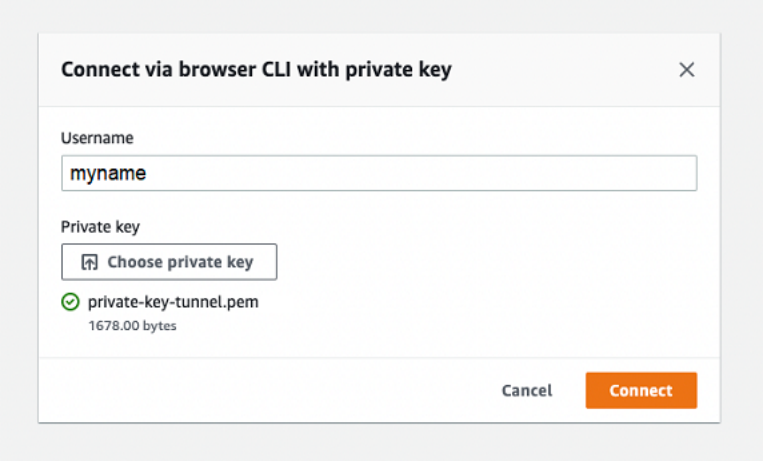
```
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

使用瀏覽器型 SSH

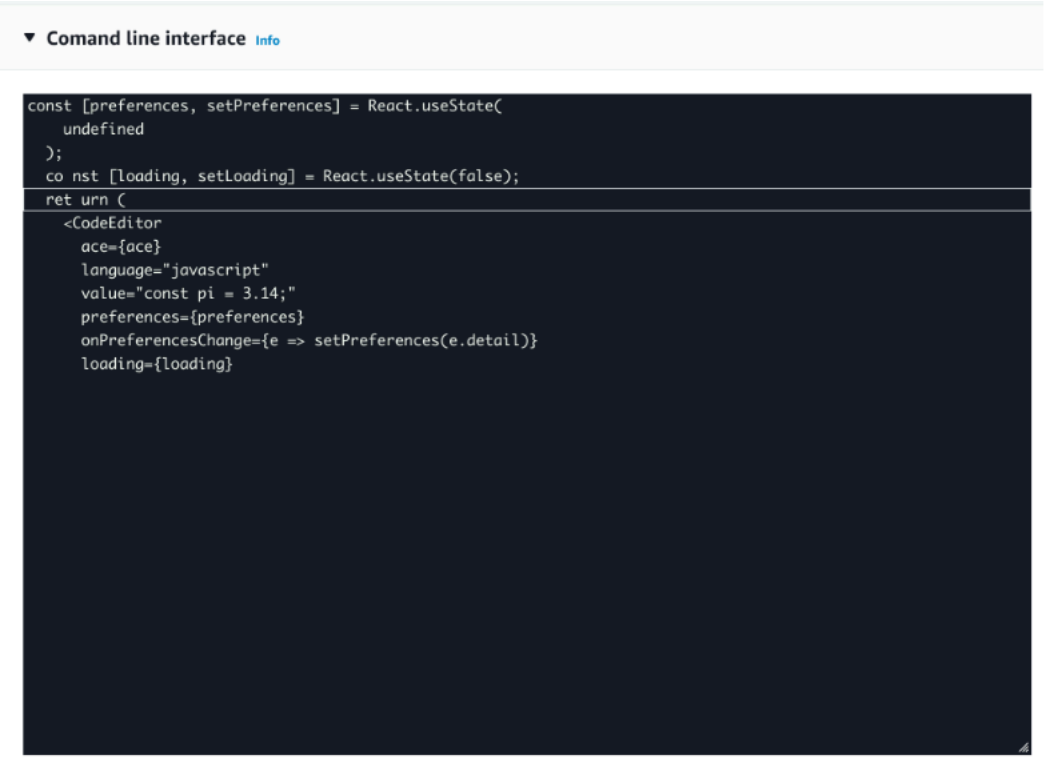
在使用快速設定方法建立通道並且目的地裝置已連線至通道後，您就可以使用瀏覽器型 SSH 存取遠端裝置。透過瀏覽器型 SSH，您可以在主控台內的上下文命令列介面中輸入命令，即可直接與遠端裝置進行通訊。此功能可讓您更輕鬆地與遠端裝置互動，因為您不必在主控台外開啟終端或設定本機代理。

若要使用瀏覽器型 SSH

1. 前往 AWS IoT 主控台的 Tunnels hub (通道中樞)，然後選擇您建立的通道以檢視其詳細資訊。
2. 展開 Secure Shell (SSH) 區段，然後選擇 Connect (連線)。
3. 選擇是否要提供您的使用者名稱和密碼來驗證 SSH 連線，或者如需更安全的驗證，則可以使用裝置的私有金鑰。如果您使用私有金鑰進行身分驗證，請注意，只有 PEM 格式 (256 和 512) RSA 金鑰可與 AWS IoT Secure Tunneling SSH 主控台搭配使用。
   - 若要使用使用者名稱和密碼進行連線，請選擇 Use password (使用密碼)。然後，您可以輸入使用者名稱和密碼，並開始使用瀏覽器內的 CLI。
   - 若要使用目的地裝置的私有金鑰進行連線，請選擇 Use private key (使用私有金鑰)。指定您的使用者名稱並上傳裝置的私有金鑰檔案，然後選擇 Connect (連線) 以開始使用瀏覽器內 CLI。



在通過 SSH 連接驗證後，您即可迅速開始使用瀏覽器 CLI 輸入命令並與裝置進行互動，因為系統已為您設定本機代理。



如果瀏覽器 CLI 在通道持續時間之後保持開啟狀態，則可能會發生逾時，導致命令列介面中斷連線。您可以複製通道並啟動另一工作階段，以便在主控台本身內部與遠端裝置互動。

解決使用瀏覽器型 SSH 時發生的問題

以下介紹如何解決使用瀏覽器型 SSH 時可能發生的一些問題。

- 系統顯示錯誤，而不是命令列介面

  您可能會看到錯誤，原因是目的地裝置中斷連線。您可以選擇 Generate new access tokens (產生新的存取權杖) 來產生新的存取權杖，並使用 MQTT 將權杖傳送到遠端裝置。新的權杖可以用來重新連線至通道。重新連線至通道會清除歷程記錄並重新整理命令列工作階段。

- 使用私有金鑰進行驗證時，您會看到通道中斷連線的錯誤

  您可能會看到錯誤訊息，原因可能是目的地裝置尚未接受您的私有金鑰。若要解決此錯誤，請檢查您上傳用於進行驗證的私有金鑰檔案。如果仍然看到錯誤，請檢查您的裝置日誌。您也可以向遠端裝置發送新的存取權杖，藉此嘗試重新連線至通道。

- 您的通道已在使用工作階段時關閉

  如果您的通道因保持開啟超過指定持續時間而關閉，則您的命令列工作階段可能會中斷連線。通道一旦關閉就不能重新開放。若要重新連線，您必須開啟另一個連往裝置的通道。

您可以複製通道，以建立與已關閉通道相同組態的新通道。您可以從 AWS IoT 主控台複製封閉的通道。若要複製通道，請選擇已關閉的通道以檢視其詳細資訊，然後選擇 Duplicate tunnel (複製通道)。指定您要使用的通道持續時間，然後建立新通道。

清除

- 關閉通道

  我們建議您在使用完通道後關閉通道。如果開放時間超過指定的通道持續時間，通道也可能會關閉。通道一旦關閉就不能重新開放。您仍然可以選擇已關閉的通道，然後選擇 Duplicate tunnel (複製通道) 來複製通道。指定您要使用的通道持續時間，然後建立新通道。

  - 若要從 AWS IoT 主控台關閉個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要關閉的通道，然後選擇Close tunnel (關閉通道)。
  - 若要使用 AWS IoT API 參考 API 關閉個別通道或多個通道，請使用 CloseTunnel API。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 刪除通道

  您可以從 永久刪除通道 AWS 帳戶。

  > **Warning**
  > 刪除動作為永久性動作，且無法還原。

  - 若要從 AWS IoT 主控台刪除個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要刪除的通道，然後選擇Delete tunnel (刪除通道)。
  - 若要使用 AWS IoT API 參考 API 刪除個別通道或多個通道，請使用 CloseTunnel API。使用 API 時，請將 `delete` 旗標設定為 `true`。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
        --delete true
    ```

使用手動設定開啟通道並連線至遠端裝置

開啟通道時，您可以選擇以快速設定或手動設定方法來開啟通道進入遠端裝置。本教學課程說明如何使用手動設定方法開啟通道，並設定及啟動本機代理以連線至遠端裝置。

若使用手動設定方法，則必須在建立通道時手動指定通道組態。建立通道之後，您可以在瀏覽器中 SSH，或在 AWS IoT 主控台外部開啟終端機。本教學課程說明如何使用主控台外部的終端存取遠端裝置。您也將了解如何設定本機代理，然後連線至本機代理以與遠端裝置進行互動。若要連線至本機代理，您必須在建立通道時下載來源存取權杖。

在此設定方法中，您可以使用 SSH 以外的服務，例如透過 FTP 連線至遠端裝置。如需不同設定方法的詳細資訊，請參閱 通道設定方法。

手動設定方法的先決條件

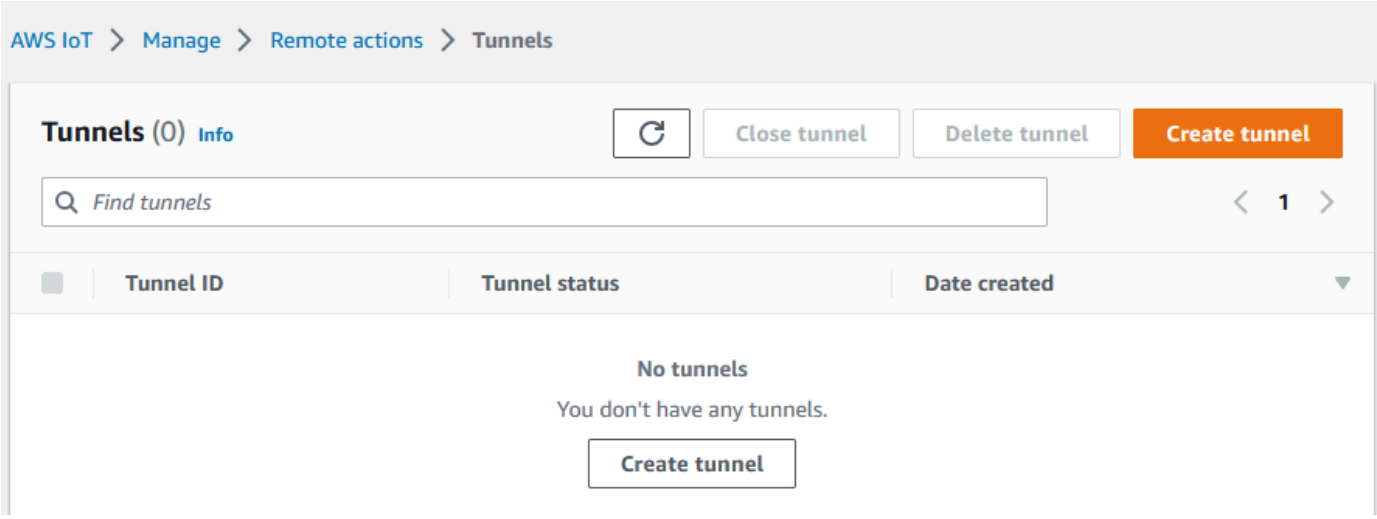
- 遠端裝置所在的防火牆必須允許連接埠 443 上的傳出流量。您建立的通道將使用此通訊埠連線至遠端裝置。
- 您有 IoT 裝置代理程式 （ 請參閱 IoT Agent Snippet) 在連線至 AWS IoT 裝置閘道的遠端裝置上執行，並使用 MQTT 主題訂閱進行設定。如需詳細資訊，請參閱將裝置連線至 AWS IoT 裝置閘道。
- 您必須具有一個在遠端裝置上執行的 SSH 常駐程式。
- 您已從 GitHub 下載了本機代理原始碼，並為您選擇的平台建置它。在本教學課程中，我們將建置的本機代理可執行檔案稱作 `localproxy`。

開啟通道

您可以使用 AWS 管理主控台、 AWS IoT API 參考或 來開啟安全通道 AWS CLI。您可以選擇設定目的地名稱，但這並非本教學課程的必要步驟。如果您設定目的地，安全通道會使用 MQTT 自動將存取權杖傳遞至遠端裝置。如需詳細資訊，請參閱AWS IoT 主控台中的通道建立方法。

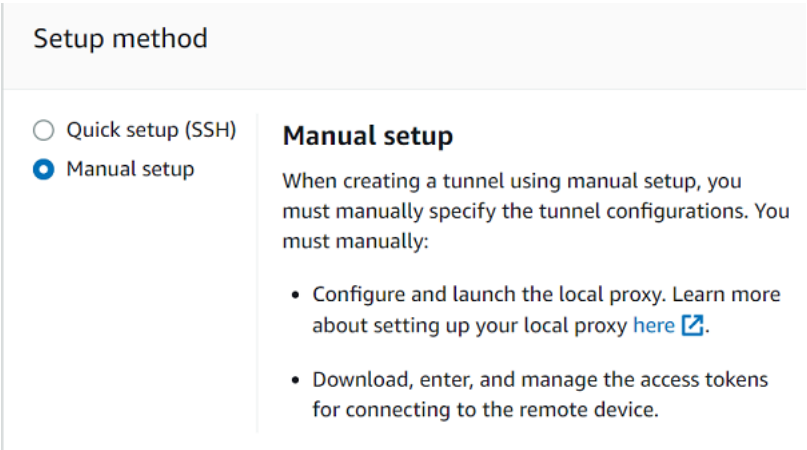
如要在主控台中開啟通道

1. 前往 AWS IoT 主控台的 Tunnels hub (通道中樞)，然後選擇 Create job (建立通道)。



2. 在本教學課程中，請選擇 Manual setup (手動設定) 作為通道建立方法，然後選擇 Next (下一步)。如需了解如何使用快速設定方法建立通道，請參閱 開啟通道並使用瀏覽器型 SSH 來存取遠端裝置。

> **Note**
>
> 如果從建立物件的詳細資訊頁面建立安全通道，您可以選擇要建立新通道或使用現有通道。如需詳細資訊，請參閱開啟遠端裝置的通道並使用瀏覽器型 SSH。



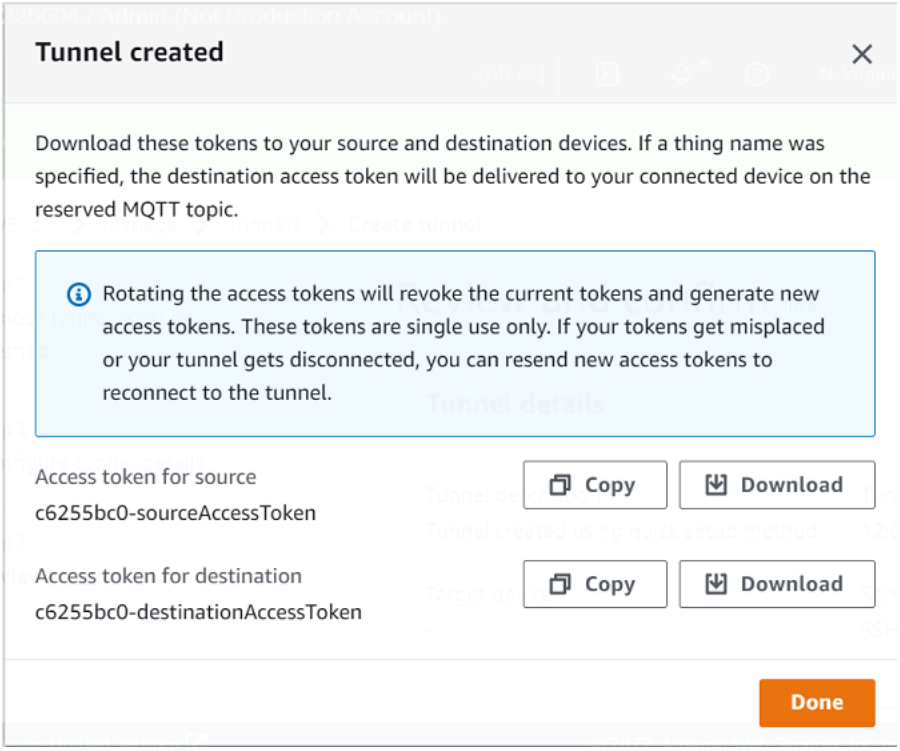
3. (選擇性) 輸入通道的組態設定。您也可以略過此步驟，繼續進行下一個步驟來建立通道。

   輸入通道描述、通道逾時持續時間和資源標籤作為索引鍵值對，以協助您識別資源。在本教學課程中，您可以略過目的地組態。

> **Note**
>
> 費用不會依據通道保持開放的持續時間進行計價。只有在建立新通道時會產生費用。如需定價資訊，請參閱 AWS IoT Device Management 定價中的安全通道。

4. 下載用戶端存取權杖，然後選擇 Done (完成)。選擇 Done (完成) 後，字符將無法下載。

   這些權杖只能用於連接通道一次。如果您誤置權杖或通道中斷連線，您可以產生新權杖並將其傳送到遠端裝置，以重新連線至通道。



若要使用主控台中開啟通道

若要開放新的通道，您可以使用 OpenTunnel API 操作。您也可以使用 API 指定其他組態，例如通道持續時間和目的地組態。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
```

執行此命令會建立新的通道，並提供您來源和目的地存取權杖。

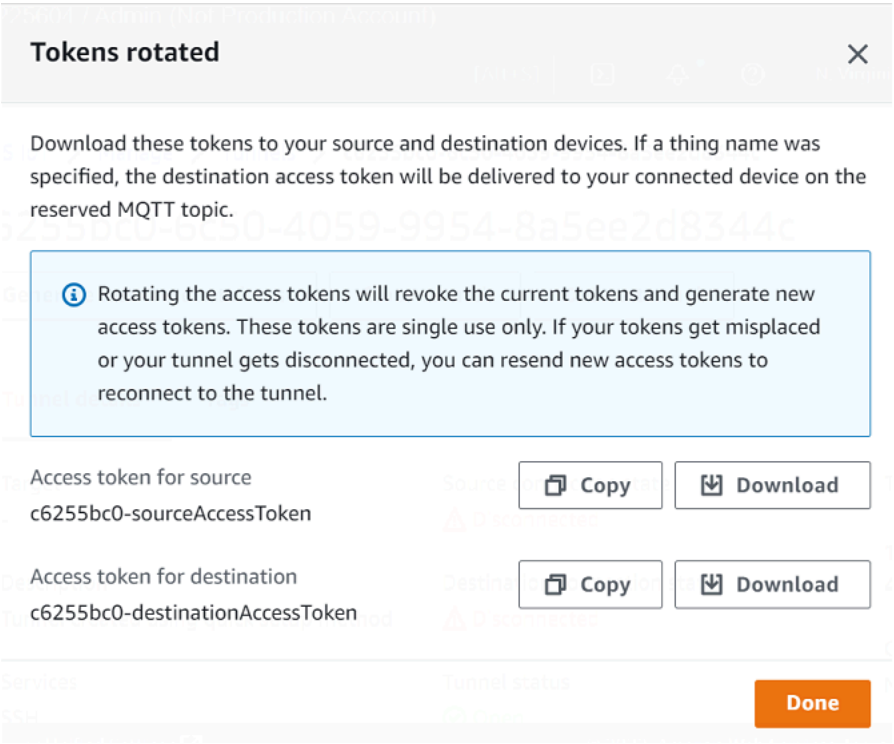
```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

重新傳送通道存取權杖

建立通道時取得的權杖，只能用來連接通道一次。如果您放錯存取權杖或通道中斷連接，您可以使用 MQTT 將新的存取權杖重新傳送至遠端裝置，無需額外費用。 AWS IoT 安全通道會撤銷目前的權杖，並傳回新的存取權杖以重新連線至通道。

從主控台輪換權杖

1. 前往 AWS IoT 主控台的通道中樞，然後選擇您建立的通道。
2. 在通道詳細資訊頁面中，選擇 Generate new access tokens (產生新存取權杖)，然後選擇 Next (下一步)。
3. 請為您的通道下載新的存取權杖，然後選擇 Done (完成)。這些權杖只能使用一次。如果誤置這些權杖或通道連線中斷，您可以重新傳送新的存取權杖。



使用 API 輪換存取權杖

若要輪換通道存取權杖，您可以使用 RotteTunnelAccesToken API 操作來撤銷目前的權杖，並傳回新的存取權杖以重新連線至通道。例如，下列指令會輪換目的地裝置的存取權杖：*RemoteThing1*。

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

執行此命令會產生新的存取權杖，如下所示。如果裝置代理程式設定正確，會使用 MQTT 將權杖傳遞到裝置以連接通道。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

如需說明如何及何時輪換存取權杖的範例，請參閱 透過輪換用戶端存取字符解決 AWS IoT 安全通道連線問題。

配置並啟動本機代理

若要連線至遠端裝置，請在筆記型電腦上開啟終端，然後設定並啟動本機代理。本機代理會透過 WebSocket 安全連線，使用安全通道來傳輸由來源裝置上執行的應用程式所傳送的資料。您可以從 GitHub下載本機代理原始碼。

設定完成本機代理後，請複製來源用戶端存取權杖，並使用它在來源模式下啟動本機代理。以下展示使用範例命令來啟動本機代理的步驟。在下列命令中，本機代理會設定為在連接埠 5555 上接聽新連線。在此命令中：

- `-r` 會指定 AWS 區域，其必須與您的通道建立所在的區域相同。
- `-s` 指定代理應該連線的連接埠。
- `-t` 指定用戶端權杖文字。

```
./localproxy -r us-east-1 -s 5555 -t source-client-access-token
```

若執行此命令，系統會以來源模式啟動本機代理。如果在執行命令後收到下列錯誤，請設定 CA 路徑。如需詳細資訊，請參閱 GitHub 上的安全通道本機代理。

```
Could not perform SSL handshake with proxy server: certificate verify failed
```

下列會顯示以 source 模式執行本機代理的範例輸出。

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

啟動 SSH 工作階段

開啟另一個終端機並使用以下命令，透過連線至連接埠 5555 上的本機代理來啟動新的 SSH 工作階段。

```
ssh username@localhost -p 5555
```

系統可能會提示您輸入 SSH 工作階段的密碼。當您完成了 SSH 工作階段時，請輸入 **exit** 以關閉工作階段。

清除

- 關閉通道

  我們建議您在使用完通道後關閉通道。如果開放時間超過指定的通道持續時間，通道也可能會關閉。通道一旦關閉就不能重新開放。您仍然可以開啟已關閉的通道，然後選擇 Duplicate tunnel (複製通道) 來複製通道。指定您要使用的通道持續時間，然後建立新通道。

  - 若要從 AWS IoT 主控台關閉個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要關閉的通道，然後選擇Close tunnel (關閉通道)。
  - 若要使用 AWS IoT API 參考 API 關閉個別通道或多個通道，請使用 CloseTunnel API。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 刪除通道

  您可以從 永久刪除通道 AWS 帳戶。

  > **Warning**
  >
  > 刪除動作為永久性動作，且無法還原。

  - 若要從 AWS IoT 主控台刪除個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要刪除的通道，然後選擇Delete tunnel (刪除通道)。
  - 若要使用 AWS IoT API 參考 API 刪除個別通道或多個通道，請使用 CloseTunnel API。使用 API 時，請將 `delete` 旗標設定為 `true`。

    ```
    aws iotsecuretunneling close-tunnel \
    ```

```
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

開啟遠端裝置的通道並使用瀏覽器型 SSH

從 AWS IoT 主控台，您可以從 Tunnels 中樞或您所建立 IoT 物件的詳細資訊頁面建立通道。當您從 Tunnel hub (通道中樞)建立通道，可以指定要使用快速設定或手動設定來建立通道。如需教學課程範例，請參閱開啟通道並啟動遠端裝置的 SSH 工作階段。

當您從 AWS IoT 主控台的物件詳細資訊頁面建立通道時，您也可以指定要為該物件建立新通道或開啟現有的通道，如本教學課程所示。如果您選擇現有通道，則可以存取您為此裝置建立的最新開放通道。然後，您可以使用終端中的命令行介面 SSH 進入設備。

先決條件

- 遠端裝置所在的防火牆必須允許連接埠 443 上的傳出流量。您建立的通道將使用此通訊埠連線至遠端裝置。
- 您已在 AWS IoT 登錄檔中建立 IoT 物件 （ 例如 RemoteDevice1)。此物件對應於您遠端裝置在雲端中的表示法。如需詳細資訊，請參閱在 AWS IoT 登錄檔中註冊裝置。
- 您有 IoT 裝置代理程式 （ 請參閱 IoT Agent Snippet) 在連線至 AWS IoT 裝置閘道的遠端裝置上執行，並使用 MQTT 主題訂閱進行設定。如需詳細資訊，請參閱將裝置連線至 AWS IoT 裝置閘道。
- 您必須具有一個在遠端裝置上執行的 SSH 常駐程式。

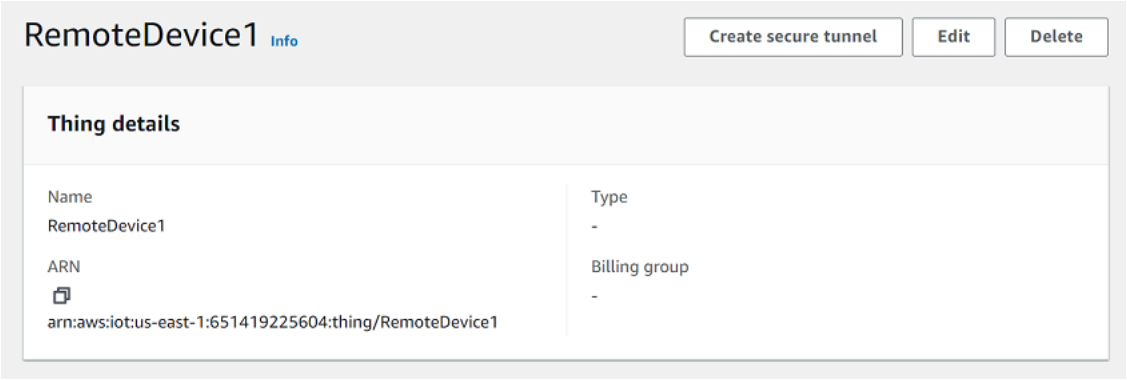
為遠端裝置開啟新通道

假設您要開啟通道進入遠端裝置：RemoteDevice1。首先，在 AWS IoT 登錄檔中建立名為 RemoteDevice1 的IoT 物件。然後，您可以使用 AWS 管理主控台、 AWS IoT API 參考 API 或 建立通道 AWS CLI。

如果您在鍵例通道時設定目的地，安全通道服務會透過 MQTT 和保留的 MQTT 主題 ($aws/things/RemoteDeviceA/tunnels/notify)，將目的地用戶端存取權杖傳遞給遠端裝置。如需詳細資訊，請參閱AWS IoT 主控台中的通道建立方法。

若要從主控台建立遠端裝置的通道

1. 選擇物件 RemoteDevice1 以檢視其詳細資訊，然後選擇 Create secure tunnel (建立安全通道)。



2. 選擇要建立新通道或是開啟現有通道。若要建立新通道，請選擇 Create new tunnel (建立新通道)。然後，您可以選擇要用快速設定或手動設定方法來建立通道。如需詳細資訊，請參閱使用手動設定開啟通道並連線至遠端裝置及開啟通道並使用瀏覽器型 SSH 來存取遠端裝置。

若要使用 API 為遠端裝置建立通道

若要開放新的通道，您可以使用 OpenTunnel API 操作。以下程式碼顯示執行此命令的範例。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
    --cli-input-json file://input.json
```

下列下顯示 input.json 檔案的內容。您可以使用 destinationConfig 參數來指定目的地裝置的名稱 (例如 *RemoteDevice1*) 以及要用來存取目的地裝置的服務，例如 *SSH*。您亦可選擇指定其他參數，如通道描述和標籤。

input.json 的內容

```
{
   "description": "Tunnel to remote device1",
   "destinationConfig": {
      "services": [ "SSH" ],
      "thingName": "RemoteDevice1"
   }
}
```

執行此命令會建立新的通道，並提供您來源和目的地存取權杖。

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
```

```
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

開啟現有的通道並使用瀏覽器型 SSH

假設您已RemoteDevice1使用手動設定方法或使用 AWS IoT API 參考 API，為遠端裝置 建立通道。此時您即可開啟裝置的現有通道，並選擇 Quick setup (快速設定) 以使用瀏覽器型 SSH 功能。現有通道的組態無法編輯，因此您無法使用手動設定方法。

若要使用瀏覽器型 SSH 功能，您不必下載來源存取權杖或設定本機代理。Web 型本機代理將為您自動設定，以便您開始與遠端裝置進行互動。

若要使用快速設定方法和瀏覽器型 SSH

1. 前往您所建立物件 RemoteDevice1 的詳細資訊頁面，然後Create secure tunnel (建立安全通道)。
2. 選擇 Use existing tunnel (使用現有通道)，開啟您為遠端裝置所建立的最新開放通道。無法編輯通道組態，因此您無法對通道使用手動設定方法。若要使用快速設定方法，請選擇 Quick setup (快速設定)。
3. 繼續檢閱，確認通道組態詳細資訊，並建立通道。通道組態不可編輯。

   當您建立通道時，安全通道會使用 RotateTunnelAccessToken API 操作來撤銷原始存取權杖並產生新的存取權杖。如果您的遠端裝置使用 MQTT，這些權杖會自動傳遞至其所訂閱 MQTT 主題的遠端裝置。您也可以選擇手動將這些權杖下載到來源裝置。

建立通道之後，您可以使用瀏覽器型 SSH，透過上下文命令列介面，直接從主控台與遠端裝置互動。若要使用此命令行介面，請選擇您所建立物件的通道，然後在詳細資訊頁面中展開 Command-line interface (命令行介面) 區段。由於已為您配置本機代理，因此您可以開始輸入命令以迅速開始存取遠端裝置並與其進行互動RemoteDevice1。

如需進一步了解快速設定方法和如何使用瀏覽器型 SSH，請參閱 開啟通道並使用瀏覽器型 SSH 來存取遠端裝置。

清除

- 關閉通道

我們建議您在使用完通道後關閉通道。如果開放時間超過指定的通道持續時間，通道也可能會關閉。通道一旦關閉就不能重新開放。您仍然可以開啟已關閉的通道，然後選擇 Duplicate tunnel (複製通道) 來複製通道。指定您要使用的通道持續時間，然後建立新通道。

- 若要從 AWS IoT 主控台關閉個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要關閉的通道，然後選擇Close tunnel (關閉通道)。
- 若要使用 AWS IoT API 參考 API 關閉個別通道或多個通道，請使用 CloseTunnel API。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- 刪除通道

  您可以從 永久刪除通道 AWS 帳戶。

  > **Warning**
  >
  > 刪除動作為永久性動作，且無法還原。

  - 若要從 AWS IoT 主控台刪除個別通道或多個通道，請移至 Tunnels hub (通道中樞)，選擇您要刪除的通道，然後選擇Delete tunnel (刪除通道)。
  - 若要使用 AWS IoT API 參考 API 刪除個別通道或多個通道，請使用 CloseTunnel API。使用 API 時，請將 `delete` 旗標設定為 `true`。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

本機代理

本機代理會透過 WebSocket 安全連線，使用安全通道來傳輸由來源裝置上執行的應用程式所傳送的資料。您可以從 GitHub下載本機代理原始碼。

本機代理可以在兩種模式下執行：`source` 或 `destination`。在來源模式中，本機代理會在與啟動 TCP 連線的用戶端應用程式相同的裝置或網路上執行。在目的地模式中，本機代理會與目的地應用程式一起在遠端裝置上執行。單一通道可以使用通道多工，一次最多支援三個資料串流。對於每個資料串

流，安全通道會使用多個 TCP 連線，進而降低逾時的可能性。如需詳細資訊，請參閱多工處理資料串流，並在安全通道中使用同步 TCP 連線。

如何使用本機代理

您可以在來源裝置和目的地裝置上執行本機代理，以將資料傳輸至安全通道端點。若您的裝置位於使用 Web 代理的網路中，Web 代理可在將連線轉送至網際網路之前加以攔截。在這種情況下，您需要將本機代理設定為使用 Web 代理。如需詳細資訊，請參閱為使用 Web 代理的裝置配置本機代理。

本機代理工作流程

下列步驟顯示了本機代理如何在來源和目的地裝置上執行。

1. 將本機代理連線至安全通道

   本機代理必須先建立安全通道的連線。當您啟動本機代理時，請使用下列引數：

   - 指定通道開啟 AWS 區域 所在 的`-r`引數。
   - `-t` 引數傳遞從 `OpenTunnel` 傳回的來源或目標用戶端存取字符。

   > **Note**
   > 使用相同用戶端存取字符值的兩個本機代理不能同時連線。

2. 執行來源或目的地動作

   建立 WebSocket 連線之後，本機代理會執行來源模式或目標模式動作，取決於其組態。

   根據預設，如果發生任何輸入/輸出 (I/O) 錯誤，或 WebSocket 連線意外關閉，本機代理會嘗試重新連線至安全通道。這會導致 TCP 連線關閉。如果發生任何 TCP 通訊端錯誤，本機代理會透過通道傳送訊息，通知對方關閉其 TCP 連線。根據預設，本機代理一律使用 SSL 通訊。

3. 停用本機代理

   使用通道之後，可以安全地停止本機代理程序。建議您呼叫 `CloseTunnel` 來明確地關閉通道。呼叫 後，作用中通道用戶端可能不會立即關閉`CloseTunnel`。

如需如何使用 AWS 管理主控台 開啟通道並啟動 SSH 工作階段的詳細資訊，請參閱 開啟通道並啟動遠端裝置的 SSH 工作階段。

本機代理最佳實務

執行本機代理時，請遵循下列最佳實務：

- 避免使用 `-t` 本機代理引數傳入存取字符。建議您使用 `AWSIOT_TUNNEL_ACCESS_TOKEN` 環境變數來設定本機代理的存取字符。
- 在作業系統或環境中以最低權限執行本機代理可執行檔。
  - 避免在 Windows 上以系統管理員身分執行本機代理。
  - 避免在 Linux 和 macOS 上以 root 身分執行本機代理。
- 考慮在個別的主機、容器、沙箱，chroot jail 或虛擬化環境上執行本機代理。
- 使用相關的安全旗標建置本地代理，具體取決於您的工具鏈。
- 在具有多個網路界面的裝置上，使用 `-b` 引數將 TCP 通訊端繫結至用來與目標應用程式通訊的網路界面。

命令和輸出範例

下列會顯示您執行的命令範例和對應的輸出。此範例顯示如何在 `source` 和 `destination` 模式下設定本機 Proxy。本機代理程式會將 HTTPS 通訊協定升級至 WebSockets，以建立長期連線，然後開始透過連線將資料傳輸至安全通道裝置端點。

在您執行這些命令之前：

您必須先開啟通道並取得來源和目的地的用戶端存取字符。您亦須依照先前所述建置本機代理。於 GitHub 儲存庫中開啟本機 Proxy 來源程式碼，並遵循建置和安裝本機 Proxy 的指示。

> **Note**
>
> 用於範例中的下列命令會使用 `verbosity` 標記，說明執行本機代理後先前所述之不同步驟的概觀。建議您僅將此標記用於測試目的。

以來源模式執行本機代理

下列命令顯示如何在來源模式下執行本機代理。

Linux/macOS

在 Linux 或 macOS 中，請在終端機中執行下列命令，以配置並啟動您來源上的本機代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -s 5555 -v 5 -r us-west-2
```

其中：

- `-s` 是來源接聽連接埠，會以來源模式啟動本機代理。
- `-v` 是輸出的詳細程度，可以是 0 到 6 之間的值。
- `-r` 是開啟通道的端點區域。

如需有關這些參數的詳細資訊，請參閱使用命令列引數設定的選項。

Windows

在 Windows 中，您可將本機代理配置為類似於您處理 Linux 或 macOS 的方式，但定義環境變數的方式則與其他平台不同。在 cmd 視窗中執行下列命令，以配置並啟動您來源上的本機代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -s 5555 -v 5 -r us-west-2
```

其中：

- `-s` 是來源接聽連接埠，會以來源模式啟動本機代理。
- `-v` 是輸出的詳細程度，可以是 0 到 6 之間的值。
- `-r` 是開啟通道的端點區域。

如需有關這些參數的詳細資訊，請參閱使用命令列引數設定的選項。

> **Note**
>
> 在來源模式下使用最新版本的本機代理時，您必須在`--destination-client-type V1`來源裝置上包含 AWS CLI 參數，才能回溯相容性。這適用於連線至下列任何目的地模式時：
>
> - AWS IoT 裝置用戶端
> - AWS IoT 安全通道元件或 AWS IoT Greengrass Version 2 安全通道元件
> - 2022 年之前寫入的任何 AWS IoT 安全通道示範程式碼
> - 本機代理的 1.X 版本

此參數可確保更新後的來源代理與較舊的目的地用戶端之間進行適當的通訊。如需本機代理版本的詳細資訊，請參閱 GitHub 上的AWS IoT 安全通道。

以下是在 source 模式下執行本機代理的範例輸出。

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

以目的地模式執行本機代理

下列命令顯示如何在目的地模式下執行本機代理。

Linux/macOS

在 Linux 或 macOS 中，請在終端機中執行下列命令，以配置並啟動您目的地上的本機代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -d 22 -v 5 -r us-west-2
```

其中：

- `-d` 是以目的地模式啟動本機代理的目的地應用程式。
- `-v` 是輸出的詳細程度，可以是 0 到 6 之間的值。
- `-r` 是開啟通道的端點區域。

如需有關這些參數的詳細資訊，請參閱使用命令列引數設定的選項。

Windows

在 Windows 中，您可將本機代理配置為類似於您處理 Linux 或 macOS 的方式，但定義環境變數的方式則與其他平台不同。在 cmd 視窗中執行下列命令，以配置並啟動您目的地上的本機代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -d 22 -v 5 -r us-west-2
```

其中：

- `-d` 是以目的地模式啟動本機代理的目的地應用程式。
- `-v` 是輸出的詳細程度，可以是 0 到 6 之間的值。
- `-r` 是開啟通道的端點區域。

如需有關這些參數的詳細資訊，請參閱使用命令列引數設定的選項。

> **Note**
>
> 在目的地模式下使用最新版本的本機代理時，您必須在`--destination-client-type V1`目的地裝置上包含 AWS CLI 參數，才能回溯相容性。這適用於連線至下列其中一個來源模式時：

- 從 AWS 主控台以瀏覽器為基礎的安全通道。
- 本機代理的 1.X 版本

此參數可確保更新的目的地代理與較舊的來源用戶端之間進行適當的通訊。如需本機代理版本的詳細資訊，請參閱 GitHub 上的AWS IoT 安全通道。

以下是在 destination 模式下執行本機代理的範例輸出。

```
...
...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

為使用 Web 代理的裝置配置本機代理

您可以使用 AWS IoT 裝置上的本機代理，與 AWS IoT 安全通道 APIs 通訊。本機代理會透過 WebSocket 安全連線，使用安全通道來傳輸裝置應用程式所傳送的資料。本機代理可在 `source` 或 `destination` 模式中運作。在 `source` 模式中，其會在啟動 TCP 連線的相同裝置或網路上執行。在 `destination` 模式中，本機代理會與目的地應用程式一起在遠端裝置上執行。如需詳細資訊，請參閱本機代理。

本機代理需要直接連線至網際網路，才能使用 AWS IoT 安全通道。若為與安全通道的長期 TCP 連線，本機代理會升級 HTTPS 請求以建立與安全通道裝置連線端點之一的 WebSockets 連線。

若您的裝置位於使用 Web 代理的網路中，Web 代理可在將連線轉送至網際網路之前加以攔截。如要建立與安全通道裝置連線端點的長期連線，請配置本機代理使用 Web 代理，如 WebSocket 規格中所述。

> **Note**
>
> AWS IoT 裝置用戶端 不支援使用 Web 代理的裝置。如要使用 Web 代理，您需要使用本機代理，並配置其與 Web 代理搭配使用，如下所述。

下列步驟顯示了本機代理如何與 Web 代理搭配使用。

1. 本機代理傳送 HTTP `CONNECT` 請求至 Web 代理，其包含安全通道服務的遠端地址及 Web 代理身分驗證資訊。
2. Web 代理隨後會建立遠端安全通道端點的長期連線。
3. TCP 連線已建立，本機代理現在可以來源和目的地模式運作，進行資料傳輸。

如要完成此程序，請執行下列步驟：

- 建置本機代理
- 配置您的 Web 代理
- 配置並啟動本機代理

建置本機代理

於 GitHub 儲存庫中開啟本機代理來源程式碼，並遵循建置和安裝本地代理的說明。

配置您的 Web 代理

本機代理依賴 HTTP/1.1 規格所說明的 HTTP 通道機制。如要符合規格，您的 Web 代理必須允許裝置可以使用 CONNECT 方法。

您配置 Web 代理的方式依您使用的 Web 代理及 Web 代理版本而定。如要確保您正確地配置 Web 代理，請查看您的 Web 代理文件。

如要配置您的 Web 代理，請先識別您的 Web 代理 URL，並確認您的 Web 代理是否支援 HTTP 通道。當您配置並啟動本機代理時，Web 代理 URL 將於稍後使用。

1. 識別您的 Web 代理 URL

   您的 Web 代理 URL 將會採用下列格式。

   ```
   protocol://web_proxy_host_domain:web_proxy_port
   ```

   AWS IoT 安全通道僅支援 Web 代理的基本身分驗證。如要使用基本身分驗證，您必須指定 **username** 和 **password** 以做為 Web 代理 URL 的一部分。Web 代理 URL 將會採用下列格式。

   ```
   protocol://username:password@web_proxy_host_domain:web_proxy_port
   ```

   - *####*可為 http 或 https。建議您使用 https。
   - *web_proxy_host_domain* 是您 Web 代理的 IP 地址，或可解析 Web 代理 IP 地址的 DNS 名稱。
   - *web_proxy_port* 是 Web 代理接聽的連接埠。
   - web 代理會使用此 **username** 和 **password** 來驗證請求。
2. 測試您的 Web 代理 URL

   如要確認您的 Web 代理是否支援 TCP 通道，請使用 curl 命令並確保您取得 2xx 或 3xx 回應。

   例如，若您的 Web 代理 URL 是 https://server.com:1235，請使用具 curl 命令的 proxy-insecure 旗標，因為 Web 代理可能會依賴自簽憑證。

   ```
   export HTTPS_PROXY=https://server.com:1235
   curl -I https://aws.amazon.com --proxy-insecure
   ```

   若您的 Web 代理 URL 具有 http 連接埠 (例如，http://server.com:1234)，則您無須使用 proxy-insecure 旗標。

```
export HTTPS_PROXY=http://server.com:1234
curl -I https://aws.amazon.com
```

配置並啟動本機代理

如要將本機代理配置為使用 Web 代理，您必須以 Web 代理使用的 DNS 網域名稱或 IP 地址和連接埠號來配置 HTTPS_PROXY 環境環境變數。

在您配置本機代理之後，您可以使用本機代理，如讀我檔案文件中所述。

> **Note**
>
> 您的環境變數宣告區分大小寫。我們建議您使用全部大寫或全部小寫字母來定義每個變數一次。下列範例顯示環境變數以全部大寫字母進行宣告。若同時使用大寫和小寫字母指定相同的變數，則使用小寫字母指定的變數優先。

下列命令顯示如何將執行於目的地上的本機代理配置為使用 Web 代理並啟動本機代理。

- AWSIOT_TUNNEL_ACCESS_TOKEN：此變數會保存目的地的用戶端存取字符 (CAT)。
- HTTPS_PROXY：此變數會保留 Web 代理 URL 或 IP 地址，以配置本機代理。

下列範例中顯示的命令取決於您使用的作業系統，及 Web 代理是否在 HTTP 或 HTTPS 連接埠上進行接聽。

在 HTTP 連接埠上接聽 Web 代理

若您的 Web 代理正在接聽 HTTP 連接埠，您可提供 HTTPS_PROXY 變數的 Web 代理 URL 或 IP 地址。

Linux/macOS

在 Linux 或 macOS 中，請在終端機中執行下列命令，以配置並啟動您目的地上的本機代理，以使用接聽 HTTP 連接埠的 Web 代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
```

```
./localproxy -r us-east-1 -d 22
```

若您必須使用代理進行驗證，則必須指定 **username** 和 **password** 以作為 HTTPS_PROXY 變數的一部分。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

Windows

在 Windows 中，您可將本機代理配置為類似於您處理 Linux 或 macOS 的方式，但定義環境變數的方式則與其他平台不同。在 cmd 視窗中執行下列命令，以配置並啟動您目的地上的本機代理，以使用接聽 HTTP 連接埠的 Web 代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22
```

若您必須使用代理進行驗證，則必須指定 **username** 和 **password** 以作為 HTTPS_PROXY 變數的一部分。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22
```

在 HTTPS 連接埠上接聽 Web 代理

若您的 Web 代理正在接聽 HTTPS 連接埠，請執行下列命令。

> **Note**
>
> 若您使用 Web 代理的自我簽署憑證，或是您在沒有原生 OpenSSL 支援和預設組態的作業系統上執行本機代理，則必須依照 GitHub 儲存庫中的憑證設定一節中的說明，設定您的 Web 代理憑證。

下列命令與您為 HTTP 代理配置 Web 代理的方式類似，不同之處在於您還需要指定之前所述所安裝憑證檔案的路徑。

Linux/macOS

在 Linux 或 macOS 中，請在終端機中執行下列命令，以配置執行於您目的地上的本機代理，以使用接聽 HTTPS 連接埠的 Web 代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

若您必須使用代理進行驗證，則必須指定 **username** 和 **password** 以作為 HTTPS_PROXY 變數的一部分。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

Windows

在 Windows 中，執行 cmd 視窗中的下列命令，可配置並啟動執行於目的地上的本機代理，以使用接聽 HTTP 連接埠的 Web 代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

若您必須使用代理進行驗證，則必須指定 **username** 和 **password** 以作為 HTTPS_PROXY 變數的一部分。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

命令和輸出範例

下列會顯示您執行於 Linux 作業系統上的命令範例，及相對應的輸出。該範例顯示接聽 HTTP 連接埠上的 Web 代理，及如何配置本機代理以 `source` 和 `destination` 兩種模式來使用 Web 代理。在執行這些命令之前，您必須先開啟通道並取得來源和目的地的用戶端存取字符。您還必須依照先前所述建置本機代理並配置 Web 代理。

此處概述您啟動本機代理後的步驟。本機代理：

- 識別 Web 代理 URL，以便其可使用 URL 連線至代理伺服器。
- 建立與 Web 代理的 TCP 連線。
- 將 HTTP CONNECT 請求傳送至 Web 代理，並等待 HTTP/1.1 200 回應，表示已建立連線。
- 將 HTTPS 通訊協定升級為 WebSockets 以建立一個長期的連接。
- 透過與安全通道裝置端點的連接，開始傳輸資料。

> **Note**
>
> 用於範例中的下列命令會使用 verbosity 標記，說明執行本機代理後先前所述之不同步驟的概觀。建議您僅將此標記用於測試目的。

以來源模式執行本機代理

下列命令顯示如何以來源模式執行本機代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -s 5555 -v 5 -r us-west-2
```

下列會顯示以 source 模式執行本機代理的範例輸出。

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.11
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
```

```
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 0a109afffee745f5-00001341-000b8138-cc6c878d80e8adb0-f186064b
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

以目的地模式執行本機代理

下列命令顯示如何以目的地模式執行本機代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -d 22 -v 5 -r us-west-2
```

下列會顯示以 destination 模式執行本機代理的範例輸出。

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.1
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
```

```
Successfully completed SSL handshake with proxy server
Web socket session ID: 06717bfffed3fd05-00001355-000b8315-da3109a85da804dd-24c3d10d
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

多工處理資料串流，並在安全通道中使用同步 TCP 連線

您可使用安全通道多工功能，在每個通道中使用多個資料串流。透過多工處理，您可以使用多個資料串流對裝置進行疑難排解。您還可透過消除建置、部署及啟動多個本機裝置或為相同裝置開啟多個通道的需求來減少作業負載。例如，對於需要傳送多個 HTTP 和 SSH 資料串流的 Web 瀏覽器，可使用多工處理。

對於每個資料串流， AWS IoT 安全通道支援同時 TCP 連線。使用同時連線可減少用戶端發出多個請求時逾時的可能性。例如，它可以減少遠端存取目標裝置本機的 Web 伺服器時的載入時間。

以下各節將詳細說明多工處理和使用同時 TCP 連線，以及它們的不同使用案例。

主題

- 多工處理安全通道中的多個資料串流
- 在安全通道中使用 TCP 連線

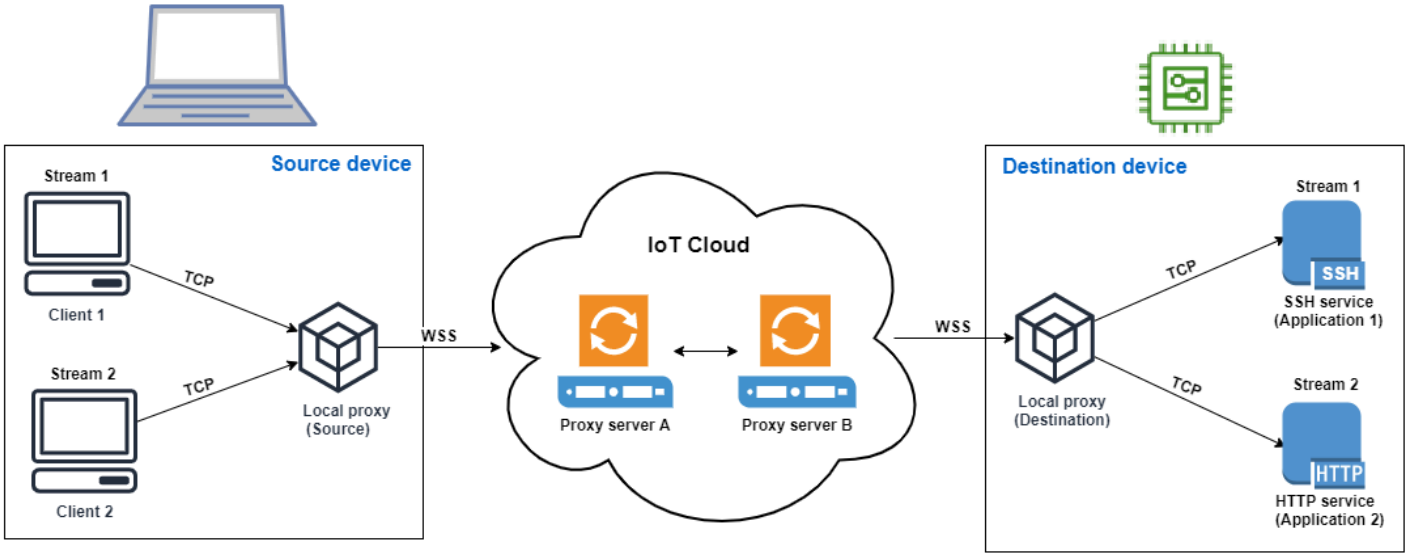
多工處理安全通道中的多個資料串流

對於使用多個連接或連接埠的裝置，您可以使用多工功能。當您需要與遠端裝置進行多個連線以疑難排解任何問題時，也可以使用多工處理。例如，可使用於需要傳送多個 HTTP 和 SSH 資料串流的 Web 瀏覽器。來自兩個串流的應用程式資料會透過多工通道同時傳輸。

範例使用案例

例如，您可能需要連接至裝置上的 Web 應用程式以更改某些連網參數，同時透過終端機發出 shell 命令，驗證裝置是否在使用新的連網參數時仍然正常運作。在此案例中，您可能需要透過 HTTP 和 SSH

連線到裝置，並傳輸兩個平行資料串流，以同時存取 Web 應用程式和終端機。利用多工功能，這兩個獨立串流可在同一時間透過同一個通道進行傳輸。



如何設定多工通道

下列程序將逐步引導您如何使用需要連線至多個連接埠的應用程式，來設定多工通道，以便對裝置進行疑難排解。您會設定一個具有兩個多工串流的通道：一個 HTTP 串流和一個 SSH 串流。

1. (選擇性) 建立組態檔

   您可以選擇性地使用組態檔來設定來源和目的地裝置。如果您的連接埠映射可能經常變更，請使用組態檔。如果您喜歡使用 CLI 指定連接埠映射，或無需在指定的接聽連接埠上啟動本機 Proxy，則可略過此步驟。如需有關如何使用組態檔的詳細資訊，請參閱 GitHub 中透過 —config 設定的選項。

   1. 在來源裝置上，在要執行本機 Proxy 的資料夾中，建立名為 `Config` 的組態資料夾。在此資料夾中建立一個名為 `SSHSource.ini` 的檔案，其內容如下：

      ```
      HTTP1 = 5555
      SSH1 = 3333
      ```

   2. 在目的地裝置上，在要執行本機 Proxy 的資料夾中，建立名為 `Config` 的組態資料夾。在此資料夾中建立一個名為 `SSHDestination.ini` 的檔案，其內容如下：

      ```
      HTTP1 = 80
      SSH1 = 22
      ```

2. 開啟通道

使用 OpenTunnel API 操作或 open-tunnel CLI 命令開啟通道。透過指定 SSH1和 HTTP1做為服務，以及對應至遠端裝置的 AWS IoT 物件名稱來設定目的地。您的 SSH 和 HTTP 應用程式正在此遠端裝置上執行。您必須已在 AWS IoT 登錄檔中建立 IoT 物件。如需詳細資訊，請參閱使用登錄檔管理物件。

```
aws iotsecuretunneling open-tunnel \
 --destination-config thingName=RemoteDevice1,services=HTTP1,SSH1
```

執行此命令會產生您將用來執行本機 Proxy 的來源和目的地存取字符。

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

3. 配置並啟動本機代理

在您執行本機代理之前，請先設定 AWS IoT 裝置用戶端，或從 GitHub 下載本機代理原始程式碼，然後針對您選擇的平台建置它。然後，您可以啟動目的地和來源本機 Proxy 以連接到安全通道。如需設定和使用本機 Proxy 的詳細資訊，請參閱 如何使用本機代理。

> **Note**
>
> 在來源裝置上，如果您不使用任何組態檔或使用 CLI 指定連接埠映射，您仍然可以使用相同的命令來執行本機 Proxy。本機 Proxy 將會自動選取可用的連接埠供您使用並管理映射。

Start local proxy using configuration files

執行下列命令，以使用組態檔在來源和目的地模式下執行本機 Proxy。

```
// ----------------- Start the destination local proxy ----------------------
./localproxy -r us-east-1 -m dst -t destination_client_access_token
```

```
// ----------------- Start the source local proxy ----------------------------
// You also run the same command below if you want the local proxy to
// choose the mappings for you instead of using configuration files.
./localproxy -r us-east-1 -m src -t source_client_access_token
```

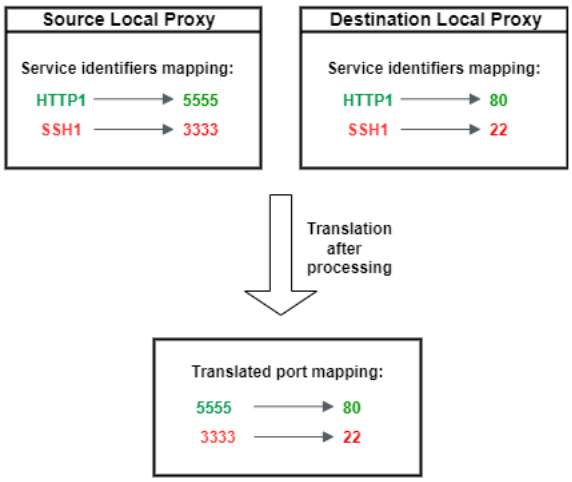
Start local proxy using CLI port mapping

使用 CLI 明確指定連接埠映射，以執行下列命令，在來源和目的地模式中執行本機 Proxy。

```
// ----------------- Start the destination local proxy
 ----------------------------------
./localproxy -r us-east-1 -d HTTP1=80,SSH1=22 -t destination_client_access_token

// ----------------- Start the source local proxy
 ---------------------------------------
./localproxy -r us-east-1 -s HTTP1=5555,SSH1=33 -t source_client_access_token
```

來自 SSH 和 HTTP 連線的應用程式資料現在可以透過多工通道同時進行傳輸。從下方對應圖中可看出，服務識別符的作用是一種可讀取格式，用來轉換來源和目的地裝置之間的連接埠映射。透過此組態，安全通道會將來源裝置上連接埠 *5555* 的任何傳入 HTTP 流量轉送至目的地裝置上的連接埠 *80*，以及從連接埠 *3333* 傳入的 SSH 流量轉送至目的地裝置上的連接埠 *22*。



在安全通道中使用 TCP 連線

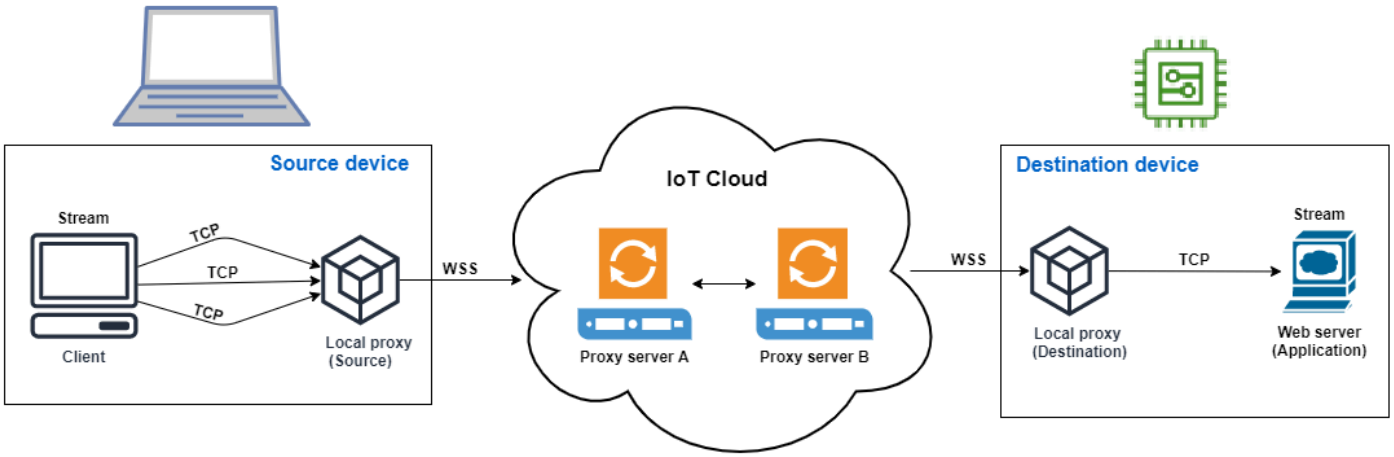
AWS IoT 安全通道為每個資料串流同時支援多個 TCP 連線。當您需要同時連線至遠端裝置時，可以使用此功能。如果有多個請求來自用戶端，使用同時 TCP 連線可以降低逾時的可能性。例如，當存取具有執行多個元件的 Web 伺服器時，同時 TCP 連線可減少載入網站所需的時間。

> **Note**
>
> 同時 TCP 連線的頻寬限制為每秒 800 KB AWS 帳戶。 AWS IoT 安全通道可根據傳入請求的數量為您設定此限制。

範例使用案例

假設您需要遠端存取位於目的地裝置本機上，且其上有多個元件運行的 Web 服務器。若使用單一 TCP 連線，嘗試存取 Web 伺服器時，循序載入可能會增加載入網站上資源所需的時間。同時 TCP 連線可透過滿足網站的資源需求來減少加載時間，從而縮短存取時間。下圖顯示對於通往遠端裝置上執行之 Web 伺服器應用程式的資料串流如何支援同時 TCP 連線。

> **Note**
>
> 如果您想要使用通道存取遠端裝置上執行的多個應用程式，您可以使用通道多工處理。如需詳細資訊，請參閱多工處理安全通道中的多個資料串流。



如何使用同時 TCP 連線

下列程序將逐步引導您如何使用同時 TCP 連線來存取遠端裝置上的 Web 瀏覽器。當用戶端有多個請求時， AWS IoT 安全通道會自動設定同時 TCP 連線來處理請求，進而縮短載入時間。

1. 開啟通道

   使用 OpenTunnel API 操作或 open-tunnel CLI 命令開啟通道。將 HTTP 指定為與遠端裝置對應的服務和 AWS IoT 物件名稱，以設定目的地。您的 Web 伺服器應用程式正在此遠端裝置上執行。您必須已在 AWS IoT 登錄檔中建立 IoT 物件。如需詳細資訊，請參閱使用登錄檔管理物件。

   ```
   aws iotsecuretunneling open-tunnel \
    --destination-config thingName=RemoteDevice1,services=HTTP
   ```

   執行此命令會產生您將用來執行本機 Proxy 的來源和目的地存取字符。

   ```
   {
    "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
    "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
   afa28b00cecd",
    "sourceAccessToken": source_client_access_token,
    "destinationAccessToken": destination_client_access_token
   }
   ```

2. 配置並啟動本機代理

   您已從 GitHub 下載了本機代理原始碼，並為您選擇的平台進行建置。然後，您可以啟動目的地和來源本機 Proxy，以連線到安全通道，並開始使用遠端 Web 伺服器應用程式。

   > **Note**
   >
   > 若要讓 AWS IoT 安全通道使用同時 TCP 連線，您必須升級至最新版本的本機代理。如果您使用 AWS IoT 裝置用戶端設定本機 Proxy，則無法使用此功能。

   ```
   // Start the destination local proxy
   ./localproxy -r us-east-1 -d HTTP=80 -t destination_client_access_token

   // Start the source local proxy
   ./localproxy -r us-east-1 -s HTTP=5555 -t source_client_access_token
   ```

   如需設定和使用本機 Proxy 的詳細資訊，請參閱 如何使用本機代理。

您現在可以使用通道存取 Web 伺服器應用程式。當用戶端有多個請求時， AWS IoT 安全通道會自動設定和處理同時 TCP 連線。

設定遠端裝置並使用 IoT 代理程式

IoT 代理程式用來接收包含用戶端存取字符的 MQTT 訊息，並在遠端裝置上啟動本機代理。如果您想要安全通道使用 MQTT 交付用戶端存取字符，則必須在遠端裝置上安裝並執行 IoT 代理程式。IoT 代理程式必須訂閱下列保留的 IoT MQTT 主題：

> **Note**
>
> 如果您想要透過訂閱保留的 MQTT 主題以外的方法將目的地用戶端存取字符傳遞給遠端裝置，則可能需要目的地用戶端存取字符 (CAT) 接聽程式和本機 Proxy。CAT 接聽程式必須使用您選擇的用戶端存取字符傳遞機制，並且能夠以目的地模式啟動本機 Proxy。

IoT Agent Snippet

IoT 代理程式必須訂閱下列保留的 IoT MQTT 主題，才能接收 MQTT 訊息並啟動本機 Proxy：

`$aws/things/`*`thing-name`*`/tunnels/notify`

其中 `thing-name`是與遠端裝置相關聯的 AWS IoT 物件名稱。

以下是 MQTT 訊息酬載的範例：

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

在收到 MQTT 訊息之後，IoT 代理程式必須使用適當的參數，在遠端裝置上啟動本機代理。

下列 Java 程式碼示範如何使用 Java 程式庫中的 AWS IoT 裝置 SDK 和 ProcessBuilder，來建置簡單的 IoT 代理程式，以使用安全通道。

```
// Find the IoT device endpoint for your AWS ##
final String endpoint = iotClient.describeEndpoint(new
 DescribeEndpointRequest().withEndpointType("iot:Data-ATS")).getEndpointAddress();
```

```
// Instantiate the IoT Agent with your AWS credentials
final String thingName = "RemoteDeviceA";
final String tunnelNotificationTopic = String.format("$aws/things/%s/tunnels/notify",
 thingName);
final AWSIotMqttClient mqttClient = new AWSIotMqttClient(endpoint, thingName,
                 "your_aws_access_key", "your_aws_secret_key");

try {
    mqttClient.connect();
    final TunnelNotificationListener listener = new
 TunnelNotificationListener(tunnelNotificationTopic);
    mqttClient.subscribe(listener, true);
}
finally {
    mqttClient.disconnect();
}

private static class TunnelNotificationListener extends AWSIotTopic {
    public TunnelNotificationListener(String topic) {
        super(topic);
    }

    @Override
    public void onMessage(AWSIotMessage message) {
        try {
            // Deserialize the MQTT message
            final JSONObject json = new JSONObject(message.getStringPayload());

            final String accessToken = json.getString("clientAccessToken");
            final String region = json.getString("region");

            final String clientMode = json.getString("clientMode");
            if (!clientMode.equals("destination")) {
                throw new RuntimeException("Client mode " + clientMode + " in the MQTT
 message is not expected");
            }

            final JSONArray servicesArray = json.getJSONArray("services");
            if (servicesArray.length() > 1) {
                throw new RuntimeException("Services in the MQTT message has more than
 1 service");
            }
            final String service = servicesArray.get(0).toString();
```

```
            if (!service.equals("SSH")) {
                throw new RuntimeException("Service " + service + " is not supported");
            }

            // Start the destination local proxy in a separate process to connect to
 the SSH Daemon listening port 22
            final ProcessBuilder pb = new ProcessBuilder("localproxy",
                        "-t", accessToken,
                        "-r", region,
                        "-d", "localhost:22");
            pb.start();
        }
        catch (Exception e) {
            log.error("Failed to start the local proxy", e);
        }
    }
}
```

控制 Kibana 的存取

安全通道提供服務特有的動作、資源和條件內容金鑰，可用於 IAM 許可政策。

通道存取先決條件

- 了解如何使用 IAM 政策保護 AWS 資源。
- 了解如何建立和評估 IAM 條件。
- 了解如何使用 AWS 資源標籤保護資源。

通道存取政策

您必須使用以下政策授予許可，才能使用安全通道 API。如需 AWS IoT 安全性的詳細資訊，請參閱 的身分和存取管理 AWS IoT。

iot:OpenTunnel

`iot:OpenTunnel` 政策動作可授與委託人呼叫 OpenTunnel 的許可。

在 IAM 政策陳述式中的 `Resource` 元素：

- 指定萬用字元通道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

- 指定一個物件 ARN 來管理特定 IoT 物件的 OpenTunnel 許可：

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

例如，以下政策陳述式允許您開啟一個通道，前往名為 TestDevice 的 IoT 物件。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:OpenTunnel 政策動作支援下列條件金鑰：

- iot:ThingGroupArn
- iot:TunnelDestinationService
- aws:RequestTag/*tag-key*
- aws:SecureTransport
- aws:TagKeys

下列政策陳述式可讓您對事物開啟通道，前提是該事物屬於名稱開頭為 TestGroup 的事物群組，並且通道上配置的目的地服務為 SSH。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
```

```
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ]
        }
    }
}
```

您也可以使用資源標籤來控制開啟通道的許可。例如，如果標籤索引鍵 Owner 的值為 Admin 且未指定其他標籤，則下列政策陳述式允許開啟通道。如需使用標籤的詳細資訊，請參閱 標記您的 AWS IoT 資源。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/Owner": "Admin"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "Owner"
        }
    }
}
```

iot:RotateTunnelAccessToken

`iot:RotateTunnelAccessToken` 政策動作可授予委託人呼叫 RotateTunnelAccessToken 的許可。

在 IAM 政策陳述式中的 `Resource` 元素：

- 指定完全的合格通道 ARN：

  `arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`

  您也可以使用萬用字元通道 ARN：

  `arn:aws:iot:aws-region:aws-account-id:tunnel/*`

- 指定一個物件 ARN 來管理特定 IoT 物件的 RotateTunnelAccessToken 許可：

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

例如，以下政策陳述式允許您輪換通道的來源存取字符，或用於名為 TestDevice 的 IoT 物件之用戶端目的地存取字符。

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:RotateTunnelAccessToken 政策動作支援下列條件金鑰：

- iot:ThingGroupArn
- iot:TunnelDestinationService
- iot:ClientMode
- aws:SecureTransport

下列政策陳述式可讓您將目的地存取字符輪換至物件，前提是該事物屬於名稱開頭為 TestGroup 的物件群組，並且通道上配置的目的地服務為 SSH，用戶端狀態為 DESTINATION。

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
```

```
                "SSH"
            ],
            "iot:ClientMode": "DESTINATION"
        }
    }
}
```

iot:DescribeTunnel

`iot:DescribeTunnel` 政策動作可授與委託人呼叫 DescribeTunnel 的許可。

在 IAM 政策陳述式的 `Resource` 元素中指定完全合格的通道 ARN：

`arn:aws:iot:`*`aws-region`*`:` *`aws-account-id`*`:tunnel/`*`tunnel-id`*

您也可以使用萬用字元 ARN：

`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`

`iot:DescribeTunnel` 政策動作支援下列條件金鑰：

- `aws:ResourceTag/`*`tag-key`*
- `aws:SecureTransport`

如果使用值為 `Admin` 的金鑰 `Owner` 標記所請求的通道，下列政策陳述式可讓您呼叫 `DescribeTunnel`。

```
{
    "Effect": "Allow",
    "Action": "iot:DescribeTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/Owner": "Admin"
        }
    }
}
```

iot:ListTunnels

`iot:ListTunnels` 政策動作可授與委託人呼叫 ListTunnels 的許可。

在 IAM 政策陳述式中的 Resource 元素：

- 指定萬用字元通道 ARN：

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

- 指定一個物件 ARN 來管理所選 IoT 物件的 ListTunnels 許可：

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

iot:ListTunnels 政策動作支援條件金鑰 aws:SecureTransport。

以下政策陳述式允許您列出名為 TestDevice 之物件的通道。

```
{
    "Effect": "Allow",
    "Action": "iot:ListTunnels",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:ListTagsForResource

iot:ListTagsForResource 政策動作可授與委託人呼叫 ListTagsForResource 的許可。

在 IAM 政策陳述式的 Resource 元素中指定完全合格的通道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用萬用字元通道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:ListTagsForResource 政策動作支援條件金鑰 aws:SecureTransport。

iot:CloseTunnel

iot:CloseTunnel 政策動作可授與委託人呼叫 CloseTunnel 的許可。

在 IAM 政策陳述式的 Resource 元素中指定完全合格的通道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用萬用字元通道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:CloseTunnel 政策動作支援下列條件金鑰：

- iot:Delete
- aws:ResourceTag/*tag-key*
- aws:SecureTransport

如果請求的 Delete 參數為 false，並使用值為 QATeam 的金鑰 Owner 標記所請求的通道，下列政策陳述式可讓您呼叫 CloseTunnel。

```
{
    "Effect": "Allow",
    "Action": "iot:CloseTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "Bool": {
            "iot:Delete": "false"
        },
        "StringEquals": {
            "aws:ResourceTag/Owner": "QATeam"
        }
    }
}
```

iot:TagResource

iot:TagResource 政策動作可授與委託人呼叫 TagResource 的許可。

在 IAM 政策陳述式的 Resource 元素中指定完全合格的通道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用萬用字元通道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:TagResource 政策動作支援條件金鑰 aws:SecureTransport。

iot:UntagResource

iot:UntagResource 政策動作可授與委託人呼叫 UntagResource 的許可。

在 IAM 政策陳述式的 Resource 元素中指定完全合格的通道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用萬用字元通道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:UntagResource 政策動作支援條件金鑰 aws:SecureTransport。

透過輪換用戶端存取字符解決 AWS IoT 安全通道連線問題

當您使用 AWS IoT 安全通道時，即使通道已開啟，也可能會遇到連線問題。以下部分介紹了一些可能的問題，以及如何透過輪換用戶端存取字符解決這些問題。若要輪換用戶端存取字符 (CAT)，請使用 RotateTunnelAccessToken API 或 rotate-tunnel-access-token AWS CLI。根據在來源模式或目的地模式下使用用戶端時遇到錯誤，可以在來源模式或目的地模式下輪換 CAT，也可以同時輪換。

> **Note**
>
> - 如果您不確定是否需要在來源或目的地輪換 CAT，可以在使用 RotateTunnelAccessToken API 時將 ClientMode 設為 ALL，同時輪換來源和目的地的 CAT。
> - 輪換 CAT 不會延長通道持續時間。例如，假設通道持續時間為 12 小時，且通道已開放 4 小時。當您輪換存取字符，產生的新字符只能在剩下 8 小時內使用。

主題

- 無效的用戶端存取字符錯誤
- 用戶端字符不相符錯誤
- 遠端裝置連線問題

無效的用戶端存取字符錯誤

使用 AWS IoT 安全通道時，當您使用相同的用戶端存取字符 (CAT) 重新連線至相同的通道時，可能會遇到連線錯誤。在這種情況下，本機代理無法連線到安全通道代理伺服器。如果您使用來源模式的用戶端，可能會看到以下錯誤訊息：

```
Invalid access token: The access token was previously used and cannot be used again
```

出現這個錯誤的原因是用戶端存取字符 (CAT) 只能由本機代理使用一次，之後就無效。如果要解決這個錯誤，請使用 SOURCE 模式輪換用戶端存取字符，為來源產生新的 CAT。如需顯示如何輪換來源 CAT 的範例，請參閱 輪換來源 CAT 範例。

用戶端字符不相符錯誤

> **Note**
>
> 不推薦以用戶端字符重複使用 CAT。建議您使用 RotateTunnelAccessToken API 來輪換用戶端存取字符以重新連線到通道。

如果您使用的是用戶端字符，可以重複使用 CAT 重新連線到通道。要重複使用 CAT，必須在首次連線到安全通道時為用戶端字符提供 CAT。安全通道存放用戶端字符，因此對於使用相同字符的後續連線嘗試，也必須提供用戶端字符。如需有關使用用戶端字符的詳細資訊，請參閱在 GitHub 中的本地代理參考實作。

使用用戶端字符時，如果您使用來源模式的用戶端，可能會看到以下錯誤：

```
Invalid client token: The provided client token does not match the client token
    that was previously set.
```

出現這個錯誤的原因，是提供的用戶端字符與存取通道時隨 CAT 提供的用戶端字符不相符。若要解決這個錯誤，請使用 SOURCE 模式輪換 CAT，為來源產生新的 CAT。如下列範例所示：

輪換來源 CAT 範例

以下說明如何以 SOURCE 模式執行 RotateTunnelAccessToken API，為來源產生一個新的 CAT：

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --region <region> \
```

```
    --tunnel-id <tunnel-id> \
    --client-mode SOURCE
```

執行這個命令會產生新的來源存取字符，並回傳通道的 ARN。

```
{
    "sourceAccessToken": "<source-access-token>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/<tunnel-id>"
}
```

您現在可以使用新的來源字符以來源模式與本機代理連線。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=<source-access-token>
./localproxy -r <region> -s <port>
```

下列會顯示執行本機代理的範例輸出：

```
...

[info]    Starting proxy in source mode
...
[info]    Successfully established websocket connection with proxy server ...
[info]    Listening for new connection on port <port>
...
```

遠端裝置連線問題

使用 AWS IoT 安全通道時，即使通道已開啟，裝置也可能會意外中斷連線。若要識別裝置是否仍連接至通道，您可以使用 DescribeTunnel API 或 describe-tunnel AWS CLI。

裝置可能會因多種原因而中斷連線。要解決連線問題，如果裝置由於以下可能原因而中斷連線，可以在輪換目的地的 CAT：

- 目的地的 CAT 變為無效。
- 字符未透過安全通道保留的 MQTT 主題傳遞到裝置：

  `$aws/things/<thing-name>/tunnels/notify`

以下範例說明如何解決這個問題：

輪換目標 CAT 範例

考慮使用遠端裝置 *<RemoteThing1>*。要開放該物件的通道，可以使用以下命令：

```
aws iotsecuretunneling open-tunnel \
    --region <region> \
    --destination-config thingName=<RemoteThing1>,services=SSH
```

執行這個命令會產生來源和目的地的通道詳細資訊和 CAT。

```
{
    "sourceAccessToken": "<source-access-token>",
    "destinationAccessToken": "<destination-access-token>",
    "tunnelId": "<tunnel-id>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/tunnel-id"
}
```

但是，當您使用 DescribeTunnel API，輸出顯示裝置已中斷連線，如下圖所示：

```
aws iotsecuretunneling describe-tunnel \
    --tunnel-id <tunnel-id> \
    --region <region>
```

執行這個命令會顯示裝置仍未連線。

```
{
    "tunnel": {
        ...
        "destinationConnectionState": {
            "status": "DISCONNECTED"
        },
        ...
    }
}
```

若要解決這個錯誤，請以 DESTINATION 模式的用戶端和目的地的組態執行 RotateTunnelAccessToken API。執行這個命令會撤銷舊的存取字符，產生新字符，並將這個字符重新傳送到 MQTT 主題：

$aws/things/*<thing-name>*/tunnels/notify

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

執行這個命令會產生新的存取字符，如下所示。如果裝置代理程式設定正確，會將字符傳遞到裝置以連接通道。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

裝置佈建

AWS 提供多種不同的方法來佈建裝置，並在其上安裝唯一的用戶端憑證。本節描述每一種方式，以及如何選取最適合您 IoT 解決方案的方式。這些選項會在標題名稱為 Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core 的白皮書中詳細描述。

選取最適合您情況的選項

- 您可以在 IoT 裝置上安裝憑證，然後再交付這些憑證

  如果您可以在 IoT 裝置上安全地安裝唯一的用戶端憑證，然後再交付這些憑證，以供最終使用者使用，則您想要使用即時佈建 (JITP)或即時註冊 (JITR)。

  使用 JITP 和 JITR，用於簽署裝置憑證的憑證授權單位 (CA) 會向 註冊， AWS IoT 並在裝置第一次連線 AWS IoT 時由 識別。裝置會在其第一個連線 AWS IoT 上使用其佈建範本的詳細資訊在 中佈建。

  如需有關單一物件、JITP、JITR，以及大量佈建具有唯一憑證之裝置的詳細資訊，請參閱the section called “佈建具有裝置憑證的裝置”。
- 最終使用者或安裝程式可以使用應用程式，在其 IoT 裝置上安裝憑證

  如果您無法在 IoT 裝置上安全地安裝唯一的用戶端憑證，然後再將這些憑證交付給最終使用者，但最終使用者或安裝程式可以使用應用程式，來註冊裝置並安裝唯一的裝置憑證，則您想要使用透過信任的使用者佈建程序。

  使用信任的使用者 (例如具有已知帳戶的最終使用者或安裝程式) 可以簡化裝置製造程序。裝置沒有唯一的用戶端憑證，而是具有暫時憑證，讓裝置 AWS IoT 只連線到 5 分鐘。在這 5 分鐘的時間範圍期間，信任的使用者會取得連線時間更長的唯一用戶端憑證，並將其安裝在裝置上。宣告憑證的有限連線時間可將憑證洩露的風險降至最低。

  如需詳細資訊，請參閱 the section called “由信任的使用者佈建”。
- 最終使用者無法使用應用程式，在其 IoT 裝置上安裝憑證

  如果先前的選項都無法在您的 IoT 解決方案中運作，透過要求佈建程序是一個選項。透過此程序，您的 IoT 裝置會擁有由機群中其他裝置共用的宣告憑證。裝置第一次與宣告憑證連線時， 會使用其佈建範本 AWS IoT 註冊裝置，並發行裝置唯一的用戶端憑證，以供後續存取 AWS IoT。

此選項會在裝置連線時啟用裝置自動佈建 AWS IoT，但在遺失宣告憑證時可能會帶來更大的風險。如果宣告憑證洩露，您可以停用該憑證。停用宣告憑證可防止未來不得註冊具有該宣告憑證的所有裝置。不過，停用宣告憑證並不會封鎖已佈建的裝置。

如需詳細資訊，請參閱the section called “透過要求佈建”。

在 中佈建裝置 AWS IoT

當您使用 佈建裝置時 AWS IoT，您必須建立 資源，以便您的裝置和 AWS IoT 可以安全地通訊。可以建立其他資源來協助您管理裝置機群。可以在佈建過程中建立下列資源：

- IoT 物件。

  IoT 物件是 AWS IoT 裝置登錄檔中的項目。每個物件都有一個唯一名稱和一組屬性，並與實體裝置相關聯。物件可以使用物件類型來定義，也可以分組為物件群組。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。

  雖然不一定要建立物件，但建立物件可讓您依物件類型、物件群組和物件屬性搜尋裝置，更有效地管理您的裝置機群。如需詳細資訊，請參閱機群索引。

  > **Note**
  >
  > 若要索引物件的連線狀態資料，請佈建您的物件並加以設定，讓物件名稱與 Connect 請求中使用的用戶端 ID 相符。

- X.509 憑證。

  裝置使用 X.509 憑證來執行交互身分驗證 AWS IoT。您可以註冊現有的憑證，或讓 為您 AWS IoT 產生並註冊新的憑證。您可以將憑證附加至代表裝置的物件，將該憑證與裝置產生關聯。您也必須將憑證和關聯的私有金鑰複製到裝置。裝置在連線到 時出示憑證 AWS IoT。如需詳細資訊，請參閱身分驗證。

- IoT 政策。

  IoT 政策定義了裝置可在 AWS IoT中執行哪些操作。IoT 政符會附加至裝置憑證。當裝置向 出示憑證時 AWS IoT，會授予政策中指定的許可。如需詳細資訊，請參閱Authorization。每個裝置都需要憑證才能與 AWS IoT進行通訊。

AWS IoT 支援使用佈建範本的自動機群佈建。佈建範本說明佈建裝置 AWS IoT 所需的資源。範本包含的變數可讓您使用一個範本來佈建多個裝置。佈建裝置時，您可以使用字典或 map，為裝置特定的變數指定值。若要佈建另一個裝置，請在字典中指定新值。

無論您的裝置是否具有唯一憑證 (及其關聯的私有金鑰)，您都可以使用自動佈建。

機群佈建 API

機群佈建中使用的 API 有數種類別：

- 這些控制平面 API 可建立和管理機群佈建範本，以及設定受信任的使用者政策。
  - CreateProvisioningTemplate
  - CreateProvisioningTemplateVersion
  - DeleteProvisioningTemplate
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioningTemplate
  - DescribeProvisioningTemplateVersion
  - ListProvisioningTemplates
  - ListProvisioningTemplateVersions
  - UpdateProvisioningTemplate
- 受信任的使用者可以使用此控制平面功能來產生暫時上線要求。此暫時要求會在 Wi-Fi 組態或類似方法期間傳遞至裝置。
  - CreateProvisioningClaim
- 裝置在佈建過程中使用的 MQTT API，具有內嵌在裝置中的佈建宣告憑證，或是由受信任使用者傳遞給它的佈建要求。
  - the section called “CreateCertificateFromCsr”
  - the section called “CreateKeysAndCertificate”
  - the section called “RegisterThing”

使用機群佈建來佈建沒有裝置憑證的裝置

透過使用 AWS IoT 機群佈建， AWS IoT 可以在 AWS IoT 首次連線至 時產生裝置憑證和私有金鑰，並將其安全地交付至您的裝置。 AWS IoT 提供由 Amazon 根憑證授權單位 (CA) 簽署的用戶端憑證。

使用機群佈建的方式有兩種：

- 透過要求佈建
- 由信任的使用者佈建

透過要求佈建

裝置可以使用內嵌的佈建宣告憑證和私有金鑰 (這是特殊用途的憑證憑證) 來製造。如果這些憑證已向註冊 AWS IoT，服務可以將它們交換為裝置可用於一般操作的唯一裝置憑證。此程序包含以下步驟：

在您交付裝置之前

1. 呼叫 `CreateProvisioningTemplate` 來建立佈建範本。這個 API 傳回範本 ARN。如需詳細資訊，請參閱裝置佈建 MQTT API。

   您也可以在 AWS IoT 主控台中建立機群佈建範本。

   a. 從導覽窗格中，選擇連接許多裝置下拉式清單。然後，選擇連接許多裝置。

   b. 選擇建立佈建範本。

   c. 選擇最適合您安裝程序的佈建案例。然後選擇下一步。

   d. 完成範本工作流程。

2. 建立要用作佈建宣告憑證的憑證和關聯私有金鑰。

3. 向 註冊這些憑證， AWS IoT 並將限制憑證使用的 IoT 政策建立關聯。下列範例 IoT 政策會限制使用與此政策相關聯的憑證來佈建裝置。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
```

```
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/
*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/templateName/provision/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/
create/*",
    "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/templateName/provision/*"
            ]
        }
    ]
}
```

4. 授予 AWS IoT 服務許可，以在佈建裝置時建立或更新 IoT 資源，例如帳戶中的物件和憑證。透過將 AWSIoTThingsRegistration受管政策連接至信任 AWS IoT 服務主體的 IAM 角色（稱為佈建角色）來執行此操作。
5. 製造裝置，並安全地將佈建宣告憑證內嵌其中。

裝置現已準備就緒，可以傳送到將安裝的位置以供使用。

> **Important**
>
> 佈建要求私有金鑰應該始終加以保護，包括在裝置上。我們建議您使用 AWS IoT CloudWatch 指標和日誌來監控是否有濫用跡象。如果您偵測到誤用，請關閉佈建宣告憑證，使其無法用於裝置佈建。

初始化裝置以供使用

1. 裝置使用 AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端連線至 ，並使用安裝在裝置上的 AWS IoT 佈建宣告憑證來驗證 。

> **Note**
>
> 為了安全起見，由 CreateCertificateFromCsr 和 CreateKeysAndCertificate 傳回的 certificateOwnershipToken 會在一小時後過期。必須在 certificateOwnershipToken 過期之前呼叫 RegisterThing。如果由 CreateCertificateFromCsr 或者 CreateKeysAndCertificate 建立的憑證尚未啟用，並且在權杖過期時，尚未附加到政策或物件，憑證會遭到刪除。如果字符過期，裝置可以再次呼叫 CreateCertificateFromCsr 或 CreateKeysAndCertificate 來產生新憑證。

2. 裝置會使用其中一個選項取得永久憑證和私密金鑰。裝置將使用憑證和金鑰進行所有未來的身分驗證 AWS IoT。

   a. 呼叫 CreateKeysAndCertificate 以使用憑證授權單位建立新的 AWS 憑證和私有金鑰。

      或

   b. 呼叫 CreateCertificateFromCsr 以從保持私有金鑰安全的憑證簽署要求產生憑證。

3. 從裝置呼叫 RegisterThing，以向 AWS IoT 註冊裝置，並建立雲端資源。

   同時，機群佈建服務會使用佈建範本來定義並建立雲端資源，例如 IoT 物件。範本可以指定物件所屬的屬性與群組。必須先有物件群組，才能將新物件加入其中。

4. 在裝置上儲存永久憑證後，裝置必須中斷與佈建宣告憑證初始化的工作階段連線，然後使用永久憑證重新連線。

裝置現在已準備好與 正常通訊 AWS IoT。

由信任的使用者佈建

在許多情況下，當信任的使用者，例如最終使用者或安裝技術人員，使用行動應用程式在其部署位置設定裝置時，裝置會 AWS IoT 第一次連線到 。

> **Important**
>
> 您必須管理信任之使用者的存取權和許可，才能執行此程序。其中一種方法是為信任的使用者提供和維護帳戶，以驗證他們，並授予他們存取執行此程序所需的 AWS IoT 功能和 API 操作。

在您交付裝置之前

1. 呼叫 CreateProvisioningTemplate 以建立佈建範本，並傳回其 *templateArn* 和 *templateName*。
2. 建立受信任使用者用來啟動佈建程序的 IAM 角色。佈建範本只允許該使用者佈建裝置。例如：

```
{
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:provisioningtemplate/templateName"
    ]
}
```

3. 授予 AWS IoT 服務許可，以在佈建裝置時建立或更新 IoT 資源，例如帳戶中的物件和憑證。您可以透過將 AWSIoTThingsRegistration受管政策連接至信任 AWS IoT 服務主體的 IAM 角色（稱為佈建角色) 來執行此操作。
4. 提供識別信任使用者的方法，例如為他們提供可進行身分驗證的帳戶，並授權他們與註冊其裝置所需的 AWS API 操作互動。

初始化裝置以供使用

1. 信任的使用者會登入您的佈建行動應用程式或 Web 服務。
2. 行動應用程式或 Web 應用程式會使用 IAM 角色並呼叫 CreateProvisioningClaim，以從 AWS IoT取得暫時佈建宣告憑證。

> **Note**
>
> 為了安全起見，CreateProvisioningClaim 傳回的暫時佈建宣告憑證會在五分鐘後過期。在暫時佈建宣告憑證到期之前，下列步驟必須成功傳回有效的憑證。暫時佈建宣告憑證不會出現在您帳戶的憑證清單中。

3. 行動應用程式或 Web 應用程式會將暫時佈建宣告憑證以及任何必要的設定資訊 (例如 Wi-Fi 認證) 提供給裝置。
4. 裝置會使用暫時佈建宣告憑證，使用 AWS IoT 來連線至 AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端。

5. 裝置會在 AWS IoT 使用暫時佈建宣告憑證連線至 的五分鐘內，使用其中一個選項取得永久憑證和私有金鑰。裝置將使用憑證和金鑰，這些選項會傳回以供所有未來的身分驗證使用 AWS IoT。

   a. 呼叫 CreateKeysAndCertificate 以使用憑證授權單位建立新的 AWS 憑證和私有金鑰。

      或

   b. 呼叫 CreateCertificateFromCsr 以從保持私有金鑰安全的憑證簽署要求產生憑證。

   > **Note**
   >
   > 請記住， CreateKeysAndCertificate 或 CreateCertificateFromCsr 必須在 AWS IoT 使用暫時佈建宣告憑證連線至 的五分鐘內傳回有效的憑證。

6. 裝置會呼叫 RegisterThing向 註冊裝置 AWS IoT 並建立雲端資源。

   同時，機群佈建服務會使用佈建範本來定義並建立雲端資源，例如 IoT 物件。範本可以指定物件所屬的屬性與群組。必須先有物件群組，才能將新物件加入其中。

7. 在裝置上儲存永久憑證之後，裝置必須中斷與使用暫時佈建宣告憑證起始的工作階段的連線，然後使用永久憑證重新連線。

裝置現在已準備好與 正常通訊 AWS IoT。

搭配 CLI AWS 使用預先佈建掛鉤

下列程序會建立具有預先佈建掛接的佈建範本。這裡使用的 Lambda 函數是一個可修改的範例。

若要建立預先佈建掛接，並將其套用至佈建範本

1. 建立具有所定義輸入和輸出的 Lambda 函數。Lambda 函數是可高度自訂的。需有 `allowProvisioning` 和 `parameterOverrides` 才能建立預先佈建的掛接。如需建立 Lambda 函數的詳細資訊，請參閱搭配使用 AWS Lambda 與 AWS 命令列界面。

   以下是 Lambda 函數輸出的範例：

```
{
  "allowProvisioning": True,
  "parameterOverrides": {
    "incomingKey0": "incomingValue0",
    "incomingKey1": "incomingValue1"
```

```
  }
}
```

2. AWS IoT 使用資源型政策來呼叫 Lambda，因此您必須授予 AWS IoT 呼叫 Lambda 函數的許可。

> **Important**
>
> 請務必在連接到 Lambda 動作的政策的全域條件內容金鑰中包含 `source-arn` 或 `source-account`，以防止許可操作。如需此項目的詳細資訊，請參閱預防跨服務混淆代理人。

以下是使用 add-permission 將 IoT 許可授予您 Lambda 的範例。

```
aws lambda add-permission \
    --function-name myLambdaFunction \
    --statement-id iot-permission \
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. 使用 create-provisioning-template 或 update-provisioning-template 命令，將預先啟動設定掛接新增至樣板。

下列 CLI 範例使用 create-provisioning-template 建立具有預先啟動設定掛接的佈建範本：

```
aws iot create-provisioning-template \
    --template-name myTemplate \
    --provisioning-role-arn arn:aws:iam:us-east-1:1234564789012:role/myRole \
    --template-body file://template.json \
    --pre-provisioning-hook file://hooks.json
```

此令命的輸出結果如下所示：

```
{
    "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
myTemplate",
    "defaultVersionId": 1,
    "templateName": myTemplate
}
```

您也可以從檔案載入參數，而不必一一輸入所有命令列參數值，以節省時間。如需詳細資訊，請參閱從檔案載入 AWS CLI 參數。以下顯示了擴展 JSON 格式的 `template` 參數：

```
{
    "Parameters" : {
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingTypeName" : {"Fn::Join":["",["ThingTypePrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["widgets", "WA"],
                "BillingGroup": "BillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
```

```
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:504350838278:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
    }
}
```

以下顯示了擴展 JSON 格式的 pre-provisioning-hook 參數：

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

佈建具有裝置憑證的裝置

AWS IoT 當裝置已有裝置憑證（和相關聯的私有金鑰）時，提供三種佈建裝置的方式：

- 含佈建範本的單一物件佈建。如果您一次只需要佈建一部裝置，適合使用此選項。
- Just-in-time佈建 (JITP)，可在首次連線時佈建裝置 AWS IoT。如果您需要註冊大量裝置，但您沒有裝置的相關資訊，無法組合到大量佈建清單內，適合使用此選項。

- 大量註冊 此選項可讓您指定儲存在 S3 儲存貯體內的檔案中的單一物件佈建範本值清單。如果您有大量已知裝置且您可將其所需特性組合到清單內，適合使用此選項。

主題

- 單一物件佈建
- 即時佈建
- 大量註冊

單一物件佈建

若要佈建物件，請使用 RegisterThing API 或 `register-thing` CLI 命令。`register-thing` CLI 命令會使用以下引數：

--template-body

該佈建範本。

--parameters

佈建範本使用的名稱/值對參數清單 (JSON 格式) (例如，`{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`)。

請參閱 佈建範本。

RegisterThing 或 `register-thing` 會針對資源及其建立的憑證文字，回傳 ARN：

```
{
    "certificatePem": "certificate-text",
    "resourceArns": {
    "PolicyLogicalName": "arn:aws:iot:us-
west-2:123456789012:policy/2A6577675B7CD1823E271C7AAD8184F44630FFD7",
    "certificate": "arn:aws:iot:us-west-2:123456789012:cert/
cd82bb924d4c6ccbb14986dcb4f40f30d892cc6b3ce7ad5008ed6542eea2b049",
    "thing": "arn:aws:iot:us-west-2:123456789012:thing/MyProvisionedThing"
    }
}
```

若字典省略參數，則會使用預設值。若未指定預設值，則參數不會替換成值。

即時佈建

您可以在裝置首次嘗試連線至 AWS IoT時，使用即時佈建 (JITP) 來佈建裝置。若要佈建裝置，您必須啟用自動註冊，而且您在佈建裝置時，其裝置憑證簽署所用的憑證授權機構憑證，必須與佈建範本建立關聯。佈建成功且錯誤會在 Amazon CloudWatch 中記錄為 裝置佈建指標。

主題

- JITP 概觀
- 使用佈建範本註冊 CA
- 使用佈建立範本名稱來註冊 CA

JITP 概觀

當裝置嘗試 AWS IoT 使用已註冊 CA 憑證簽署的憑證連線至 時， 會從 CA 憑證 AWS IoT 載入範本，並使用它呼叫 RegisterThing。JITP 工作流程會先註冊狀態值為 PENDING_ACTIVATION 的憑證。裝置佈建完成時，憑證狀態將變為 ACTIVE。

AWS IoT 定義了下列參數，您可以在佈建範本中宣告和參考這些參數：

- `AWS::IoT::Certificate::Country`
- `AWS::IoT::Certificate::Organization`
- `AWS::IoT::Certificate::OrganizationalUnit`
- `AWS::IoT::Certificate::DistinguishedNameQualifier`
- `AWS::IoT::Certificate::StateName`
- `AWS::IoT::Certificate::CommonName`
- `AWS::IoT::Certificate::SerialNumber`
- `AWS::IoT::Certificate::Id`

這些佈建範本參數的值僅限於 JITP 會自佈建中的裝置憑證的主旨欄位擷取的值。憑證必須包含範本主體中所有參數的值。`AWS::IoT::Certificate::Id` 參數代表的是內部產生的 ID，而不是憑證內包含的 ID。您可以使用 AWS IoT 規則內的 `principal()`函數來取得此 ID 的值。

> **Note**
>
> 您可以使用 AWS IoT Core just-in-time(JITP) 功能佈建裝置，而不必在裝置的第一個連線上傳送整個信任鏈 AWS IoT Core。雖不要求顯示憑證授權機構憑證，但需要裝置在連線至 AWS IoT Core時傳送伺服器名稱指示 (SNI) 延伸。

範例範本內文

以下 JSON 檔案為完整 JITP 範本的範例範本內文。

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
```

```
                {
                   "Ref":"AWS::IoT::Certificate::Country"
                }
             ]
          },
          "OverrideSettings":{
             "AttributePayload":"MERGE",
             "ThingTypeName":"REPLACE",
             "ThingGroups":"DO_NOTHING"
          }
       },
       "certificate":{
          "Type":"AWS::IoT::Certificate",
          "Properties":{
             "CertificateId":{
                "Ref":"AWS::IoT::Certificate::Id"
             },
             "Status":"ACTIVE"
          }
       },
       "policy":{
          "Type":"AWS::IoT::Policy",
          "Properties":{
             "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
          }
       }
    }
}
```

此範例範本會宣告擷取自憑證並使用在 Resources 部分的 AWS::IoT::Certificate::CommonName、AWS::IoT::Certificate::SerialNumber、AWS::IoT::Certificate::Country 和 AWS::IoT::Certificate::Id 佈建參數的值。JITP 工作流程接著使用此範本執行下列動作：

- 註冊憑證並將其狀態設為 PENDING_ACTIVE。
- 建立一個物件資源。
- 建立一個政策資源。
- 將政策連接至憑證。
- 將憑證連接至物件。
- 更新憑證狀態為 ACTIVE。

如果憑證沒有 Parameters區段中提及的所有屬性，裝置佈建會失敗templateBody。例如，如果 AWS::IoT::Certificate::Country 併入在範本中，但憑證沒有 Country 屬性，裝置佈建會失敗。

您也可以使用 CloudTrail 來排除 JITP 範本的問題。如需 Amazon CloudWatch 中所記錄之指標的相關資訊，請參閱裝置佈建指標。如需佈建範本的詳細資訊，請參閱佈建範本。

> **Note**
>
> 在佈建過程中，just-in-time佈建 (JITP) 會呼叫其他 AWS IoT 控制平面 API 操作。這些呼叫可能會超過針對您帳戶設定的 AWS IoT 調節配額，並導致調節呼叫。如有必要，請聯絡 AWS 客戶支援，以提高您的調節配額。

使用佈建範本註冊 CA

若要使用完整的佈建範本註冊 CA，請依照下列步驟執行：

1. 將您的佈建範本和角色 ARN 資訊 (如下列範例) 儲存為 JSON 檔案：

```
{
     "templateBody" : "{\r\n    \"Parameters\" : {\r\n
 \"AWS::IoT::Certificate::CommonName\": {\r\n            \"Type\": \"String\"\r
\n        },\r\n        \"AWS::IoT::Certificate::SerialNumber\": {\r\n
  \"Type\": \"String\"\r\n        },\r\n        \"AWS::IoT::Certificate::Country
\": {\r\n            \"Type\": \"String\"\r\n        },\r\n
 \"AWS::IoT::Certificate::Id\": {\r\n            \"Type\": \"String\"\r
\n        }\r\n    },\r\n    \"Resources\": {\r\n        \"thing\": {\r
\n            \"Type\": \"AWS::IoT::Thing\",\r\n            \"Properties
\": {\r\n                \"ThingName\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::CommonName\"\r\n                },\r\n
    \"AttributePayload\": {\r\n                    \"version\": \"v1\",\r\n
                \"serialNumber\": {\r\n                        \"Ref\":
 \"AWS::IoT::Certificate::SerialNumber\"\r\n                    }\r\n
    },\r\n                \"ThingTypeName\": \"lightBulb-versionA\",\r\n
       \"ThingGroups\": [\r\n                    \"v1-lightbulbs\",\r\n
          {\r\n                        \"Ref\": \"AWS::IoT::Certificate::Country
\"\r\n                    }\r\n                ]\r\n            },\r\n
  \"OverrideSettings\": {\r\n                \"AttributePayload\": \"MERGE\",\r\n
              \"ThingTypeName\": \"REPLACE\",\r\n                \"ThingGroups
\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"certificate\": {\r
\n            \"Type\": \"AWS::IoT::Certificate\",\r\n            \"Properties
```

```
\": {\r\n                \"CertificateId\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::Id\"\r\n                },\r\n                \"Status\":
 \"ACTIVE\"\r\n            },\r\n            \"OverrideSettings\": {\r\n
     \"Status\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"policy
\": {\r\n            \"Type\": \"AWS::IoT::Policy\",\r\n            \"Properties
\": {\r\n                \"PolicyDocument\": \"{ \\\"Version\\\": \\\"2012-10-17
     \\\", \\\"Statement\\\": [{ \\\"Effect\\\": \\\"Allow\\\", \\\"Action\\\":[\\
\"iot:Publish\\\"], \\\"Resource\\\": [\\\"arn:aws:iot:us-east-1:123456789012:topic
\/foo\/bar\\\"] }] }\"\r\n            }\r\n        }\r\n    }\r\n}",
     "roleArn" : "arn:aws:iam::123456789012:role/JITPRole"
}
```

在此範例中，`templateBody` 欄位的值必須是指定為溢出字串的 JSON 物件，且只能使用前述清單中的值。您可以使用各種工具來建立必要的 JSON 輸出，例如 `json.dumps` (Python) 或 `JSON.stringify` (節點)。`roleARN` 欄位的值必須為連接 `AWSIoTThingsRegistration` 之角色的 ARN。此外，您的範本可以使用現有的 `PolicyName`，而不是範例中的內嵌 `PolicyDocument`。

2. 在使用 RegisterCACertificate API 操作或 `register-ca-certificate` CLI 命令來註冊 CA 憑證時。您將指定啟動佈建範本的目錄，以及您在上一個步驟中儲存的角色 ARN 資訊：

   以下範例展示如何使用 AWS CLI在 `DEFAULT` 模式下註冊 CA 憑證：

   ```
   aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
   verification-cert file://your-verification-cert
                  --set-as-active --allow-auto-registration --registration-config
    file://your-template
   ```

   以下範例展示如何使用 AWS CLI在 `SNI_ONLY` 模式下註冊 CA 憑證：

   ```
   aws iot register-ca-certificate --ca-certificate file://your-ca-cert --certificate-
   mode SNI_ONLY
                   --set-as-active --allow-auto-registration --registration-config
    file://your-template
   ```

   如需詳細資訊，請參閱註冊 CA 憑證。

3. (選用) 使用 UpdateCACertificate API 操作或 `update-ca-certificate` CLI 命令來更新 CA 憑證的設定。

   以下範例展示如何使用 AWS CLI更新 CA 憑證：

```
aws iot update-ca-certificate --certificate-id caCertificateId
                --new-auto-registration-status ENABLE --registration-config
 file://your-template
```

使用佈建立範本名稱來註冊 CA

若要使用佈建範本名稱註冊 CA，請依照下列步驟執行：

1. 將佈建範本內文儲存為 JSON 檔案。您可以在範例範本內文中找到範例範本內文。
2. 若要建立佈建範本，請使用 CreateProvisioningTemplate API 或 `create-provisioning-template` CLI 命令：

```
aws iot create-provisioning-template --template-name your-template-name \
        --template-body file://your-template-body.json --type JITP \
        --provisioning-role-arn arn:aws:iam::123456789012:role/test
```

> **Note**
>
> 針對即時佈建 (JITP)，您必須在建立佈建範本時，將範本類型指定為 `JITP`。如需範本類型的詳細資訊，請參閱《AWS API 參考》中的 CreateProvisioningTemplate。

3. 若要使用範本名稱註冊 CA，請使用 RegisterCACertificate API 或 `register-ca-certificate` CLI 命令：

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
verification-cert file://your-verification-cert \
        --set-as-active --allow-auto-registration --registration-config
 templateName=your-template-name
```

大量註冊

您可以使用 `start-thing-registration-task` 命令來大量註冊物件。此命令需要佈建範本、S3 儲存貯體名稱、金鑰名稱，以及可存取 S3 儲存貯體檔案的角色 ARN。S3 儲存貯體的檔案，內含用於替換範本參數的值。該檔案必須為換行分隔的 JSON 檔案。每行均包含用於註冊單一裝置的所有參數值。例如：

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

下列與大量註冊相關的 API 操作也許有用：

- ListThingRegistrationTasks：列出目前的大量實物佈建任務。
- DescribeThingRegistrationTask：提供特定大量物件註冊任務的相關資訊。
- StopThingRegistrationTask：停止大量物件註冊任務。
- ListThingRegistrationTaskReports：用於檢查大量物件註冊任務的結果及/或失敗。

> **Note**
>
> - 一次僅可執行一個大量註冊操作任務 (每個帳戶)。
> - 大量註冊操作會呼叫其他 AWS IoT 控制平面 API 操作。這些呼叫可能會超過您帳戶中的 AWS IoT 調節配額，並導致調節錯誤。如有必要，請聯絡 AWS 客戶支援以提高您的限 AWS IoT 流配額。

佈建範本

佈建範本是 JSON 文件，使用參數來描述裝置與之互動時必須使用的資源 AWS IoT。佈建範本內含兩個部分：`Parameters` 和 `Resources`。佈建範本有兩種類型 AWS IoT。一個用於即時佈建 (JITP) 和大量註冊，第二個則用於機群佈建。

主題

- 參數部分
- 資源部分
- 大量註冊的範本範例
- 即時佈建 (JITP) 的範本範例
- 機群佈建

參數部分

Parameters 部分宣告 Resources 部分所使用的參數。每個參數都會宣告名稱、類型和可選的預設值。與範本一同傳入的字典若未包含該參數的值，將使用預設值。範本文件的 Parameters 部分如下：

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

此範本內文片段宣告四項參數：ThingName、SerialNumber、Location 和 CSR。這些參數均為類型 String。Location 參數宣告的預設值為 "WA"。

資源部分

範本內文的 Resources區段會宣告裝置與之通訊所需的資源 AWS IoT：物件、憑證，以及一或多個 IoT 政策。每一資源都要指定一個邏輯名稱、一種類型和一組屬性。

邏輯名稱可讓您引用範本其他位置的資源。

類型則指定您欲宣告的資源種類。有效類型為：

- AWS::IoT::Thing
- AWS::IoT::Certificate
- AWS::IoT::Policy

您指定的屬性取決於您欲宣告的資源類型。

物件資源

物件資源宣告使用下列屬性：

- `ThingName`：字串。
- `AttributePayload`：選用。名稱/值對清單。
- `ThingTypeName`：選用。與該物件類型相關的字串。
- `ThingGroups`：選用。該物件所屬的群組清單。
- `BillingGroup`：選用。關聯帳單群組名稱的字串。
- `PackageVersions`：選用。相關套件和版本名稱的字串。

憑證資源

您可以使用下列其中一種方式來指定憑證：

- 憑證簽署要求 (CSR)。
- 現有裝置憑證的憑證 ID。(只有憑證 ID 可與機群佈建範本搭配使用。)
- 搭配憑證授權機構憑證向 AWS IoT註冊所建立的裝置憑證。如果您的多個 憑證授權機構憑證註冊使用相同的主體欄位，您也必須輸入用於簽署該裝置憑證的憑證授權機構憑證。

> **Note**
>
> 若您使用範本宣告憑證，請透過這些方法其中之一。例如，若您使用 CSR，則無法同時指定憑證 ID 或裝置憑證。如需詳細資訊，請參閱X.509 用戶端憑證。

如需詳細資訊，請參閱X.509 憑證概觀。

憑證資源宣告使用下列屬性：

- `CertificateSigningRequest`：字串。
- `CertificateId`：字串。
- `CertificatePem`：字串。
- `CACertificatePem`：字串。
- `Status`：選用。可以是 `ACTIVE` 或 `INACTIVE` 的字串。預設為 ACTIVE。

- ThingPrincipalType：選用。指定物件與委託人（憑證）之間關係類型的字串。
  - EXCLUSIVE_THING：建立專屬關係。主體只能連接到此特定物件，不能連接到其他物件。
  - NON_EXCLUSIVE_THING：將指定的主體連接到物件。您可以將多個物件連接至委託人。如果未指定，則此為預設值。

> **Note**
>
> 您也可以在沒有裝置憑證的情況下佈建裝置。如需詳細資訊，請參閱使用機群佈建來佈建沒有裝置憑證的裝置。

範例：

- CSR 指定的憑證：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- 以現有憑證 ID 指定的憑證：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- 以現有憑證 .pem 和憑證授權機構憑證 .pem 指定的憑證：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
```

```
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

- 僅將一個物件連接至委託人：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "ThingPrincipalType" : "EXCLUSIVE_THING"
        }
    }
}
```

政策資源

政策資源宣告使用下列屬性之一：

- PolicyName：選用。字串. 預設為政策文件雜湊。此 PolicyName 只能參考 AWS IoT 政策，而不是 IAM 政策。如果您使用的是現有 AWS IoT 政策，則請在 PolicyName 屬性輸入該政策名稱。請勿包含 PolicyDocument 屬性。
- PolicyDocument：選用。指定為逸出字串的 JSON 物件。若未提供 PolicyDocument，則該政策必已建立。

> **Note**
>
> 若有 Policy 部分，必須指定 PolicyName 或 PolicyDocument 其一。

覆寫設定

若範本指定已存在的資源，OverrideSettings 部分可讓您指定欲採取的動作：

DO_NOTHING

保留原本資源。

REPLACE

以範本指定的資源來取代原有資源。

FAIL

因 ResourceConflictsException 造成請求失敗。

MERGE

僅適用 ThingGroups 的 AttributePayload 及 thing 屬性。將物件現有屬性或群組成員資格，與範本指定的屬性合併。

當您宣告物件資源時，可為下列屬性指定 OverrideSettings：

- ATTRIBUTE_PAYLOAD
- THING_TYPE_NAME
- THING_GROUPS

當您宣告憑證資源時，可為 OverrideSettings 屬性指定 Status。

OverrideSettings 不適用於政策資源。

資源範例

下列範本片段宣告一個物件、一個憑證及一個政策：

```
{
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
```

```
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

物件以下列項目進行宣告：

- 邏輯名稱 "thing"。
- 類型 AWS::IoT::Thing。
- 物件的一組屬性。

  物件的屬性包含物件名稱、一組屬性、選用的物件類型名稱，以及該物件所屬的物件群組選用清單。

{"Ref":"*parameter-name*"} 會參考這些參數。在評估範本時，參數將取代為與範本一同傳入的字典的參數值。

憑證以下列項目進行宣告：

- 邏輯名稱 "certificate"。
- 類型 AWS::IoT::Certificate。
- 一組屬性。

  這些屬性包括憑證的 CSR 及設定為 ACTIVE 的狀態。與範本一同傳入的字典內，CSR 文字以參數傳遞。

政策以下列項目進行宣告：

- 邏輯名稱 "policy"。
- 類型 AWS::IoT::Policy。
- 現有政策名稱或政策文件。

大量註冊的範本範例

下列 JSON 檔案為以 CSR 指定憑證的完整佈建範本的其中一個範例：

(PolicyDocument 欄位值必須是指定為逸出字串的 JSON 物件。)

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE",
```

```
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

即時佈建 (JITP) 的範本範例

下列 JSON 檔案為以憑證 ID 指定現有憑證的完整佈建範本的其中一個範例：

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
```

```
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE",
            "ThingPrincipalType" : "EXCLUSIVE_THING"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

> **Important**
>
> 您必須在用於 JIT 佈建的範本中使用 `CertificateId`。

如需佈建範本類型的詳細資訊，請參閱 AWS API 參考CreateProvisioningTemplate中的 。

如需了解如何使用此範本進行即時佈建，請參閱：即時佈建。

機群佈建

機群佈建範本由 AWS IoT 用來設定雲端和裝置組態。這些範本會使用與 JITP 和大量註冊範本相同的參數和資源。如需詳細資訊，請參閱佈建範本。機群佈建範本可以包含一個 Mapping 區段和一個 DeviceConfiguration 區段。您可以使用機群佈建範本內的內建函數來產生裝置特定的組態。機群佈建範本是具名資源，由 ARN 識別 (例如，arn:aws:iot:us-west-2:1234568788:provisioningtemplate/*templateName*)。

映射項目

選用的 Mappings 區段會比對索引鍵與對應的一組命名值。例如，如果您想要根據 AWS 區域設定值，您可以建立使用 AWS 區域 名稱做為索引鍵的映射，並包含您要為每個特定區域指定的值。您可以使用 Fn::FindInMap 內部函數來擷取映射中的值。

您不能在 Mappings 區段中包含參數、虛擬參數或呼叫內部函數。

裝置組態

裝置組態區段包含您要在佈建時傳送至裝置的任意資料。例如：

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
}
```

如果您使用 JavaScript 物件標記法 (JSON) 承載格式傳送訊息至裝置， 會將此資料 AWS IoT Core 格式化為 JSON。如果您使用的是 Concise Binary Object Representation (CBOR) 承載格式， AWS IoT Core 會將此資料格式化為 CBOR。DeviceConfiguration 區段不支援巢狀 JSON 物件。

內建函數

內部函數用於佈建範本的任何區段，但 Mappings 區段除外。

Fn::Join

將一組值附加至單一值，並以指定的分隔符號隔開。如果分隔符號是空白字串，系統即會串連這些值，而不使用分隔符號。

> **Important**
>
> the section called “政策資源” 不支援 `Fn::Join`。

`Fn::Select`

依索引從物件清單傳回單一物件。

> **Important**
>
> `Fn::Select` 不會檢查 `null` 值或索引是否超出陣列邊界。這兩個條件都會導致佈建錯誤，因此請務必選擇有效的索引值，且清單包含非空值。

`Fn::FindInMap`

傳回對應 `Mappings` 區段所宣告之兩個層級映射之索引鍵的值。

`Fn::Split`

將字串分割成字串值清單，以便您可以從字串清單中選取元素。您可以指定分隔符號來決定分割字串的位置 (例如，逗號)。分割字串後，請使用 `Fn::Select` 來選取元素。

例如，如果以逗號分隔的子網路 ID 字串匯入至您的堆疊範本，您可以在每個逗號處分割字串。從子網路 ID 清單中，使用 `Fn::Select` 來指定資源的子網路 ID。

`Fn::Sub`

用您指定的值替代輸入字串中的變數。您可以使用此函數來建構命令或輸出，其中包含建立或更新堆疊後才可供使用的數值。

機群佈建的範本範例

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber": {
            "Type": "String"
        },
```

```
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Ref" : "ThingName"},
                "ThingTypeName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["v1-lightbulbs", "WA"],
                "BillingGroup": "LightBulbBillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
```

```
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
        }
}
```

> **Note**
>
> 您可以更新現有的佈建範本，以新增 pre-provisioning hook。

預先佈建掛接

AWS 建議在建立佈建範本時使用預先佈建掛接函數，以進一步控制您的帳戶加入哪些裝置和裝置數量。預先佈建掛接是 Lambda 函數，會先驗證從裝置傳遞的參數，然後才能佈建裝置。此 Lambda 函數必須存在於您的帳戶中，才能佈建裝置，因為每次裝置透過 the section called “RegisterThing” 傳送要求時，都會呼叫該函數。

> **Important**
>
> 請務必在連接到 Lambda 動作的政策的全域條件內容金鑰中包含 `source-arn` 或 `source-account`，以防止許可操作。如需此項目的詳細資訊，請參閱預防跨服務混淆代理人。

對於要佈建的裝置，您的 Lambda 函數必須接受輸入物件，並傳回本節所述的輸出物件。只有當 Lambda 函數透過 `"allowProvisioning": True` 傳回物件時，才會繼續佈建。

預先佈建掛接輸入

AWS IoT 當裝置向 註冊時， 會將此物件傳送至 Lambda 函數 AWS IoT。

```
{
    "claimCertificateId" : "string",
    "certificateId" : "string",
    "certificatePem" : "string",
    "templateArn" : "arn:aws:iot:us-east-1:1234567890:provisioningtemplate/MyTemplate",
    "clientId" : "221a6d10-9c7f-42f1-9153-e52e6fc869c1",
    "parameters" : {
        "string" : "string",
        ...
    }
}
```

傳遞給 Lambda 函數的 `parameters` 物件包含在 the section called “RegisterThing” 請求承載中傳遞之 `parameters` 引數中的屬性。

預先佈建掛接傳回值

Lambda 函數必須傳回回應，指出它是否已授權佈建請求，以及要覆寫的任何屬性值。

以下是預先佈建函數成功回應的範例。

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
        "Key": "newCustomValue",
        ...
    }
}
```

`"parameterOverrides"` 值將新增至 the section called “RegisterThing” 請求承載的 `"parameters"` 參數。

> **Note**
>
> - 如果 Lambda 函數失敗，則佈建要求會失敗，顯示 `ACCESS_DENIED`，且錯誤會記錄到 CloudWatch Logs。

- 如果 Lambda 函數沒有在回覆中傳回 `"allowProvisioning": "true"`，則佈建要求會失敗，顯示 `ACCESS_DENIED`。
- 該 Lambda 函數必須在 5 秒內完成執行並返回，否則佈建請求將失敗。

預先佈建掛接 Lambda 範例

Python

Python 中預先佈建掛接 Lambda 的範例。

```
import json

def pre_provisioning_hook(event, context):
    print(event)

    return {
        'allowProvisioning': True,
        'parameterOverrides': {
            'DeviceLocation': 'Seattle'
        }
    }
```

Java

Java 中預先佈建掛接 Lambda 的範例。

處理常式類別：

```
package example;

import java.util.Map;
import java.util.HashMap;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class PreProvisioningHook implements
 RequestHandler<PreProvisioningHookRequest, PreProvisioningHookResponse> {

    public PreProvisioningHookResponse handleRequest(PreProvisioningHookRequest
 object, Context context) {
```

```
        Map<String, String> parameterOverrides = new HashMap<String, String>();
        parameterOverrides.put("DeviceLocation", "Seattle");

        PreProvisioningHookResponse response = PreProvisioningHookResponse.builder()
                .allowProvisioning(true)
                .parameterOverrides(parameterOverrides)
                .build();

        return response;
    }

}
```

請求類別：

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookRequest {
    private String claimCertificateId;
    private String certificateId;
    private String certificatePem;
    private String templateArn;
    private String clientId;
    private Map<String, String> parameters;
}
```

回應類別：

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
```

```
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;


@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookResponse {
    private boolean allowProvisioning;
    private Map<String, String> parameterOverrides;
}
```

JavaScript

JavaScript 中預先佈建掛接 Lambda 的範例。

```
exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    var reply = {
        allowProvisioning: true,
        parameterOverrides: {
            DeviceLocation: 'Seattle'
        }
     };
     callback(null, reply);
}
```

使用憑證提供者進行自我管理 AWS IoT Core 的憑證簽署

您可以建立 AWS IoT Core 憑證提供者，在 AWS IoT 機群佈建中簽署憑證簽署請求 CSRs)。憑證提供者參考 Lambda 函數和 CreateCertificateFromCsr MQTT API 進行機群佈建。Lambda 函數接受 CSR 並傳回已簽署的用戶端憑證。

當您的 沒有憑證提供者時 AWS 帳戶，會在機群佈建中呼叫 CreateCertificateFromCsr MQTT API，以從 CSR 產生憑證。建立憑證提供者之後，CreateCertificateFromCsr MQTT API 的行為會變更，而對此 MQTT API 的所有呼叫都會叫用憑證提供者來發出憑證。

透過 AWS IoT Core 憑證提供者，您可以實作解決方案，利用私有憑證授權單位 (CAs)，例如 AWS 私有 CA、其他公開信任CAs，或您自己的公有金鑰基礎設施 (PKI) 來簽署 CSR。此外，您可以使用憑證提供者自訂用戶端憑證的欄位，例如有效期間、簽署演算法、發行者和延伸項目。

> **Important**
>
> 每個 只能建立一個憑證提供者 AWS 帳戶。簽署行為變更適用於呼叫 CreateCertificateFromCsr MQTT API 的整個機群，直到您從 中刪除憑證提供者為止 AWS 帳戶。

在本主題中：

- 自我管理憑證簽署如何在機群佈建中運作
- 憑證提供者 Lambda 函數輸入
- 憑證提供者 Lambda 函數傳回值
- Lambda 函數範例
- 機群佈建的自我管理憑證簽署
- AWS CLI 憑證提供者的 命令

自我管理憑證簽署如何在機群佈建中運作

重要概念

下列概念提供詳細資訊，可協助您了解自我管理憑證簽署如何在 AWS IoT 機群佈建中運作。如需詳細資訊，請參閱使用機群佈建佈建沒有裝置憑證的裝置。

AWS IoT 機群佈建

透過機 AWS IoT 群佈建 （機群佈建的簡稱）， 會在 AWS IoT Core 裝置第一次連線至 時 AWS IoT Core ，產生並安全地將裝置憑證交付至您的裝置。您可以使用機群佈建來連接沒有裝置憑證的裝置 AWS IoT Core。

憑證簽署請求 (CSR)

在機群佈建過程中，裝置 AWS IoT Core 會透過機群佈建 MQTT APIs 向 發出請求。此請求包含憑證簽署請求 (CSR)，該請求將簽署以建立用戶端憑證。

AWS 機群佈建中的受管憑證登入

AWS 受管是機群佈建中憑證簽署的預設設定。透過 AWS 受管憑證簽署， AWS IoT Core 將使用自己的 CAs簽署 CSRs。

機群佈建中的自我管理憑證簽署

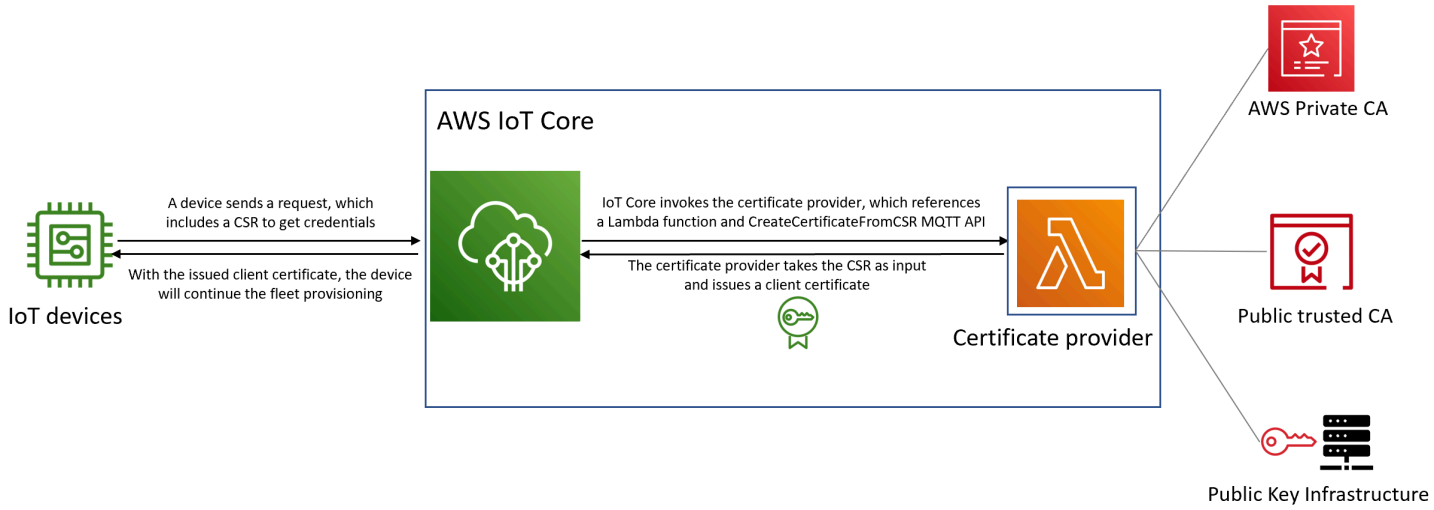
自我管理是在機群佈建中簽署憑證的另一個選項。透過自我管理憑證簽署，您可以建立 AWS IoT Core 憑證提供者來簽署 CSRs。您可以使用自我管理憑證簽署，透過 AWS 私有 CA、其他公開信任 CA 或您自己的公有金鑰基礎設施 (PKI) 產生的 CA 簽署 CSRs。

AWS IoT Core 憑證提供者

AWS IoT Core 憑證提供者 （憑證提供者的簡稱） 是客戶管理的資源，用於在機群佈建中進行自我管理憑證簽署。

圖表

下圖是自我認證簽署如何在 AWS IoT 機群佈建中運作的簡化說明。



- 當新的 IoT 裝置製造或引入機群時，它需要用戶端憑證來驗證自己 AWS IoT Core。
- 在機群佈建程序中，裝置會透過機群佈建 MQTT APIs 向 提出用戶端憑證 AWS IoT Core 的請求。此請求包含憑證簽署請求 (CSR)。
- AWS IoT Core 叫用憑證提供者，並將 CSR 做為輸入傳遞給提供者。
- 憑證提供者接受 CSR 做為輸入，並發出用戶端憑證。

  對於 AWS 受管憑證簽署， 會使用自己的 CA AWS IoT Core 簽署 CSR 並發出用戶端憑證。
- 使用發行的用戶端憑證，裝置會繼續機群佈建，並與 建立安全連線 AWS IoT Core。

憑證提供者 Lambda 函數輸入

AWS IoT Core 當裝置向 Lambda 函數註冊時， 會將下列物件傳送至該函數。的值`certificateSigningRequest`是`CreateCertificateFromCsr`請求中提供的隱私權增強郵件 (PEM) 格式的 CSR。`principalId` 是在提出`CreateCertificateFromCsr`請求 AWS IoT Core 時用來連線 的委託人 ID。 `clientId`是 MQTT 連線的用戶端 ID 集。

```
{
 "certificateSigningRequest": "string",
 "principalId": "string",
 "clientId": "string"
}
```

憑證提供者 Lambda 函數傳回值

Lambda 函數必須傳回包含 `certificatePem`值的回應。以下是成功回應的範例。 AWS IoT Core 將使用傳回值 (`certificatePem`) 來建立憑證。

```
{
 "certificatePem": "string"
}
```

如果註冊成功， `CreateCertificateFromCsr`會在`CreateCertificateFromCsr`回應`certificatePem`中傳回相同的 。如需詳細資訊，請參閱 CreateCertificateFromCsr 的回應承載範例。

Lambda 函數範例

在建立憑證提供者之前，您必須建立 Lambda 函數來簽署 CSR。以下是 Python 中的 Lambda 函數範例。此函數 AWS 私有 CA 會使用私有 CA 和`SHA256WITHRSA`簽署演算法呼叫 來簽署輸入 CSR。傳回的用戶端憑證有效期為一年。如需 AWS 私有 CA 和如何建立私有 CA 的詳細資訊，請參閱什麼是 AWS 私有 CA？ 和建立私有 CA。

```
import os
import time
import uuid
import boto3

def lambda_handler(event, context):
```

```
    ca_arn = os.environ['CA_ARN']
    csr = (event['certificateSigningRequest']).encode('utf-8')

    acmpca = boto3.client('acm-pca')
    cert_arn = acmpca.issue_certificate(
        CertificateAuthorityArn=ca_arn,
        Csr=csr,
        Validity={"Type": "DAYS", "Value": 365},
        SigningAlgorithm='SHA256WITHRSA',
        IdempotencyToken=str(uuid.uuid4())
    )['CertificateArn']

    # Wait for certificate to be issued
    time.sleep(1)
    cert_pem = acmpca.get_certificate(
        CertificateAuthorityArn=ca_arn,
        CertificateArn=cert_arn
    )['Certificate']

    return {
        'certificatePem': cert_pem
    }
```

Important

- Lambda 函數傳回的憑證必須具有與憑證簽署請求 (CSR) 相同的主體名稱和公有金鑰。
- Lambda 函數必須在 5 秒內完成執行。
- Lambda 函數必須與憑證提供者資源位於相同的 AWS 帳戶 和 區域。
- 必須授予 AWS IoT 服務主體叫用 Lambda 函數的許可。為了避免混淆代理人問題，建議您`sourceAccount`為調用許可設定 `sourceArn`和 。如需詳細資訊，請參閱預防跨服務混淆代理人。

下列以資源為基礎的 Lambda 政策範例 AWS IoT 會授予叫用 Lambda 函數的許可：

```
{
 "Version":"2012-10-17",
 "Id": "InvokePermission",
 "Statement": [
```

```
  {
   "Sid": "LambdaAllowIotProvider",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "lambda:InvokeFunction",
   "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
   "Condition": {
    "StringEquals": {
     "AWS:SourceAccount": "123456789012"
    },
    "ArnLike": {
    "AWS:SourceArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider/my-
certificate-provider"
    }
   }
  }
 ]
}
```

機群佈建的自我管理憑證簽署

您可以使用 或 為機群佈建 AWS CLI 選擇自我管理憑證簽署 AWS 管理主控台。

AWS CLI

若要選擇自我管理憑證簽署，您必須建立 AWS IoT Core 憑證提供者以在機群佈建中簽署 CSRs。 會 AWS IoT Core 叫用憑證提供者，以 CSR 做為輸入並傳回用戶端憑證。若要建立憑證提供者，請使用 `CreateCertificateProvider` API 操作或 CLI `create-certificate-provider` 命令。

> **Note**
>
> 建立憑證提供者之後，機CreateCertificateFromCsr群佈建 API 的行為將會變更，因此的所有呼叫CreateCertificateFromCsr都會叫用憑證提供者來建立憑證。建立憑證提供者後，此行為可能需要幾分鐘的時間才能變更。

```
aws iot create-certificate-provider \
               --certificateProviderName my-certificate-provider \
```

```
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下顯示此命令的範例輸出：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

如需詳細資訊，請參閱 AWS IoT API CreateCertificateProvider 參考中的 。

AWS 管理主控台

若要使用 選擇自我管理憑證簽署 AWS 管理主控台，請遵循下列步驟：

1. 前往 AWS IoT 主控台。
2. 在左側導覽的 安全性下，選擇憑證簽署。
3. 在憑證簽署頁面的憑證簽署詳細資訊下，選擇編輯憑證簽署方法。
4. 在編輯憑證簽署方法頁面的憑證簽署方法下，選擇自我管理。
5. 在自我管理設定區段中，輸入憑證提供者的名稱，然後建立或選擇 Lambda 函數。
6. 選擇更新憑證簽署。

AWS CLI 憑證提供者的 命令

建立憑證提供者

若要建立憑證提供者，請使用 CreateCertificateProvider API 操作或 CLI create-certificate-provider 命令。

> **Note**
>
> 建立憑證提供者之後，機CreateCertificateFromCsr群佈建 API 的行為將會變更，因此所有對 的呼叫CreateCertificateFromCsr都會叫用憑證提供者來建立憑證。建立憑證提供者後，此行為可能需要幾分鐘的時間才能變更。

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下顯示此命令的範例輸出：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

如需詳細資訊，請參閱 AWS IoT API CreateCertificateProvider 參考中的 。

更新憑證提供者

若要更新憑證提供者，請使用 UpdateCertificateProvider API 操作或 CLI update-certificate-provider 命令。

```
aws iot update-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-2 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下顯示此命令的範例輸出：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

如需詳細資訊，請參閱 API UpdateCertificateProvider 參考中的 。 AWS IoT

描述憑證提供者

若要描述憑證提供者，請使用 DescribeCertificateProvider API 操作或 CLI describe-certificate-provider 命令。

```
aws iot describe-certificate-provider --certificateProviderName my-certificate-provider
```

以下顯示此命令的範例輸出：

```
{
 "certificateProviderName": "my-certificate-provider",
 "lambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
 "accountDefaultForOperations": [
  "CreateCertificateFromCsr"
 ],
 "creationDate": "2022-11-03T00:15",
 "lastModifiedDate": "2022-11-18T00:15"
}
```

如需詳細資訊，請參閱 AWS IoT API DescribeCertificateProvider 參考中的 。

刪除憑證提供者

若要刪除憑證提供者，請使用 DeleteCertificateProvider API 操作或 CLI delete-certificate-provider 命令。如果您刪除憑證提供者資源， 的行為CreateCertificateFromCsr將會繼續，並且 AWS IoT 會從 AWS IoT CSR 建立由 簽署的憑證。

```
aws iot delete-certificate-provider --certificateProviderName my-certificate-provider
```

此命令不會產生任何輸出。

如需詳細資訊，請參閱 AWS IoT API DeleteCertificateProvider 參考中的 。

列出憑證提供者

若要列出 中的憑證提供者 AWS 帳戶，請使用 ListCertificateProviders API 操作或 CLI list-certificate-providers 命令。

```
aws iot list-certificate-providers
```

以下顯示此命令的範例輸出：

```
{
```

```
 "certificateProviders": [
  {
   "certificateProviderName": "my-certificate-provider",
   "certificateProviderArn": "arn:aws:iot:us-
east-1:123456789012:certificateprovider:my-certificate-provider"
  }
 ]
}
```

如需詳細資訊，請參閱 AWS IoT API `ListCertificateProvider` 參考中的 。

為安裝裝置的使用者建立 IAM 政策和角色

> **Note**
>
> 這些程序僅在 AWS IoT 主控台指示時使用。
> 若要從主控台前往此頁面，請開啟建立新的佈建範本。

為什麼無法在 AWS IoT 主控台中執行此操作？

為了獲得最安全的體驗，IAM 動作會在 IAM 主控台中執行。本區段中的程序會逐步引導您建立使用佈建範本所需的 IAM 角色和政策。

為安裝裝置的使用者建立 IAM 政策

此程序說明如何建立 IAM 政策，以授權使用者使用佈建範本安裝裝置。

執行此程序時，您會在 IAM 主控台和 AWS IoT 主控台之間切換。我們建議您在執行此程序時同時開啟兩個主控台。

為安裝裝置的使用者建立 IAM 政策

1. 請開啟 IAM 主控台的政策中樞。
2. 選擇建立政策。
3. 在建立政策頁面上，選擇 JSON 標籤。
4. 在 AWS IoT 主控台中切換到您選擇設定使用者政策和角色的頁面。
5. 在 Sample provisioning policy (範例佈建政策) 中選擇 Copy (複製)。

6. 切換回 IAM 主控台。
7. 在 JSON 編輯器中，貼上您從 AWS IoT 主控台複製的政策。此政策專屬於您在 AWS IoT 主控台中建立的範本。
8. 若要繼續，請選擇 Next: Tags (下一步：標籤)。
9. 針對您要新增到此政策的每個標籤在 Add tags (Optional) (新增標籤 (選擇性)) 頁面上選擇 Add tag (新增標籤)。如果您沒有要新增任何標籤，則可省略此步驟。
10. 若要繼續，請選擇 Next: Review (下一步：檢閱)。
11. 在 Review Policy (檢閱政策) 頁面上，執行下列動作：
    a. 針對 Name* (姓名*)，請輸入能協助您記住政策用途的政策名稱。

       請記住您提供給此政策的名稱，因為後續程序仍會使用該名稱。
    b. 您可以為您建立的政策選擇性地輸入描述。
    c. 檢閱本政策的其餘部分及其標籤。
12. 請選擇 Create Policy (建立政策) 來完成您的政策建立程序。

建立新政策之後，請繼續 the section called “為安裝裝置的使用者建立 IAM 角色” 以建立您要連接到此政策的使用者角色項目。

為安裝裝置的使用者建立 IAM 角色

這些步驟說明如何建立 IAM 角色，以驗證使用佈建範本安裝裝置的使用者。

為安裝裝置的使用者建立 IAM 政策

1. 開啟 IAM 主控台中 Roles (角色) 頁面。
2. 選擇建立角色。
3. 在 Select trusted entity (選擇信任實體) 選擇您要授予存取權的信任實體類型，這些實體類型能存取您建立範本。
4. 選擇或輸入您要授予存取許可的信任實體識別，然後選擇 Next (下一頁)。
5. 在 Add permissions (新增許可) 頁面的 Permission policies (許可政策) 上，於搜尋方塊中輸入您在上一個程序中建立的政策名稱。
6. 針對政策清單，請選擇您在前一個程序所建立的政策，然後選擇 Next (下一頁)。
7. 在 Name, review, and create (命名、檢閱和建立) 區段上，執行以下作業：

a. 針對 Role name (角色名稱)，請輸入可協助您記住此角色用途的角色名稱。

b. 針對 Description (描述)，您可以選擇輸入角色的選擇性描述。不需要此操作即可繼續。

c. 檢閱 Step 1 (步驟 1) 和 Step 2 (步驟 2) 的值。

d. 針對 Add tags (Optional) (新增標籤 (選擇性))，您可以選擇新增標籤至此角色。不需要此操作即可繼續。

e. 確認此頁面上的資訊是否完整無誤，然後選擇 Create role (建立角色)。

建立新角色之後，請返回 AWS IoT 主控台以繼續建立範本。

更新現有政策以授權新範本

下列步驟說明如何將新範本新增至 IAM 政策，以授權使用者使用佈建範本安裝裝置。

將新的範本新增至現有 IAM 政策

1. 請開啟 IAM 主控台的政策中樞。
2. 在搜尋方塊中，輸入政策名稱。
3. 在搜尋方塊下的清單中找到您要更新的政策，然後選擇政策名稱。
4. 針對 Policy summary (政策摘要)，如果尚未顯示該面板，請選擇 JSON 標籤。
5. 如果想要編輯政策，請選擇 Edit policy (編輯政策)。
6. 如果尚未顯示該面板，請在編輯器中選擇 JSON 標籤。
7. 在政策文件中，尋找包含 `iot:CreateProvisioningClaim` 動作的政策陳述式。

   如果政策文件不包含有 `iot:CreateProvisioningClaim` 動作的政策陳述式，請複製下列陳述式程式碼片段，並將其做為其他項目貼到政策文件中的 `Statement` 陣列。

> **Note**
>
> 這個程式碼片段必須放在 `Statement` 陣列中的結尾 `]` 字元之前。您可能需要在此程式碼片段之前或之後加上逗號，以更正任何語法錯誤。

```
{
    "Effect": "Allow",
    "Action": [
```

```
            "iot:CreateProvisioningClaim"
        ],
        "Resource": [
            "--PUT YOUR NEW TEMPLATE ARN HERE--"
        ]
}
```

8. 在 AWS IoT 主控台中切換到您選擇修改使用者角色許可的頁面。
9. 尋找 Resource ARN (資源 ARN) 範本，並選擇 Copy (複製)。
10. 切換回 IAM 主控台。
11. 將複製的 Amazon Resource Name (ARN) 貼到範本 ARN 的頂端 `Statement` 陣列成為第一個項目。

    如果這是陣列中唯一的 ARN，請移除貼上值的結尾逗號。
12. 檢閱更新後的政策陳述式，並更正編輯器指出的任何錯誤。
13. 若要儲存更新的政策文件，請選擇 Review policy (檢閱政策)。
14. 檢閱政策然後選擇 Save changes (儲存變更)。
15. 返回 AWS IoT 主控台。

裝置佈建 MQTT API

機群佈建服務支援下列 MQTT API 操作：

- the section called “CreateCertificateFromCsr”
- the section called “CreateKeysAndCertificate”
- the section called “RegisterThing”

這個 API 支援 Concise Binary Object Representation (CBOR) 格式和 JavaScript 物件標記法 (JSON) 的回應緩衝區，具體取決於主題的####而定。為了清楚起見，本節中的回應和請求範例會以 JSON 格式顯示。

####	回應格式資料類型
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript 物件標記法 (JSON)

> **Important**
>
> 發佈請求訊息主題之前，請先訂閱回應主題以接收回應。此 API 使用的訊息使用 MQTT 的發佈/訂閱通訊協定，以提供請求和回應互動。
> 如果您在發佈請求之前未訂閱回應主題，則可能不會收到該請求的結果。
> IoT Core Fleet Provisioning 透過用於發佈 API 請求的相同 MQTT 連線，傳回裝置佈建 MQTT API 結果。

CreateCertificateFromCsr

從憑證簽署請求 (CSR) 建立憑證。 AWS IoT 提供由 Amazon 根憑證授權機構 (CA) 簽署的用戶端憑證。新憑證的狀態為 PENDING_ACTIVATION。當您呼叫 RegisterThing 以使用此憑證佈建實物時，憑證狀態會變更為 INACTIVE 或 ACTIVE，如範本中所述。

如需有關使用憑證授權單位憑證和憑證簽署請求建立用戶端憑證的詳細資訊，請參閱 使用您的憑證授權機構憑證建立用戶端憑證。

> **Note**
>
> 為了安全起見，由 CreateCertificateFromCsr 傳回的 certificateOwnershipToken 會在一小時後過期。必須在 certificateOwnershipToken 過期之前呼叫 RegisterThing。如果字符過期時， CreateCertificateFromCsr 建立的憑證尚未啟用並連接到政策或物件，則會刪除憑證。如果字符過期，裝置可以呼叫 CreateCertificateFromCsr 來產生新憑證。

CreateCertificateFromCsr 請求

發佈包含 $aws/certificates/create-from-csr/*payload-format* 主題的訊息。

payload-format

訊息承載格式為 cbor 或 json。

CreateCertificateFromCsr 請求承載

```
{
    "certificateSigningRequest": "string"
```

```
}
```

certificateSigningRequest

CSR，採用 PEM 格式。

CreateCertificateFromCsr 回應

訂閱 $aws/certificates/create-from-csr/*payload-format*/accepted。

payload-format

訊息承載格式為 cbor 或 json。

CreateCertificateFromCsr 回應承載

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

certificateOwnershipToken

在佈建期間證明憑證擁有權的字符。

certificateId

憑證的 ID。僅需 certificateId 即可進行憑證管理操作。

certificatePem

憑證資料 (PEM 格式)。

CreateCertificateFromCsr 錯誤

若要接收錯誤回應，請訂閱 $aws/certificates/create-from-csr/*payload-format*/rejected。

payload-format

訊息承載格式為 cbor 或 json。

CreateCertificateFromCsr 錯誤承載

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

狀態碼。

`errorCode`

錯誤代碼。

`errorMessage`

錯誤訊息。

CreateKeysAndCertificate

建立新的金鑰和憑證。 AWS IoT 提供由 Amazon 根憑證授權單位 (CA) 簽署的用戶端憑證。新憑證的狀態為 `PENDING_ACTIVATION`。當您呼叫 `RegisterThing` 以使用此憑證佈建實物時，憑證狀態會變更為 `INACTIVE` 或 `ACTIVE`，如範本中所述。

> **Note**
>
> 為了安全起見，由 `CreateKeysAndCertificate` 傳回的 `certificateOwnershipToken` 會在一小時後過期。必須在 `certificateOwnershipToken` 過期之前呼叫 `RegisterThing`。如果在字符過期時， `CreateKeysAndCertificate` 建立的憑證尚未啟用並連接到政策或物件，則會刪除憑證。如果字符過期，裝置可以呼叫 `CreateKeysAndCertificate` 來產生新憑證。

CreateKeysAndCertificate 請求

在 `$aws/certificates/create/`*`payload-format`* 上發佈訊息，其中包含空的訊息承載。

`payload-format`

訊息承載格式為 `cbor` 或 `json`。

CreateKeysAndCertificate 回應

訂閱 $aws/certificates/create/*payload-format*/accepted。

payload-format

訊息承載格式為 cbor 或 json。

CreateKeysAndCertificate 回應

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

certificateId

憑證 ID。

certificatePem

憑證資料 (PEM 格式)。

privateKey

私有金鑰。

certificateOwnershipToken

在佈建期間證明憑證擁有權的字符。

CreateKeysAndCertificate 錯誤

若要接收錯誤回應，請訂閱 $aws/certificates/create/*payload-format*/rejected。

payload-format

訊息承載格式為 cbor 或 json。

CreateKeysAndCertificate 錯誤承載

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

狀態碼。

`errorCode`

錯誤代碼。

`errorMessage`

錯誤訊息。

RegisterThing

使用預先定義的範本來佈建實物。

RegisterThing 請求

在 `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`* 上發佈訊息。

`payload-format`

訊息承載格式為 `cbor` 或 `json`。

`templateName`

佈建範本名稱。

RegisterThing 請求承載

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
```

```
        "string": "string",
        ...
    }
}
```

certificateOwnershipToken

用來證明憑證擁有權的字符。 AWS IoT 當您透過 MQTT 建立憑證時， 會產生字符。

parameters

選用。來自預先佈建掛接所使用之裝置的鍵值組，以評估註冊請求。

RegisterThing 回應

訂閱 $aws/provisioning-templates/*templateName*/provision/*payload-format*/accepted。

payload-format

訊息承載格式為 cbor 或 json。

templateName

佈建範本名稱。

RegisterThing 回應承載

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

deviceConfiguration

範本中定義的裝置組態。

thingName

佈建期間建立的 IoT 物件名稱。

RegisterThing 錯誤回應

若要接收錯誤回應，請訂閱 $aws/provisioning-templates/*templateName*/provision/*payload-format*/rejected。

payload-format

訊息承載格式為 cbor 或 json。

templateName

佈建範本名稱。

RegisterThing 錯誤回應承載

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

statusCode

狀態碼。

errorCode

錯誤代碼。

errorMessage

錯誤訊息。

機群索引

您可以使用機群索引，從以下來源索引、搜尋和彙總裝置的資料：AWS IoT 登錄檔、AWS IoT Device Shadow、AWS IoT 連線、AWS IoT 裝置管理軟體套件目錄和 AWS IoT Device Defender 違規事件。您可以查詢一組裝置，並根據裝置屬性的不同組合 (包括狀態、連線和裝置違規) 彙總裝置記錄上的統計資料。您可以利用機群索引來組織和調查裝置機群，以及解決相關問題。

機群索引提供下列功能。

管理索引更新

您可以設定機群索引，為物件群組、物件登錄檔、裝置影子、裝置連線和裝置違規的更新編制索引。啟動機群索引時， AWS IoT 會為物件或物件群組建立索引。AWS_Things 會為所有物件建立索引。AWS_ThingGroups 是包含所有物件群組的索引。啟用機群索引後，您可以在索引中執行查詢。例如，您可以找到所有手持且電池壽命超過 70% 的裝置。 會使用最新的資料持續 AWS IoT 更新索引。如需詳細資訊，請參閱管理機群索引。

查詢特定裝置的連線狀態

此 API 提供最新裝置特定連線資訊的低延遲、高輸送量存取。如需詳細資訊，請參閱裝置連線狀態。

跨資料來源搜尋

您可以建立一個基於查詢語言的查詢字串，然後將其用來跨資料來源搜尋。您也需要在機群索引設定中設定資料來源，以便索引組態包含您要從中搜尋的資料來源。查詢字串描述了您想要尋找的物件。您可以使用 AWS 受管欄位、自訂欄位和索引資料來源中的任何屬性來建立查詢。如需支援機群索引的資料來源詳細資訊，請參閱管理物件索引。

查詢彙總資料

您可以在裝置上搜尋彙總資料並傳回統計資料、百分位數、基數，或物件清單，其中包含有關特定欄位的搜尋查詢。您可以在 AWS 受管欄位或您設定為機群索引設定中自訂欄位的任何屬性上執行彙總。如需彙總查詢的詳細資訊，請參閱查詢彙總資料。

使用機群指標監控彙總資料和建立警示

您可以根據預先定義的閾值，使用機群指標自動將彙總資料傳送至 CloudWatch、分析趨勢並建立警示來監控機群的彙總狀態。如需機群指標的詳細資訊，請參閱機群指標。

管理機群索引

機群索引可以管理兩種類型的索引：物件索引和物件群組索引。

物件索引

為所有物件建立的索引為 AWS_Things。物件索引支援以下資料來源：AWS IoT 登錄檔資料、AWS IoT Device Shadow 資料、AWS IoT 連線資料以及 AWS IoT Device Defender 違規資料。透過將這些資料來源新增至機群索引組態，您可以搜尋物件、查詢彙總資料，以及根據搜尋查詢建立動態物件群組和機群指標。

登錄檔 - AWS IoT 提供登錄檔，協助您管理物件。您可以將登錄資料新增至機群索引組態，根據物件名稱、描述和其他登錄屬性來搜尋裝置。如需登錄檔的詳細資訊，請參閱如何使用登錄檔管理物件。

Shadow - AWS IoT Device Shadow 服務提供影子，協助您存放裝置狀態資料。物件索引同時支持傳統的未命名影子和命名影子。若要索引已命名影子，請啟用已命名影子的設定，然後在物件索引組態中指定影子名稱。根據預設，每個 最多可以新增 10 個陰影名稱 AWS 帳戶。若要了解如何增加影子名稱的數量限制，請參閱《AWS 一般參考》中的 AWS IoT Device Management 配額。

若要新增要索引的已命名影子：

- 如果使用的是 AWS IoT 主控台，請開啟 Thing indexing (物件索引)，選擇 Add named shadows (新增已命名影子)，再透過 Named shadow selection (已命名影子選擇) 新增影子名稱。
- 如果您使用 AWS Command Line Interface (AWS CLI)，namedShadowIndexingMode請將 設定為 ON，並在 中指定陰影名稱IndexingFilter。若要查看範例 CLI 命令，請參閱管理物件索引。

如需影子的詳細資訊，請參閱 AWS IoT Device Shadow 服務。

連線 - 裝置連線資料可協助您識別裝置的連線狀態。此連線資料是由生命週期事件驅動。當用戶端連線或中斷連線時， 會將含有訊息的生命週期事件 AWS IoT 發佈至 MQTT 主題。連線或中斷連線訊息可以是提供連線狀態詳細資訊的 JSON 元素清單，包括用戶端 ID、持續作用持續時間、清除工作階段旗標和工作階段過期。如需裝置連線的詳細資訊，請參閱生命週期事件。

Device Defender 違規 - AWS IoT Device Defender 違規資料有助於根據您在安全設定檔中定義的正常行為來識別異常裝置行為。安全性設定檔包含一組預期裝置行為。每個行為都會使用指定為裝置正常行為的指標。如需 Device Defender 違規的詳細資訊，請參閱AWS IoT Device Defender 偵測。

如需詳細資訊，請參閱管理物件索引。

物件群組索引

AWS_ThingGroups 是包含所有物件群組的索引。您可以使用此索引，根據群組名稱、描述、屬性和所有父系群組名稱搜尋群組。

如需詳細資訊，請參閱管理物件群組索引。

受管欄位

受管欄位包含與物件、物件群組、裝置影子、裝置連線能力和 Device Defender 違規相關的資料。 定義受 AWS IoT 管欄位中的資料類型。您可以在建立 AWS IoT 物件時指定每個受管欄位的值。例如，物件名稱、物件群組和物件描述都是受管欄位。機群索引會根據您指定的索引模式編製受管欄位的索引。無法在 customFields 中變更或顯示受管欄位。如需詳細資訊，請參閱自訂欄位。

下文列出物件索引的受管欄位：

- 登錄檔的受管欄位

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingTypeName, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- 傳統未命名影子的受管欄位

```
"managedFields" : [
  {name:shadow.version, type:Number},
  {name:shadow.hasDelta, type:Boolean}
]
```

- 命名影子的受管欄位

```
"managedFields" : [
```

```
  {name:shadow.name.shadowName.version, type:Number},
  {name:shadow.name.shadowName.hasDelta, type:Boolean}
]
```

- 物件連線的受管欄位

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean},
  {name:connectivity.disconnectReason, type:String},
  {name:connectivity.clientId, type:String},
  {name:connectivity.keepAliveDuration, type:Number},
  {name:connectivity.cleanSession, type:Boolean},
  {name:connectivity.sessionExpiry, type:Number}
]
```

- Device Defender 的受管欄位

```
"managedFields" : [
  {name:deviceDefender.violationCount, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.metricName, type:String},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationTime, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationValue, type:String},
  {name:deviceDefender.securityprofile.behaviorname.inViolation, type:Boolean}
]
```

- 物件群組的受管欄位

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

下表列出無法搜尋的受管欄位。

資料來源	無法搜尋的受管欄位
登錄檔	`registry.version`
未命名的影子	`shadow.version`
已命名的影子	`shadow.name.*.version`
Device Defender	`deviceDefender.version`
物件群組	`version`

自訂欄位

您可以建立自訂欄位來編製索引，可以彙總物件屬性、Device Shadow 資料和 Device Defender 違規資料。`customFields` 屬性是欄位名稱和資料類型配對的清單。您可以執行基於資料類型的彙總查詢。您選擇的索引模式會影響可以在 `customFields` 中指定的欄位。例如，如果您指定 `REGISTRY` 索引模式，則無法從物件影子中指定自訂欄位。您可以使用 update-indexing-configuration CLI 命令來建立或更新自訂欄位 (請參閱更新索引組態範例的範例命令)。

自訂欄位名稱

物件和物件群組屬性的自訂欄位名稱以 `attributes.` 開頭，後接屬性名稱。如果開啟未命名的影子索引，則物件可使用以 `shadow.desired` 或者 `shadow.reported` 開頭的自訂欄位名稱，後接未命名的影子資料值名稱。如果開啟已命名的影子索引，則物件可使用以 `shadow.name.*.desired.` 或者 `shadow.name.*.reported.` 開頭的自訂欄位名稱，後接已命名的影子資料值。如果開啟 Device Defender 違規索引，則物件可使用以 `deviceDefender.` 開頭的自訂欄位名稱，後接 Device Defender 的違規資料值。

字首後面接的屬性或資料值名稱只能使用英數字元、- (連字號) 和 _ (底線) 字元。不可有任何空格。

如果組態中的自訂欄位與正要編製索引的值之間出現類型不一致，機群索引會忽略彙總查詢的不一致值。CloudWatch Logs 在針對彙總查詢問題進行疑難排解時很有幫助。如需詳細資訊，請參閱機群索引服務的疑難排解彙總查詢。

自訂欄位類型

自訂欄位類型擁有下列支援的值：`Number`、`String`，以及 `Boolean`。

管理物件索引

為所有物件建立的索引為 AWS_Things。您可以控制要從下列資料來源編製索引的內容：AWS IoT 登錄檔資料、AWS IoT Device Shadow 資料，AWS IoT 連線資料，以及 AWS IoT Device Defender 違規資料。

在本主題中：

- 啟用物件索引
- 描述物件索引
- 查詢物件索引
- 法規與限制
- Authorization

啟用物件索引

您可以使用 update-indexing-configuration CLI 命令或 UpdateIndexingConfiguration API 操作來建立 AWS_Things 索引，並控制其組態。您可以透過使用 `--thing-indexing-configuration` (`thingIndexingConfiguration`) 參數，來控制要編製索引的資料類型 (例如，登錄檔資料、影子整理、裝置連線資料以及 Device Defender 違規資料)。

`--thing-indexing-configuration` 參數採用具有以下結構的字符：

```
{
  "thingIndexingMode": "OFF"|"REGISTRY"|"REGISTRY_AND_SHADOW",
  "thingConnectivityIndexingMode": "OFF"|"STATUS",
  "deviceDefenderIndexingMode": "OFF"|"VIOLATIONS",
  "namedShadowIndexingMode": "OFF"|"ON",
  "managedFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "customFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
```

```
    ...
  ],
  "filter": {
     "namedShadowNames": [ "string" ],
     "geoLocations": [
        {
            "name": "String",
            "order": "LonLat|LatLon"
        }
    ],
     "connectivity": {
        "includeSocketInformation": [
           "GET_THING_CONNECTIVITY_DATA"
        ]
     }
  }
}
```

物件索引模式

您可以在索引組態中指定不同的物件索引模式，視您要從中編製索引和搜尋裝置的資料來源而定：

- `thingIndexingMode`：控制登錄檔或影子是否編製索引。當 `thingIndexingMode` 設定為時`OFF`，物件索引會停用。

- `thingConnectivityIndexingMode`：指定物件連線資料是否已編製索引。必須啟用此功能，才能在 SearchIndex 查詢中取得連線狀態資訊，或使用 GetThingConnectivityData API。如果在篩選條件中啟用，且`includeSocketInformation`旗標位於 GetThingConnectivityData API 請求`true`中，GetThingConnectivityData API 回應也會包含通訊端資訊。

- `deviceDefenderIndexingMode`：指定 Device Defender 違規資料是否編製索引。
- `namedShadowIndexingMode`：指定命名陰影資料是否已編製索引。若要選取要新增至機群索引組態的已命名影子，請將 `namedShadowIndexingMode` 設為 `ON`，再在 `filter` 中指定已命名影子名稱。

下表顯示每個索引模式的有效值，以及為每個值編製索引的資料來源。

屬性	有效值	登錄檔	影子	連線能力	DD 違規	已命名影子
thingIndexingMode	OFF					
	REGISTRY	✓				
	REGISTRY_AND_SHADOW	✓	✓			
thingConnectivityIndexingMode	未指定。					
	OFF					
	STATUS			✓		
deviceDefenderIndexingMode	未指定。					
	OFF					
	VIOLATIONS				✓	
namedShadowIndexingMode	未指定。					
	OFF					
	ON					✓

受管欄位和自訂欄位

受管欄位

受管欄位包含與物件、物件群組、裝置影子、裝置連線能力和 Device Defender 違規相關的資料。 定義受 AWS IoT 管欄位中的資料類型。請在建立 AWS IoT 物件時指定每個受管欄位的值。例如，物件名稱、物件群組和物件描述都是受管欄位。機群索引會根據您指定的索引模式編製受管欄位的索引。無法在 `customFields` 中變更或顯示受管欄位。

自訂欄位

您可以建立自訂欄位來彙總屬性、Device Shadow 資料和 Device Defender 違規資料，以便為這些資料編製索引。customFields 屬性是欄位名稱和資料類型配對的清單。您可以執行基於資料類型的彙總查詢。您選擇的索引模式會影響可以在 customFields 中指定的欄位。例如，如果您指定 REGISTRY 索引模式，則無法從物件影子中指定自訂欄位。您可以使用 update-indexing-configuration CLI 命令來建立或更新自訂欄位 (請參閱更新索引組態範例的範例命令)。如需詳細資訊，請參閱自訂欄位。

索引篩選條件

索引篩選條件為具名影子、地理位置和連線資料提供額外的選擇。

namedShadowNames

若要將具名影子新增至機群索引組態，namedShadowIndexingMode請將 設定為 ，ON並在namedShadowNames篩選條件中指定具名影子名稱。

範例

```
"filter": {
     "namedShadowNames": [ "namedShadow1", "namedShadow2" ]
}
```

geoLocations

若要將地理位置資料新增至機群索引組態：

- 如果您的地理位置資料存放在傳統 （ 未命名 ） 陰影中，請將 thingIndexingMode設定為 REGISTRY_AND_SHADOW，並在geoLocations篩選條件中指定您的地理位置資料。

  以下範例篩選條件指定傳統 （ 未命名 ） 陰影中的 geoLocation 物件：

  ```
  "filter": {
       "geoLocations": [
          {
              "name": "shadow.reported.location",
              "order": "LonLat"
          }
       ]
    }
  ```

- 如果您的地理位置資料存放在具名影子中，請將 namedShadowIndexingMode設定為 ON、在namedShadowNames篩選條件中新增影子名稱，並在geoLocations篩選條件中指定您的地理位置資料。

  以下範例篩選條件指定具名影子中的 geoLocation 物件 (nameShadow1)：

```
"filter": {
     "namedShadowNames": [ "namedShadow1" ],
     "geoLocations": [
        {
            "name": "shadow.name.namedShadow1.reported.location",
            "order": "LonLat"
        }
     ]
  }
```

connectivity

- 若要在GetThingConnectivityData回應中包含通訊端資訊（來源 IP 地址、目標 IP 地址、來源連接埠、目標連接埠和 VPC 端點 ID)，請使用連線篩選條件中的 includeSocketInformation選項，如下列範例所示。呼叫 GetThingConnectivityData API true時，您也必須將 includeSocketInformation 參數設定為 。若要限制特定呼叫者存取通訊端資訊，您必須將iot:IncludeSocketInformation條件內容金鑰設定為 ，在其 IAM 政策中明確指定此項目false。

  > **Note**
  >
  > 包含通訊端資訊的選項僅支援GetThingConnectivityData回應，不支援 SearchIndex或其他機群索引 APIs。

```
"filter": {
  "connectivity": {
    "includeSocketInformation": [
      "GET_THING_CONNECTIVITY_DATA"
    ]
  }
}
```

若要停用通訊端資訊索引，您必須在連線篩選條件includeSocketInformation中將 設定為空陣列，如以下範例所示。

> **Note**
>
> 在 UpdateIndexingConfiguration 請求中，如果啟用連線索引，且未設定連線篩選條件includeSocketInformation，或未指定或為 null，則會在更新索引組態時保留連線篩選條件的先前值。

停用通訊端資訊索引的範例

```
"filter": {
  "connectivity": {
    "includeSocketInformation": [ ]
   }
  }
```

如需詳細資訊，請參閱來自 AWS IoT API 參考的 IndexingFilter。

更新索引組態範例

若要更新您的索引組態，請使用 AWS IoT CLI update-indexing-configuration 命令。下列範例示範如何使用 update-indexing-configuration。

簡短語法：

```
aws iot update-indexing-configuration --thing-indexing-configuration \
'thingIndexingMode=REGISTRY_AND_SHADOW, deviceDefenderIndexingMode=VIOLATIONS,
namedShadowIndexingMode=ON,thingConnectivityIndexingMode=STATUS,
filter={namedShadowNames=[namedShadow1],geoLocations=[{name=shadow.name.namedShadow1.reported.l
customFields=[{name=attributes.version,type=Number},
{name=shadow.name.namedShadow1.desired.DefaultDesired, type=String},
 {name=shadow.desired.power, type=Boolean},
{name=deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number,
 type=Number}]'
```

JSON 語法：

```
aws iot update-indexing-configuration --cli-input-json \ '{
          "thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY_AND_SHADOW",
          "thingConnectivityIndexingMode": "STATUS",
          "deviceDefenderIndexingMode": "VIOLATIONS",
          "namedShadowIndexingMode": "ON",
          "filter": { "namedShadowNames": ["namedShadow1"], "geoLocations": [{"name":
 "shadow.name.namedShadow1.reported.location", "order": "LonLat"}], "connectivity":
 {"includeSocketInformation": ["GET_THING_CONNECTIVITY_DATA"]}},
          "customFields": [ { "name": "shadow.desired.power", "type": "Boolean" },
          {"name": "attributes.version", "type": "Number"},
          {"name": "shadow.name.namedShadow1.desired.DefaultDesired", "type":
 "String"},
          {"name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
 "type": "Number"} ] } }'
```

此命令不會產生任何輸出。

若要檢查物件索引狀態，請執行 describe-index CLI 命令：

```
aws iot describe-index --index-name "AWS_Things"
```

describe-index 命令的輸出如下所示：

```
{
    "indexName": "AWS_Things",
    "indexStatus": "ACTIVE",
    "schema": "MULTI_INDEXING_MODE"
}
```

> **Note**
>
> 機群索引可能需要一些時間才能更新機群索引。建議等到 `indexStatus` 顯示 ACTIVE (啟用) 之後才使用。結構描述欄位可以擁有不同的值，視已設定的資料來源而定。如需詳細資訊，請參閱描述物件索引。

若要獲取索引組態的詳細資訊，請執行 get-indexing-configuration CLI 命令：

```
aws iot get-indexing-configuration
```

get-indexing-configuration 命令的輸出如下所示：

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "thingConnectivityIndexingMode": "STATUS",
        "deviceDefenderIndexingMode": "VIOLATIONS",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "connectivity.keepAliveDuration",
                "type": "Number"
            },
            {
                "name": "connectivity.version",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.metricName",
                "type": "String"
            },
            {
                "name": "connectivity.connected",
                "type": "Boolean"
            },
            {
                "name": "shadow.version",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.lastViolationTime",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.inViolation",
                "type": "Boolean"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "deviceDefender.version",
                "type": "Number"
```

```
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "connectivity.sessionExpiry",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "deviceDefender.violationCount",
                "type": "Number"
            },
            {
                "name": "connectivity.timestamp",
                "type": "Number"
            },
            {
                "name": "connectivity.clientId",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
            },
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "shadow.hasDelta",
                "type": "Boolean"
```

```
            },
            {
                "name": "connectivity.cleanSession",
                "type": "Boolean"
            },
            {
                "name": "connectivity.disconnectReason",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "attributes.version",
                "type": "Number"
            },
            {
                "name": "shadow.desired.power",
                "type": "Boolean"
            },
            {
                "name": "shadow.name.namedShadow1.desired.DefaultDesired",
                "type": "String"
            },
            {
                "name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "namedShadow1"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.namedShadow1.reported.location",
                    "order": "LonLat"
                }
            ],
            "connectivity": {
                "includeSocketInformation": [
                    "GET_THING_CONNECTIVITY_DATA"
                ]
            }
```

```
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

若要更新自訂欄位，您可以執行 update-indexing-configuration 命令。範例如下：

```
aws iot update-indexing-configuration --thing-indexing-configuration

 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number},
{name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean},
{name=shadow.desired.intensity,type=Number}]'
```

此命令已將 `shadow.desired.intensity` 新增至索引組態。

> **Note**
>
> 更新索引組態中的自訂欄位會覆寫所有現有的自訂欄位。請務必在呼叫 update-indexing-configuration 時指定所有自訂欄位。

在重建索引之後，您可以在剛新增的欄位上使用彙總查詢，然後搜尋登錄檔資料、影子資料和物件連線狀態資料。

變更索引模式時，請使用新的索引模式確定所有自訂欄位都有效。例如，如果以自訂欄位名為 `shadow.desired.temperature` 的 `REGISTRY_AND_SHADOW` 模式開始，則必須先刪除 `shadow.desired.temperature` 自訂欄位，再將索引模式變更為 `REGISTRY`。如果索引組態包含未被索引模式編製索引的自訂欄位，更新就會失敗。

描述物件索引

以下命令說明如何使用 describe-index CLI 命令，擷取物件索引的目前狀態。

```
aws iot describe-index --index-name "AWS_Things"
```

命令的回應可能如下所示：

```
{
    "indexName": "AWS_Things",
```

```
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

第一次建立機群索引時， 會 AWS IoT 建立您的索引。indexStatus 為 BUILDING 狀態時，便法在索引內進行查詢。物件索引的 schema 會指出系統會為何種類型的資料 (REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS) 建立索引。

若您變更索引的組態，將導致索引重新建立。在此程序期間的 indexStatus 為 REBUILDING。您可以在重建物件索引時，對物件索引內的資料執行查詢。例如，若您將索引組態從 REGISTRY 變更為 REGISTRY_AND_SHADOW，重建索引時，您可以查詢登錄檔資料 (包括最新的更新)。不過，您無法查詢影子資料，除非重建完成。建立或重建索引所需時間，取決於資料量。

結構描述欄位可以顯示不同的值，視已設定的資料來源而定。下表顯示了不同的結構描述值和相應描述：

結構描述	說明
OFF	未設定任何資料來源或為其編制索引。
REGISTRY	登錄檔資料會編製索引。
REGISTRY_AND_SHADOW	登錄檔資料和未命名 (傳統) 的影子資料都會編製索引。
REGISTRY_AND_CONNECTIVITY	登錄檔資料和連線資料都會編製索引。
REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS	登錄檔資料、未命名 (傳統) 影子資料和連線資料都會編製索引。
MULTI_INDEXING_MODE	除了登錄檔、未命名 (傳統) 影子或連線資料之外，還會編製已命名影子或 Device Defender 違規資料的索引。

查詢物件索引

使用 search-index CLI 命令來查詢索引中的資料。

```
aws iot search-index --index-name "AWS_Things" --query-string
```

```
          "thingName:mything*"
```

```
{
  "things": [
    {
      "thingName": "mything1",
      "thingGroupNames": ["mygroup1"],
      "thingId": "a4b9f759-b0f2-4857-8a4b-967745ed9f4e",
      "attributes": {
        "attribute1": "abc"
      },
      "connectivity": {
        "connected": false,
        "timestamp": 1556649874716,
        "disconnectReason": "CONNECTION_LOST",
        "clientId": "mything1",
        "keepAliveDuration": 60,
        "cleanSession": false,
        "sessionExpiry": 3600
      }
    },
    {
      "thingName": "mything2",
      "thingTypeName": "MyThingType",
      "thingGroupNames": ["mygroup1", "mygroup2"],
      "thingId": "01014ef9-e97e-44c6-985a-d0b06924f2af",
      "attributes": {
        "model": "1.2",
        "country": "usa"
      },
      "shadow": {
        "desired": {
          "location": "new york",
          "myvalues": [3, 4, 5]
        },
        "reported": {
          "location": "new york",
          "myvalues": [1, 2, 3],
          "stats": {
            "battery": 78
          }
        },
        "metadata": {
```

```
          "desired": {
            "location": { "timestamp": 123456789 },
            "myvalues": { "timestamp": 123456789 }
          },
          "reported": {
            "location": { "timestamp": 34535454 },
            "myvalues": { "timestamp": 34535454 },
            "stats": {
              "battery": { "timestamp": 34535454 }
            }
          }
        },
        "version": 10,
        "timestamp": 34535454
      },
      "connectivity": {
        "connected": true,
        "timestamp": 1556649855046,
        "clientId": "mything2",
        "keepAliveDuration": 60,
        "cleanSession": true
      }
    }
  ],
  "nextToken": "AQFCuvk7zZ3D9pOYMbFCeHbdZ+h=G"
}
```

在 JSON 回應中，"connectivity"（如 thingConnectivityIndexingMode=STATUS設定所啟用）會提供裝置的連線狀態、時間戳記、中斷連線原因、用戶端 ID、持續作用持續時間、清除工作階段旗標和工作階段過期。因為 CONNECTION_LOST，裝置 "mything1" 在 POSIX 時間 1556649874716 中斷連線 (false)。如需中斷連線原因的詳細資訊，請參閱生命週期事件。

```
"connectivity": {
  "connected": false,
  "timestamp": 1556649874716,
  "disconnectReason": "CONNECTION_LOST",
  "clientId": "mything1",
  "keepAliveDuration": 60,
  "cleanSession": false,
  "sessionExpiry": 3600
}
```

裝置 "mything2" 在 POSIX 時間 1556649855046 連線 (true)：

```
"connectivity": {
  "connected": true,
  "timestamp": 1556649855046,
  "clientId": "mything2",
  "keepAliveDuration": 60,
  "cleanSession": true
}
```

時間戳記是以從 epoch 算起的毫秒提供，因此 1556649855046 代表 2019 年 4 月 30 日星期二下午 6:44:15.046 (UTC)。

> **Important**
>
> 如果裝置從未連接到 IoT Core，或在thingConnectivityIndexingMode啟用機群索引之前中斷連線超過 1 小時，回應會將 connected 欄位設定為 false，而沒有其他工作階段詳細資訊。

法規與限制

這些是 AWS_Things 的限制。

類型複雜的影子欄位

只有當欄位的值為簡單類型，像是不包含陣列的 JSON 物件，或完全由簡單類型組成的陣列時，才會建立該影子欄位的索引。簡單類型是指字串、數值或下列其中一個常值：true 或 false。以下列影子狀態為例，欄位 "palette" 的值不會編製索引，因為它是包含複雜類型項目的陣列。而因為該陣列中的每個值皆為字串，所以 "colors" 欄位的值會建立索引。

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
```

```
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
    }
}
```

巢狀影子欄位名稱

巢狀影子欄位的名稱會儲存為以句號 (.) 分隔的字串。例如，假設影子文件為：

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

three 欄位的名稱會儲存為 desired.one.two.three。如果您也有影子文件，會如此儲存：

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

同時符合 shadow.desired.one.two.three:v2 的查詢。最佳實務是不要在影子欄位名稱中使用句點。

影子中繼資料

只有當建立影子的 "state" 區段中之對應欄位的索引時，才會建立影子中繼資料區段中之欄位的索引。(在前面的範例中，也不會建立影子中繼資料區段中 "palette" 欄位的索引。)

未註冊的裝置

機群索引不會為未註冊裝置的資料編製索引。實物必須在 AWS IoT 登錄檔中註冊，且必須在連線至 clientId時使用實物名稱做為 AWS IoT Core。

未登錄檔的影子

如果您使用 UpdateThingShadow，使用尚未在 AWS IoT 帳戶中註冊的物件名稱來建立影子，則此影子中的欄位不會編製索引。這適用於傳統的未命名影子和已命名影子。

數值

如果服務將任何登錄檔或影子資料辨識為數值，則會因其為數值而建立索引。您可以對數值運用包含範圍與比較運算子的查詢 (例如 "attribute.foo<5" 或 "shadow.reported.foo:[75 TO 80]")。若要識別為數字，資料的值必須是有效的文字類型 JSON 數字。數值可為介於範圍 -2^53...2^53-1 之間的整數、使用選擇性指數表示法的雙精度浮點，或必須屬於僅包含這些值的陣列。

Null 值

Null 值不編製索引。

最大值

彙總查詢的自訂欄位數量上限是 5。

彙總查詢所要求的百分位數上限是 100。

Authorization

您可以在 AWS IoT 政策動作中將物件索引或物件 （適用於 GetThingConnectivityData 動作） 指定為 Amazon Resource Name (ARN)，如下所示。

Action	資源
iot:SearchIndex	索引 ARN (例如，arn:aws:iot: *your-aws-region* :*your-aws-account* :index/AWS_Things)。

Action	資源
iot:DescribeIndex	索引 ARN (例如，arn:aws:iot: *your-aws-region* :*your-aws-account* :index/AWS_Things)。
iot:GetThingConnectivityData	您可以將此許可範圍限定為特定物件。使用 iot:IncludeSocketInformation 條件金鑰來控制對通訊端資訊的存取。

> **Note**
>
> 如果您有查詢機群索引的許可，您就可以存取整個機群的物件資料。

管理物件群組索引

AWS_ThingGroups 是包含所有物件群組的索引。您可以使用此索引，根據群組名稱、描述、屬性和所有父系群組名稱搜尋群組。

啟用物件群組索引

您可以使用 UpdateIndexingConfiguration API 中的 thing-group-indexing-configuration 設定，來建立 AWS_ThingGroups 索引並控制其組態。您可以使用 GetIndexingConfiguration API，擷取目前的索引組態。

若要更新物件群組索引的組態，請執行 update-indexing-configuration CLI 命令：

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

您也能以如下命令更新物件和物件群組索引的組態：

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

以下是 thingGroupIndexingMode 的有效值。

OFF

不建立索引/刪除索引。

ON

建立或設定 AWS_ThingGroups 索引。

若要擷取目前物件和物件群組索引組態，請執行 get-indexing-configuration CLI 命令：

```
aws iot get-indexing-configuration
```

命令的回應如下所示：

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

描述群組索引

若要擷取 describe-index 索引的目前狀態，請使用 AWS_ThingGroups CLI 命令：

```
aws iot describe-index --index-name "AWS_ThingGroups"
```

命令的回應如下所示：

```
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT 會在您第一次編製索引時建置索引。若 indexStatus 為 BUILDING，您無法在該索引內進行查詢。

查詢物件群組索引

若要查詢索引中的資料，請使用 search-index CLI 命令：

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupName:mythinggroup*"
```

Authorization

您可以在 AWS IoT 政策動作中將物件群組索引指定為資源 ARN，如下所示。

Action	資源
iot:SearchIndex	索引 ARN (例如，arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups)。
iot:DescribeIndex	索引 ARN (例如，arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups)。

裝置連線狀態查詢

AWS IoT 機群索引支援個別裝置連線查詢，可讓您有效率地擷取特定裝置的連線狀態和相關中繼資料。此功能可補充現有的機群整體索引和查詢功能。

運作方式

裝置連線查詢支援可用於最佳化的單一裝置連線狀態擷取。此 API 提供最新裝置特定連線資訊的低延遲、高輸送量存取。啟用連線索引後，您將可以存取此查詢 API，並以標準查詢收費。如需詳細資訊，請參閱 AWS IoT Device Management 定價

功能

透過裝置連線查詢支援，您可以：

1. 使用 查詢指定裝置的目前連線狀態 （ 已連線或已中斷連線 ）thingName。
2. 擷取其他連線中繼資料，包括：
   a. 中斷連線原因
   b. 最新連線或中斷連線事件的時間戳記。
   c. 工作階段資訊，包括持續作用持續時間
   d. 通訊端層級工作階段資訊，包括 IP 地址、連接埠和 VPC 端點 ID。只有在符合下列條件時，才能使用此資訊：

i. 您可以在機群索引組態設定中啟用此選項。

ii. 叫用 API 時，您在 IAM 政策中具有對應的許可。

> **Note**
>
> 機群索引會為裝置的連線狀態建立索引，其連線 `clientId` 與登錄中已註冊物件的 `thingName` 相同。

優勢

1. 低延遲：反映最新的裝置連線狀態，並提供低延遲，以反映來自 IoT Core 的連線狀態變更。IoT Core 會在收到來自裝置的中斷連線請求時，或在裝置中斷連線但未傳送中斷連線請求的情況下，立即判斷裝置是否中斷連線。IoT 核心會等待 1.5 倍的已設定持續作用時間，然後再判斷用戶端是否中斷連線。連線狀態 API 通常會在 IoT Core 判斷裝置的連線狀態變更後一秒內反映這些變更。
2. 高輸送量：預設支援每秒 350 筆交易 (TPS)，並可依請求調整至更高。
3. 資料保留：啟用機群索引 (FI) ConnectivityIndexing 模式且未刪除物件時， 會無限期儲存事件資料。如果您停用連線索引，則不會保留記錄。

> **Note**
>
> 如果在啟動此 API 之前啟用連線狀態索引，機群索引會在 API 啟動後開始追蹤連線狀態變更，並根據這些變更反映更新的狀態。

先決條件

若要使用裝置連線查詢支援：

1. 設定 AWS 帳戶
2. 在您 AWS IoT Core 偏好的區域中，將裝置加入並註冊至
3. 使用連線索引啟用機群索引。選用：透過啟用 `includeSocketInformation`選項選擇接收通訊端層級資訊。

> **Note**
>
> 如果您已啟用連線索引，則不需要額外的設定。

如需詳細設定說明，請參閱 AWS IoT 開發人員指南

範例

CLI 命令：

```
aws iot get-thing-connectivity-data --include-socket-information --thing-name
 myThingName
```

回應：

```
{
   "thingName": "myThingName",
   "connected": true,
   "timestamp": "2024-12-19T10:00:00.000000-08:00",
   "disconnectReason": "NONE",
   "sourceIp": "192.0.2.1",
   "sourcePort": 52123,
   "targetIp": "198.51.100.1",
   "targetPort": 8883,
   "vpcEndpointId": "vpce-1234567890abcdef0",
   "keepAliveDuration": 60,
   "cleanSession": true,
   "clientId": "myThingName"
}
```

API 參數

- `thingName`：在 AWS IoT 登錄檔中註冊的裝置名稱。這必須符合`clientId`用於連線至 AWS IoT Core 的 。
- `includeSocketInformation`： `includeSocketInformation` 參數控制通訊端層級網路資訊是否包含在 API 回應中。設定為 true 時，回應會包含下列欄位：`sourceIp`, `sourcePort`, `targetIp`, `targetPort`, `vpcEndpointId`。當 `includeSocketInformation` 未指定或設定為 false 時，這些通訊端欄位會從回應中排除。若要限制特定 IAM 使用者存取通訊端資訊，您需要將`includeSocketInformation`條件索引鍵設定為 false，以在 IAM 政策中指定此資訊

回應欄位

- thingName：在 AWS IoT 登錄檔中註冊的裝置名稱。這必須符合clientId用於連線至 AWS IoT Core 的 。
- connected：布林值 true 表示此裝置目前已連線。
- disconnectReason：中斷連線的原因。將NONE適用於已連線的裝置，以及從未連線UNKNOWN的裝置。對於中斷連線的裝置，這將指出中斷連線是用戶端啟動、伺服器啟動、身分驗證/授權問題還是網路問題。如需中斷連線原因代碼，請參閱 LifeCycleEvents
- timestamp：代表裝置最近連線或中斷連線事件的時間戳記。
- clientId：MQTT 用戶端的 clientId。
- keepAliveDuration：用戶端在建立連線時指定的保持連線間隔，以秒為單位。這會決定用戶端傳送保持連線訊息以維持連線的頻率。
- cleanSession：指出用戶端是否使用乾淨的工作階段。
- sessionExpiry：用戶端在建立連線時指定的持久性工作階段到期組態。這會決定工作階段在用戶端中斷連線後會保持作用中狀態的時間。
- sourceIp：啟動連線之用戶端的 IP 地址。只有在 includeSocketInformation 設定為 true 且使用者獲授權擷取此資訊時，才會傳回 。
- sourcePort：用戶端用於連線的連接埠號碼。只有在 includeSocketInformation 設定為 true 且使用者獲授權擷取此資訊時，才會傳回 。
- targetIp：發出連線請求的 IP 地址。只有在 includeSocketInformation 設定為 true 且使用者有權擷取此資訊時才傳回
- targetPort：用戶端連線的 AWS IoT Core 端點連接埠號碼。只有在 includeSocketInformation 設定為 true 且使用者有權擷取此資訊時才傳回。
- vpcEndpointId：用戶端連線的 VPC 端點 ID，如適用。只有在 includeSocketInformation 設定為 true 且使用者獲授權擷取此資訊時，才會傳回 。

必要許可

若要使用 GetThingConnectivityData API，您需要下列 IAM 許可

iot:GetThingConnectivityData

您可以使用資源型政策，將此許可範圍限定為特定物件。使用 iot:IncludeSocketInformation條件對通訊端資訊實作精細存取控制。以下政策範例示範您拒絕

使用者存取通訊端資訊的情況。請注意，此範例適用於授予 GetThingConnectivityData 動作存取權的唯一政策。

授權政策範例

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Action": [ "iot:GetThingConnectivityData" ],
    "Resource": [ "arn:aws:iot:us-east-1:123456789012:thing/*"],
    "Condition": {
      "Bool": {
        "iot:IncludeSocketInformation": "false"
      }
    }
  }]
}
```

查詢彙總資料

AWS IoT 提供四個 APIs (`GetStatistics`、`GetPercentiles`、 `GetCardinality`和 `GetBucketsAggregation`)，可讓您搜尋裝置機群以取得彙總資料。

> **Note**
>
> 如需彙總 API 遺失或未預期值的問題，請閱讀機群索引故障診斷指南。

GetStatistics

GetStatistics API 和 get-statistics CLI 命令會針對指定的彙總欄位傳回計數、平均值、總和、最小值、最大值、平方和、方差和標準差。

get-statistics CLI 命令會使用下列參數：

`index-name`

要搜尋的索引名稱。預設值為 `AWS_Things`。

`query-string`

用於搜尋索引的查詢。您可以指定 "*" 來取得 中所有索引物件的計數 AWS 帳戶。

`aggregationField`

(選用) 要彙總的欄位。此欄位必須是您在呼叫 update-indexing-configuration 時定義的受管或自訂欄位。如果您未指定彙總欄位，則 `registry.version` 會用作彙總欄位。

`query-version`

要使用的查詢版本。預設值為 `2017-09-30`。

彙總欄位的類型可以影響傳回的統計資料。

搭配字串值的 GetStatistics

如果您在字串欄位上進行彙總，則呼叫 `GetStatistics` 會傳回多少裝置具有符合查詢的屬性。例如：

```
aws iot get-statistics --aggregation-field 'attributes.stringAttribute'
            --query-string '*'
```

此命令會傳回多少裝置包含稱為 `stringAttribute` 的屬性：

```
{
  "statistics": {
    "count": 3
  }
}
```

搭配布林值的 GetStatistics

當您`GetStatistics`使用布林值彙總欄位呼叫 時：

- AVERAGE 是符合查詢的裝置百分比。
- 根據以下規則，MINIMUM 為 0 或 1：
  - 如果彙總欄位的所有值是 `false`，MINIMUM 為 0。
  - 如果彙總欄位的所有值是 `true`，MINIMUM 為 1。
  - 如果彙總欄位的值是 `false` 和 `true` 的混合，MINIMUM 為 0。
- 根據以下規則，MAXIMUM 為 0 或 1：

- 如果彙總欄位的所有值是 `false`，MAXIMUM 為 0。
- 如果彙總欄位的所有值是 `true`，MAXIMUM 為 1。
- 如果彙總欄位的值是 `false` 和 `true` 的混合，則 MAXIMUM 為 1。

- SUM 是相當於布林值的整數總和。
- COUNT 是符合查詢字串準則並包含有效彙總欄位值的物件計數。

搭配數值的 GetStatistics

當您呼叫 `GetStatistics` 並指定類型為 `Number` 的彙總欄位時，`GetStatistics` 會傳回下列值：

count

符合查詢字串條件且包含有效彙總欄位值的物件計數。

average

符合查詢之數值的平均值。

sum

符合查詢之數值的總和。

minimum

符合查詢的最小數值。

maximum

符合查詢的最大數值。

sumOfSquares

符合查詢之數值的平方和。

variance

符合查詢之數值的方差。一組值的方差是每個值與集合平均值之平方差的平均值。

stdDeviation

符合查詢之數值的標準差。一組值的標準差是值如何分佈的度量。

以下範例展示如何使用數字自訂欄位呼叫 get-statistics。

```
aws iot get-statistics --aggregation-field 'attributes.numericAttribute2'
```

```
            --query-string '*'
```

```
{
  "statistics": {
    "count": 3,
    "average": 33.333333333333336,
    "sum": 100.0,
    "minimum": -125.0,
    "maximum": 150.0,
    "sumOfSquares": 43750.0,
    "variance": 13472.22222222222,
    "stdDeviation": 116.06990230986766
  }
}
```

對於數值彙總欄位，如果欄位值超過最大雙精度值，則統計資料為空白

GetCardinality

GetCardinality API 和 get-cardinality CLI 命令會傳回符合查詢之唯一值的近似計數。例如，您可能想要尋找電池電量低於 50% 的裝置數量：

```
aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel
          > 50" --aggregation-field "shadow.reported.batterylevel"
```

這個命令會傳回電池電量超過 50% 的物件數量：

```
{
    "cardinality": 100
}
```

`cardinality` 一律由 get-cardinality 傳回，即使沒有相符的欄位。例如：

```
aws iot get-cardinality --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_STR"
```

```
{
    "cardinality": 0
}
```

get-cardinality CLI 命令會使用下列參數：

`index-name`

要搜尋的索引名稱。預設值為 `AWS_Things`。

`query-string`

用於搜尋索引的查詢。您可以指定 "*" 來取得 中所有索引物件的計數 AWS 帳戶。

`aggregationField`

要彙總的欄位。

`query-version`

要使用的查詢版本。預設值為 `2017-09-30`。

GetPercentiles

GetPercentiles API 和 get-percentiles CLI 命令會將符合查詢的彙總值分組為百分位數群組。雖然預設的百分位數群組是：1、5、25、50、75、95、99，但是您可以在呼叫 `GetPercentiles` 時指定自己的百分位數群組。此函數會傳回所指定之每個百分位數群組 (預設百分位數群組) 的值。百分位數群組 "1" 包含的彙總欄位值，大約發生在符合查詢的百分之一值。百分位數群組 "5" 包含的彙總欄位值，大約發生在符合查詢的百分之五值，依此類推。結果是一個近似值，符合查詢的值越多，百分位數值就越精確。

以下範例顯示如何呼叫 get-percentiles CLI 命令。

```
aws iot get-percentiles --query-string "thingName:*" --aggregation-field
          "attributes.customField_NUM" --percents 10 20 30 40 50 60 70 80 90 99
```

```
{
    "percentiles": [
        {
            "value": 3.0,
            "percent": 80.0
        },
        {
            "value": 2.5999999999999996,
            "percent": 70.0
        },
        {
```

```
            "value": 3.0,
            "percent": 90.0
        },
        {
            "value": 2.0,
            "percent": 50.0
        },
        {
            "value": 2.0,
            "percent": 60.0
        },
        {
            "value": 1.0,
            "percent": 10.0
        },
        {
            "value": 2.0,
            "percent": 40.0
        },
        {
            "value": 1.0,
            "percent": 20.0
        },
        {
            "value": 1.4,
            "percent": 30.0
        },
        {
            "value": 3.0,
            "percent": 99.0
        }
    ]
}
```

以下命令顯示在沒有相符文件時從 get-percentiles 傳回的輸出。

```
aws iot get-percentiles --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_NUM"
```

```
{
    "percentiles": []
}
```

get-percentile CLI 命令會使用下列參數：

`index-name`

要搜尋的索引名稱。預設值為 `AWS_Things`。

`query-string`

用於搜尋索引的查詢。您可以指定 "*" 來取得 中所有索引物件的計數 AWS 帳戶。

`aggregationField`

要彙總的欄位，其類型必須是 `Number`。

`query-version`

要使用的查詢版本。預設值為 `2017-09-30`。

`percents`

(選用) 您可以使用此參數來指定自訂百分位數群組。

GetBucketsAggregation

GetBucketsAggregation API 和 get-buckets-aggregation CLI 命令會傳回儲存貯體清單，以及符合查詢字串準則的物件總數。

以下範例顯示如何呼叫 get-buckets-aggregation CLI 命令。

```
aws iot get-buckets-aggregation --query-string '*' --index-name AWS_Things --
aggregation-field 'shadow.reported.batterylevelpercent' --buckets-aggregation-type
 'termsAggregation={maxBuckets=5}'
```

此命令會傳回下列：

```
{
    "totalCount": 20,
    "buckets": [
        {
            "keyValue": "100",
            "count": 12
        },
        {
            "keyValue": "90",
```

```
            "count": 5
        },
        {
            "keyValue": "75",
            "count": 3
        }
    ]
}
```

get-buckets-aggregation CLI 命令會使用下列參數：

`index-name`

要搜尋的索引名稱。預設值為 AWS_Things。

`query-string`

用於搜尋索引的查詢。您可以指定 "*" 來取得 中所有索引物件的計數 AWS 帳戶。

`aggregation-field`

要彙總的欄位。

`buckets-aggregation-type`

回應形狀的基本控制和要執行的儲存貯體彙總類型。

Authorization

您可以在 AWS IoT 政策動作中將物件群組索引指定為資源 ARN，如下所示。

Action	資源
`iot:GetStatistics`	索引 ARN (例如，`arn:aws:iot:` *`your-aws-region`* `:index/AWS_Things` 或 `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`)。

查詢語法

可以在機群索引中使用查詢語法來指定查詢。

支援的功能

查詢語法支援以下功能：

- 詞彙和字詞
- 搜尋欄位
- 字首搜尋
- 範圍搜尋
- 布林運算子 AND、OR、NOT 及 -。連字號是用於從搜尋結果排除某些項目 (例如，`thingName:(tv* AND -plasma)`)。
- 分組
- 欄位分組
- 逸出特殊字元 (如同以 \)
- 主要萬用字元使用限制為每個查詢 1 個查詢詞彙。例如，您無法在相同的查詢`thingGroupNames:*echo`中搜尋 `thingName:*my`和 。包含前置萬用字元的查詢，查詢長度上限為 100 個字元。

> **Note**
>
> 使用前置萬用字元查詢詞彙時，機群索引查詢 API 限制超過每秒 30 個請求的帳戶會受到限流限制。此requests-per-second的限制適用於所有機群索引查詢 APIs 的帳戶層級。

不支援的功能

查詢語法不支援以下功能：

- 常規表達式
- 提升
- 排名
- 模糊搜尋
- 鄰近搜尋
- 排序
- 聚合

- 特殊字元：`、@、#、%、\、/、'、;、和 ,。請注意，,僅支援地理查詢。

備註

查詢語言的幾件注意事項：

- 預設運算子為 AND。對 "thingName:abc thingType:xyz" 的查詢等同於 "thingName:abc AND thingType:xyz"。
- 如果未指定欄位，AWS IoT 則會在所有登錄檔、Device Shadow 和 Device Defender 欄位中搜尋該詞彙。
- 所有欄位名稱皆會區分大小寫。
- 搜尋不區分大小寫。文字以 Java Character.isWhitespace(int) 定義的空白字元分隔。
- Device Shadow 資料 (未命名的影子和已命名的影子) 的索引包含已呈報、所需、差量和中繼資料部分。
- 無法搜尋裝置影子和登錄檔版本，但回應中會顯示這些資訊。
- 一條查詢的詞彙數量上限為 12。
- , 僅在地理查詢中支援特殊字元。

範例物件查詢

使用查詢語法指定查詢字串中的查詢。查詢會傳遞至 SearchIndex API。下表列出部分查詢字串範例。

查詢字串	結果
abc	在任何註冊表，影子 (經典未命名的影子和命名的影子) 或 Device Defender 違規欄位中的「abc」查詢。
thingName:myThingName	可查詢名為 "myThingName" 的物件。
thingName:my*	可查詢名稱起始為 "my" 的物件。
thingName:*my	查詢名稱結尾為 "my" 的物件。主要萬用字元使用限制為每個查詢 1 個查詢詞彙。包含前置萬用字元的查詢，查詢長度上限為 100 個字元。

查詢字串	結果
`thingName:*my*`	查詢包含「my」子字串的物件。主要萬用字元使用限制為每個查詢 1 個查詢詞彙。包含前置萬用字元的查詢，查詢長度上限為 100 個字元。
`thingName:ab?`	可查詢具有「ab」加上一個額外字元的物件名稱 (例如，「aba」、「abb」、「abc」等)。
`thingTypeName:aa`	查詢與類型 "aa" 關聯的物件。
`thingGroupNames:a`	查詢具有父物件群組或帳單群組名稱 "a" 的物件。
`thingGroupNames:a*`	查詢父物件群組或帳單群組名稱符合模式 "a*" 的物件。
`attributes.myAttribute:75`	可查詢具有 "myAttribute" 屬性名稱的物件，且其值為 75。
`attributes.myAttribute: [75 TO 80]`	可查詢具有 "myAttribute" 屬性名稱的物件，且其值介於數字範圍 (75–80，含) 內。
`attributes.myAttribute: {75 TO 80]`	可查詢具有 "myAttribute" 屬性名稱的物件，且其值介於數字範圍 (>75 且 <=80) 內。
`attributes.serialNumber: ["abcd" TO "abcf"]`	可查詢具有 "serialNumber" 屬性名稱的物件，且其值介於一組字母和數字的字串範圍內。以此查詢而言，傳回的物件必須具有 "serialNumber" 屬性且其值為 "abcd"、"abce" 或 "abcf"。
`attributes.myAttri bute:i*t`	可查詢具有 "myAttribute" 屬性名稱的物件，且其值以 "i" 為首、後方接任意數量的字元，最後則是 "t"。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	使用布林值運算式，查詢結合詞彙的物件。以此查詢而言，傳回的物件必須具有「attr1」屬性名稱且其值為「abc」具有「attr2」屬性名稱且其值小於 5 以及具有「attr3」屬性名稱且其值不超過 10。
`shadow.hasDelta:true`	查詢具有差量元素的未命名影子物件。
`NOT attributes.model:l egacy`	可查詢屬性名稱為 "model" 而不是 "legacy" 的物件。

查詢字串	結果
`shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy`	可查詢符合下列標準的物件： • 物件影子 `stats.battery` 屬性值在 70 至 100 之間。 • 物件名稱、類型名稱或屬性值出現文字 "v2" 或 "v3"。 • 物件 `model` 屬性未設定為 "legacy"。
`shadow.reported.myvalues:2`	查詢影子回報區段中之 `myvalues` 陣列包含值為 2 的物件。
`shadow.reported.location:* NOT shadow.desired.stats.battery:*`	可查詢符合下列標準的物件： • `location` 屬性存在於影子的 `reported` 區段中。 • `stats.battery` 屬性不存在於影子的 `desired` 區段中。
`shadow.name.<shadowName>.hasDelta:true`	查詢具有給定名稱的影子和差量元素的物件。
`shadow.name.<shadowName>.desired.filament:*`	查詢具有給定名稱的影子和所需絲狀結構屬性的物件。
`shadow.name.<shadowName>.reported.location:*`	查詢具有給定名稱影子的事物以及已命名影子報告區段中有 `location` 屬性存在的物件。
`connectivity.connected:true`	查詢所有已連線的裝置。
`connectivity.connected:false`	查詢所有中斷連線的裝置。
`connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	查詢連線時間戳記 >= 1557651600000 且 <= 1557867600000 的所有連網裝置。時間戳記是以從 Epoch 算起的毫秒提供。

查詢字串	結果
`connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	查詢中斷連線時間戳記 >= 1557651600000 且 <= 1557867600000 的所有中斷連網裝置。時間戳記是以從 Epoch 算起的毫秒提供。
`connectivity.connected:true AND connectivity.timestamp > 1557651600000`	查詢連線時間戳記 > 1557651600000 的所有連網裝置。時間戳記是以從 Epoch 算起的毫秒提供。
`connectivity.connected:*`	查詢具連線資訊的所有裝置。
`connectivity.disconnectReason:*`	查詢具有連線 disconnectReason 的所有裝置。
`connectivity.disconnectReason:CLIENT_INITIATED_DISCONNECT`	查詢因 CLIENT_INITIATED_DISCONNECT 而中斷連線的所有裝置。
`deviceDefender.violationCount:[0 TO 100]`	查詢 Device Defender 違規計數值落在數字範圍 (0-100，含) 內的物件。
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.inViolation:true`	查詢違反如安全性設定檔 `device-SecurityProfile` 所定義行為 `disconnectBehavior` 的物件。請注意，inViolation:false 不是有效的查詢。
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationValue.number>2`	查詢違反如安全性設定檔 device-SecurityProfile 中所定義的 `disconnectBehavior` 行為，且最後一個違規事件值大於 2。

查詢字串	結果
deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationTime>1634227200000	查詢違反如安全性設定檔 device-SecurityProfile 中所定義的 disconnectBehavior 行為，且在指定的 epoch 時間後發生最後一個違規事件。
shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	查詢距離座標 47.6204，-122.3491 15.5 km 範圍內的物件。當您的位置資料存放在具名影子中時，此查詢字串適用於 。
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	查詢距離座標 47.6204，-122.3491 15.5 km 範圍內的物件。當您的位置資料存放在傳統影子中時，此查詢字串適用於 。

範例物件群組查詢

查詢是以使用查詢語法的字串指定，並傳送至 SearchIndex API。下表列出部分查詢字串範例。

查詢字串	結果
abc	任何欄位的 "abc" 查詢。
thingGroupName:myGroupThingName	可查詢名為 "myGroupThingName" 的物件群組。
thingGroupName:my*	可查詢名稱以 "my" 開頭的物件群組。
thingGroupName:ab?	可查詢具有「ab」加上一個額外字元的物件群組名稱 (例如：「aba」、「abb」、「abc」等)。
attributes.myAttribute:75	可查詢具有 "myAttribute" 屬性名稱的物件群組，且其值為 75。

查詢字串	結果
`attributes.myAttribute:[75 TO 80]`	可查詢具有 "myAttribute" 屬性名稱的物件群組，且其值介於一組數字範圍 (75–80，含) 內。
`attributes.myAttribute:[75 TO 80]`	可查詢具有 "myAttribute" 屬性名稱的物件群組，且其值介於一組數字範圍 (>75 且 <=80) 內。
`attributes.myAttribute: ["abcd" TO "abcf"]`	可查詢具有 "myAttribute" 屬性名稱的物件群組，且其值介於一組字母和數字的字串範圍內。以此查詢而言，傳回的物件群組必須具有 "serialNumber" 屬性且其值為 "abcd"、"abce" 或 "abcf"。
`attributes.myAttribute:i*t`	可查詢具有 "myAttribute" 屬性名稱的物件群組，且其值以 "i" 為首、後方接任意數量的字元，最後則是 "t"。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	使用布林值表達式，查詢結合詞彙的物件群組。以此查詢而言，傳回的物件群組必須具有「attr1」屬性名稱且其值為「abc」、具有「attr2」屬性名稱且其值小於 5 以及具有「attr3」屬性名稱且其值不超過 10。
`NOT attributes.myAttribute:cde`	可查詢具有 "myAttribute" 屬性名稱的物件群組，且不是 "cde"。
`parentGroupNames:(myParentThingGroupName)`	可查詢其父群組名稱與 "myParentThingGroupName" 相符的物件群組。
`parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)`	可查詢其父群組名稱與 "myParentThingGroupName" 或 "myRootThingGroupName" 相符的物件群組。
`parentGroupNames:(myParentThingGroupNa*)`	可查詢其父群組名稱以 "myParentThingGroupNa" 開頭的物件群組。

索引位置資料

您可以使用AWS IoT 機群索引為裝置上次傳送的位置資料編製索引，並使用地理位置搜尋裝置。此功能可解決裝置監控和管理使用案例，例如位置追蹤和鄰近搜尋。位置索引的運作方式類似於其他機群索引功能，以及要在物件索引中指定的其他組態。

常見的使用案例包括： 搜尋和彙總位於所需地理邊界內的裝置、使用與裝置中繼資料和狀態相關的查詢詞彙從索引資料來源取得位置特定洞見、提供精細檢視，例如篩選特定地理區域的結果，以減少機群監控地圖中的渲染延遲，並追蹤上次報告的裝置位置，以及識別超出所需邊界限制的裝置，並使用機群指標產生警示。若要開始使用位置索引和地理查詢，請參閱 ???。

支援的資料格式

AWS IoT 機群索引支援下列位置資料格式：

1. 眾所周知的座標參考系統文字表示

   遵循地理資訊 - 座標參考系統格式的已知文字表示法的字串。範例可以是 `"POINT(long lat)"`。
2. 代表座標的字串

   格式為 `"latitude, longitude"`或 `"longitude, latitude"` 的字串。如果您使用 `"longitude, latitude"`，您還必須在 `order`中指定 `geoLocations`。範例可以是 `"41.12,-71.34"`。
3. lat（緯度）、lon（經度）鍵的物件

   此格式適用於傳統陰影和具名陰影。支援的金鑰：`lat`、`latitude`、`lon`、`long`、`longitude`。範例可以是 `{"lat": 41.12, "lon": -71.34}`。
4. 代表座標的陣列

   格式為 `[lat,lon]`或 的陣列`[lon,lat]`。如果您使用與 GeoJSON 中的座標相同的格式（適用於傳統陰影和具名陰影）`[lon,lat]`，您還必須在 `order`中指定 `geoLocations`。

   範例可以是：

```
{
  "location": {
    "coordinates": [
```

```
      **Longitude**,
      **Latitude**
    ],
    "type": "Point",
    "properties": {
      "country": "United States",
      "city": "New York",
      "postalCode": "*****",
      "horizontalAccuracy": 20,
      "horizontalConfidenceLevel": 0.67,
      "state": "New York",
      "timestamp": "2023-01-04T20:59:13.024Z"
    }
  }
}
```

如何索引位置資料

下列步驟說明如何更新位置資料的索引組態，並使用地理查詢來搜尋裝置。

1. 了解位置資料的存放位置

   機群索引目前支援在傳統陰影或具名陰影中存放的索引位置資料。

2. 使用支援的位置資料格式

   請確定您的位置資料格式遵循其中一個支援的資料格式。

3. 更新索引組態

   至少需要時，啟用物件（登錄檔）索引組態。您還必須在包含位置資料的傳統影子或具名影子上啟用索引。更新物件索引時，您應該在索引組態中包含位置資料。

4. 建立和執行地理查詢

   根據您的使用案例，建立地理查詢並執行它們來搜尋裝置。您編寫的地理結構必須遵循查詢語法。您可以在 ??? 中找到一些範例。

更新物件索引組態

若要為位置資料編製索引，您必須更新索引組態並包含位置資料。根據您的位置資料存放位置，依照步驟更新您的索引組態：

存放在傳統影子的位置資料

如果您的位置資料存放在傳統影子中，您必須thingIndexingMode將 設定為 ，REGISTRY_AND_SHADOW並在 中的geoLocations欄位 (name 和 order) 中指定位置資料filter。

在下列物件索引組態範例中，您將位置資料路徑指定shadow.reported.coordinates為 name和 LonLat order。

```
{
 "thingIndexingMode": "REGISTRY_AND_SHADOW",
 "filter": {
  "geoLocations": [
   {
    "name": "shadow.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  索引模式會控制登錄檔或影子是否編製索引。當 thingIndexingMode 設定為 時OFF，物件索引會停用。

  若要為存放在傳統影子中的位置資料編製索引，您必須thingIndexingMode將 設定為 REGISTRY_AND_SHADOW。如需詳細資訊，請參閱???。

- filter

  索引篩選條件為具名影子和地理位置資料提供額外的選擇。如需詳細資訊，請參閱???。

- geoLocations

  您選取要編製索引的地理位置目標清單。索引的預設地理位置目標數目上限為 1。若要提高限制，請參閱AWS IoT Device Management 配額。

- name

  地理位置目標欄位的名稱。的範例值name可以是影子的位置資料路徑：shadow.reported.coordinates。

- order

地理位置目標欄位的順序。有效值： LatLon和 LonLat。 LatLon表示經緯度。 LonLat表示經度和經度。此欄位為選用欄位。預設值為 LatLon。

存放在具名影子中的位置資料

如果您的位置資料存放在具名影子中，namedShadowIndexingMode請將 設為 ON()，將具名影子名稱新增至 中的 namedShadowNames 欄位filter，並在 的 geoLocations 欄位中指定位置資料路徑filter。

在下列物件索引組態範例中，您將位置資料路徑指定shadow.name.namedShadow1.reported.coordinates為 name和 LonLat order。

```
{
 "thingIndexingMode": "REGISTRY",
 "namedShadowIndexingMode": "ON",
 "filter": {
  "namedShadowNames": [
   "namedShadow1"
  ],
  "geoLocations": [
   {
    "name": "shadow.name.namedShadow1.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  索引模式會控制登錄檔或影子是否編製索引。當 thingIndexingMode 設定為 時OFF，物件索引會停用。

  若要為存放在具名影子中的位置資料編製索引，您必須thingIndexingMode將 設定為 REGISTRY (或 REGISTRY_AND_SHADOW)。如需詳細資訊，請參閱???。

- filter

  索引篩選條件為具名影子和地理位置資料提供額外的選擇。如需詳細資訊，請參閱???。

- geoLocations

您選取要編製索引的地理位置目標清單。索引的預設地理位置目標數目上限為 1。若要提高限制，請參閱AWS IoT Device Management 配額。

- name

  地理位置目標欄位的名稱。的範例值name可以是影子的位置資料路徑：shadow.name.namedShadow1.reported.coordinates。

- order

  地理位置目標欄位的順序。有效值： LatLon和 LonLat。 LatLon表示經緯度。 LonLat表示經度和經度。此欄位為選用欄位。預設值為 LatLon。

範例地理查詢

完成位置資料的索引組態後，請執行地理查詢來搜尋裝置。您也可以將地理查詢與其他查詢字串合併。如需詳細資訊，請參閱???及???。

範例查詢 1

此範例假設位置資料存放在具名影子 中gps-tracker。此命令的輸出是與座標中心點 (47.6204，-122.3491) 徑向距離 15.5 km 內的裝置清單。

```
aws iot search-index --query-string \
"shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

範例查詢 2

此範例假設位置資料存放在傳統影子中。此命令的輸出是與座標中心點 (47.6204，-122.3491) 徑向距離 15.5 km 內的裝置清單。

```
aws iot search-index --query-string \
"shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

範例查詢 3

此範例假設位置資料存放在傳統影子中。此命令的輸出是未連線且在距中心點與座標 (47.6204，-122.3491) 徑向距離 15.5 km 外的裝置清單。

```
aws iot search-index --query-string \
```

```
"connectivity.connected:false AND (NOT
 shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km)"
```

入門教學課程

本教學課程示範如何使用機群索引為您的位置資料編製索引。為求簡化，您可以建立物件來代表您的裝置，並將位置資料存放在具名影子中，更新物件索引組態以建立位置索引，並執行範例地理查詢以搜尋放射邊界內的裝置。

此教學課程約需 15 分鐘方能完成。

在本主題中：

- 先決條件
- 建立物件和陰影
- 更新物件索引組態
- 執行地理查詢

先決條件

- 安裝最新版本的 AWS CLI。
- 熟悉位置索引和地理查詢、管理物件索引和查詢語法。

建立物件和陰影

您建立物件來代表您的裝置，並建立具名影子來存放其位置資料（座標 47.61564、-122.33584)。

1. 執行下列命令來建立物件，代表您名為 Bike-1 的自行車。如需如何使用 建立物件的詳細資訊 AWS CLI，請參閱AWS CLI參考中的 create-thing。

   ```
   aws iot create-thing --thing-name "Bike-1" \
   --attribute-payload '{"attributes": {"model":"OEM-2302-12", "battery":"35",
    "acqDate":"06/09/23"}}'
   ```

   此令命的輸出如下所示：

   ```
   {
       "thingName": "Bike-1",
   ```

```
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/Bike-1",
    "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df"
}
```

2. 執行下列命令來建立具名影子，以存放 Bike-1 的位置資料（座標 47.61564、-122.33584)。如需如何使用 建立具名影子的詳細資訊 AWS CLI，請參閱 AWS CLI參考中的 update-thing-shadow。

```
aws iot-data update-thing-shadow \
--thing-name Bike-1 \
--shadow-name Bike1-shadow \
--cli-binary-format raw-in-base64-out \
--payload '{"state":{"reported":{"coordinates":{"lat": 47.6153, "lon": -122.3333}}}}'
 \
"output.txt" \
```

此命令不會產生任何輸出。若要檢視您建立的具名影子，您可以執行 list-named-shadows-for-thing CLI 命令。

```
aws iot-data list-named-shadows-for-thing --thing-name Bike-1
```

此令命的輸出如下所示：

```
{
    "results": [
        "Bike1-shadow"
    ],
    "timestamp": 1699574309
}
```

更新物件索引組態

若要為位置資料編製索引，您必須更新物件索引組態，以包含位置資料。由於您的位置資料在本教學課程中存放在具名影子中，因此請將 `thingIndexingMode` 設為 `REGISTRY`（最低要求）、將 `namedShadowIndexingMode`設為 `ON`，並將您的位置資料新增至組態。在此範例中，您必須將具名影子的名稱和影子的位置資料路徑新增至 `filter`。

1. 執行 命令以更新位置索引的索引組態。

```
aws iot update-indexing-configuration --cli-input-json '{
```

```
"thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY",
"thingConnectivityIndexingMode": "OFF",
"deviceDefenderIndexingMode": "OFF",
"namedShadowIndexingMode": "ON",
"filter": {
    "namedShadowNames": ["Bike1-shadow"],
    "geoLocations":[{
        "name":"shadow.name.Bike1-shadow.reported.coordinates"
    }]
},
"customFields": [
{ "name":"attributes.battery",
"type":"Number"}] } }'
```

命令不會產生任何輸出。您可能需要等待片刻，直到更新完成。若要檢查狀態，請執行 describe-index CLI 命令。如果您看到 indexStatus顯示：ACTIVE，表示您的物件索引更新已完成。

2. 執行 命令來驗證您的索引組態。此為選擇性步驟。

```
aws iot get-indexing-configuration
```

輸出可能如下所示：

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY",
        "thingConnectivityIndexingMode": "OFF",
        "deviceDefenderIndexingMode": "OFF",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
```

```
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "attributes.battery",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "Bike1-shadow"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.Bike1-shadow.reported.coordinates",
                    "order": "LatLon"
                }
            ]
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

執行地理查詢

現在您已更新物件索引組態，以包含位置資料。嘗試建立一些地理查詢並執行它們，以查看您是否可以取得所需的搜尋結果。地理查詢必須遵循查詢語法。您可以在 中找到一些有用的範例地理查詢???。

在下列範例命令中，您可以使用 geoquery `shadow.name.Bike1-shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`搜尋與座標中心點 (47.6204，-122.3491) 徑向距離 15.5 km 內的裝置。

```
aws iot search-index --query-string "shadow.name.Bike1-
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

由於您有一個裝置位於座標「lat」：47.6153、「lon」：-122.3333，其距離中心點 15.5 km 的範圍內，因此您應該能夠在輸出中看到此裝置 (Bike-1)。輸出可能如下所示：

```
{
    "things": [
        {
            "thingName": "Bike-1",
            "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df",
            "attributes": {
                "acqDate": "06/09/23",
                "battery": "35",
                "model": "OEM-2302-12"
            },
            "shadow": "{\"reported\":{\"coordinates\":{\"lat\":47.6153,\"lon
\":-122.3333}},\"metadata\":{\"reported\":{\"coordinates\":{\"lat\":{\"timestamp
\":1699572906},\"lon\":{\"timestamp\":1699572906}}}},\"hasDelta\":false,\"version\":1}"
        }
    ]
}
```

如需詳細資訊，請參閱???。

機群指標

機群指標是機群索引功能，這是一種受管服務，可讓您在其中編製、搜尋和彙總裝置的資料 AWS IoT。您可以使用機群指標，在 CloudWatch 中監控機群裝置隨著時間的彙總狀態，包括檢閱機群裝置的斷線速率或指定期間的平均電池電量變更。

您可以透過機群指標建置彙總查詢，其結果會以指標的形式不斷發送到 CloudWatch，以供分析趨勢和建立警示。您可以為監控任務指定以下彙總類型的彙總查詢：Statistics (統計數字)、Cardinality (基數) 以及 Percentile (百分位數)。您可以儲存所有彙總查詢來建立機群指標，以供日後重複使用。

入門教學課程

在此教學課程中，您要建立能夠監控感應器溫度的機群指標，以偵測潛在的異常情況。建立機群指標時，您可以定義彙總查詢，其可偵測溫度超過華氏 80 度的感應器數目。您可以指定每 60 秒執行一次查詢，其結果會發送到 CloudWatch；您可以在 CloudWatch 中檢視具有潛在高溫風險的感應器數量，並設定警示。為完成此教學課程，您需要使用 AWS CLI。

於本教學課程中，您會了解如何：

- 設定
- 建立機群指標
- 在 CloudWatch 中檢視指標
- 清除資源

此教學課程約需 15 分鐘方能完成。

先決條件

- 安裝 AWS CLI 的最新版本
- 熟悉如何查詢彙總資料
- 熟悉如何使用 Amazon CloudWatch 指標

設定

若要使用機群指標，請啟用機群索引。若要針對具有指定資料來源和相關組態的物件或物件群組啟用機群索引，請遵循管理物件索引和管理物件群組索引中的指示。

設定

1. 執行下列命令來啟用機群索引，並指定要從中搜尋的資料來源。

```
aws iot update-indexing-configuration \
--thing-indexing-configuration
 "thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.temperature,type=Num
```

```
{name=attributes.rackId,type=String},
{name=attributes.stateNormal,type=Boolean}],thingConnectivityIndexingMode=STATUS" \
```

上述的 CLI 命令範例可啟用機群索引，支援使用 AWS_Things 索引來搜尋登錄檔資料、影子資料和物件連線狀態。

組態變更可能需要幾分鐘時間才能完成。在建立機群指標之前，請驗證是否已啟用機群索引。

若要檢查機群索引是否已啟用，請執行下列 CLI 命令：

```
aws --region us-east-1 iot describe-index --index-name "AWS_Things"
```

如需詳細資訊，請參閱啟用物件索引。

2. 執行以下 Bash 指令碼來建立十個物件並提供相應描述。

```
# Bash script. Type `bash` before running in other shells.

Temperatures=(70 71 72 73 74 75 47 97 98 99)
Racks=(Rack1 Rack1 Rack2 Rack2 Rack3 Rack4 Rack5 Rack6 Rack6 Rack6)
IsNormal=(true true true true true true false false false false)

for ((i=0; i < 10; i++))
do
  thing=$(aws iot create-thing --thing-name "TempSensor$i" --attribute-
payload attributes="{temperature=${Temperatures[@]:$i:1},rackId=${Racks[@]:
$i:1},stateNormal=${IsNormal[@]:$i:1}}")
  aws iot describe-thing --thing-name "TempSensor$i"
done
```

這個指令碼建立了十個物件來表示十個感應器。如下表所述，每個物件都有 temperature、rackId 以及 stateNormal 屬性：

屬性	資料類型	說明
temperature	Number	以華氏為單位的溫度值
rackId	String	包含感應器的伺服器機架 ID
stateNormal	Boolean	感應器溫度值是否正常

此指令碼的輸出包含十個 JSON 檔案。其中一個 JSON 檔案如下所示：

```
{
    "version": 1,
    "thingName": "TempSensor0",
    "defaultClientId": "TempSensor0",
    "attributes": {
        "rackId": "Rack1",
        "stateNormal": "true",
        "temperature": "70"
    },
    "thingArn": "arn:aws:iot:region:account:thing/TempSensor0",
    "thingId": "example-thing-id"
}
```

如需詳細資訊，請參閱建立物件。

建立機群指標

若要建立機群指標

1. 執行下列命令建立名為 *high_temp_FM* 的機群指標。您可以建立機群指標，以此監控 CloudWatch 中華氏溫度超過 80 度的感應器數量。

```
aws iot create-fleet-metric --metric-name "high_temp_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature >80" --period 60 --aggregation-
field "attributes.temperature" --aggregation-type name=Statistics,values=count
```

--metric-name

資料類型：字串。此 `--metric-name` 參數會指定機群指標名稱。在此範例中，您建立的是名為 high_temp_FM 的機群指標。

--query-string

資料類型：字串。此 `--query-string` 參數會指定查詢字串。在這個範例中，查詢字串意味著查詢名稱開頭為 TempSensor 和溫度高於華氏 80 度的所有物件。如需詳細資訊，請參閱查詢語法。

--period

資料類型：整數。此 `--period` 參數會指定彙總資料的擷取時間 (以秒為單位)。在此範例中，您指定要建立的機群指標每 60 秒擷取一次彙總資料。

--aggregation-field

資料類型：字串。此 `--aggregation-field` 參數會指定要評估的屬性。在此範例中，要評估的是溫度屬性。

--aggregation-type

此 `--aggregation-type` 參數指定要在機群指標中顯示的統計摘要。您可以為監控任務自訂以下彙總類型的彙總查詢屬性：Statistics (統計數字)、Cardinality (基數) 以及 Percentile (百分位數)。在此範例中，您可以針對彙總類型和統計資料指定計數，以傳回具有與查詢相符之屬性的裝置計數，也就是傳回名稱開頭為 TempSensor 和溫度高於華氏 80 度的裝置計數。如需詳細資訊，請參閱查詢彙總資料。

此令命的輸出結果如下所示：

```
{
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "metricName": "high_temp_FM"
}
```

> **Note**
>
> 資料點可能需要一些時間才能在 CloudWatch 中顯示。

若要進一步了解如何建立機群指標，請參閱管理機群指標。

如果無法建立機群指標，請參閱機群指標故障診斷。

2. (選用) 執行下列命令來描述名為 high_temp_FM 的機群指標：

```
aws iot describe-fleet-metric --metric-name "high_temp_FM"
```

此令命的輸出結果如下所示：

```
{
```

```
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625249775.834,
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625249775.834,
    "metricName": "high_temp_FM"
}
```

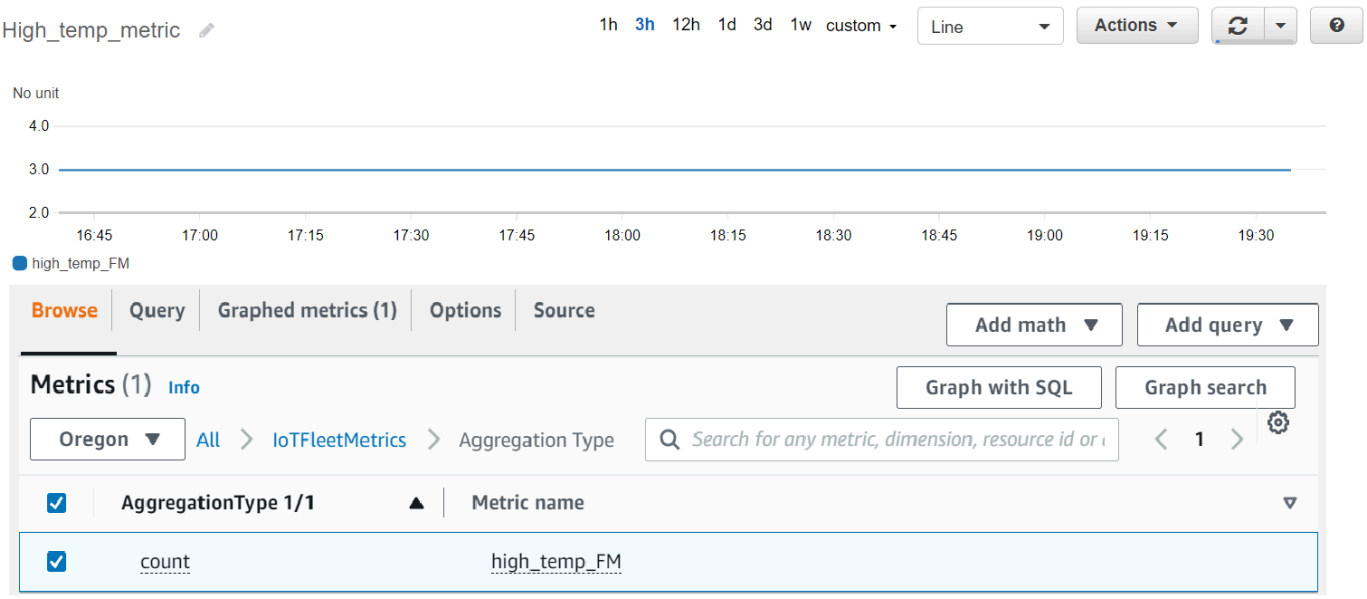
在 CloudWatch 中檢視機群指標

建立機群指標後，您可以在 CloudWatch 中檢視指標資料。在本教學課程中，您會看到一個指標，其中顯示名稱開頭為 TempSensor 和溫度高於華氏 80 度的感應器數量。

在 CloudWatch 中檢視資料點

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側面板的 CloudWatch 功能表上，選擇指標來展開子功能表，然後選擇所有指標。此操作會開啟頁面，其中上半部分會顯示圖形，下半部分則包含四個標籤式區段。
3. 第一個標籤式區段所有指標會列出可以在群組中檢視的所有指標。請選擇 IoTFleetMetrics。這包含您的所有機群指標。
4. 在 All metrics (所有指標) 索引標籤的 Aggregation type (彙總類型) 區段，選擇 Aggregation type (彙總類型) 檢視建立的所有機群指標。
5. 選擇機群指標，在 Aggregation type (彙總類型) 區段的左側顯示圖表。您會看到指標名稱左側的值 *count*，這是您在本教學課程的建立機群指標區段中指定的彙總類型值。
6. 選擇位於 All metrics (所有指標) 索引標籤右側的名為 Graphed metrics (圖表化指標) 的第二個索引標籤，檢視從上一步中選擇的機群指標。

   您應該可以看到如下所示的圖表，其中顯示溫度高於華氏 80 度的感應器數量：



> **Note**
>
> CloudWatch 中的 Period (期間) 屬性預設為 5 分鐘。這是在 CloudWatch 中顯示的資料點之間的時間間隔。您可以根據需求變更 Period (期間) 設定。

7. (選用) 您可以設定指標警示。

   1. 在左側面板的 CloudWatch 功能表上，選擇警示來展開子功能表，然後選擇所有警示。
   2. 在 Alarms (警示) 頁面中，選擇右上角的 Create alarm (建立警示)。請遵循主控台中的 Create alarm (建立警示) 指示，視需要建立警示。如需詳細資訊，請參閱使用 Amazon CloudWatch 警示。

如需進一步了解，請參閱使用 Amazon CloudWatch 指標。

如果無法在 CloudWatch 中看到資料點，請參閱機群指標故障診斷。

清除

刪除機群指標

可以使用 delete-fleet-metric CLI 命令來刪除機群指標。

若要刪除名為 high_temp_FM 的機群指標，請執行下列命令。

```
aws iot delete-fleet-metric --metric-name "high_temp_FM"
```

清理物件

可以使用 delete-thing CLI 命令來刪除物件。

若要刪除您已建立的十個物件，請執行下列指令碼：

```
# Bash script. Type `bash` before running in other shells.

for ((i=0; i < 10; i++))
do
  thing=$(aws iot delete-thing --thing-name "TempSensor$i")
done
```

清除 CloudWatch 中的指標

CloudWatch 不支援刪除指標。指標的過期時間因保留排程而定。如需進一步了解，請參閱使用 Amazon CloudWatch 指標。

管理機群指標

本主題說明如何使用 AWS IoT 主控台和 AWS CLI 來管理您的機群指標。

主題

- 管理機群指標 (主控台)
- 管理機群指標 (CLI)
- 授權對 IoT 資源進行標記

管理機群指標 (主控台)

下列各節說明如何使用 AWS IoT 主控台來管理您的機群指標。在建立機群指標之前，請確保已啟用機群索引與相關資料來源和組態。

啟用機群索引

如果已啟用機群索引，請略過此區段。

如果尚未啟用機群索引，請依照下列指示進行。

1. 在 https：//https://console.aws.amazon.com/iot/ 開啟您的 AWS IoT 主控台。
2. 在 AWS IoT 功能表中，選擇設定。

3. 若要檢視詳細設定，請在 Settings (設定) 頁面向下捲動到 Fleet indexing (機群索引) 區段。
4. 若要更新機群索引設定，請在 Fleet indexing (機群索引) 區段的右側，選取 Manage indexing (管理索引)。
5. 在 Manage fleet indexing (管理機群索引) 頁面上，根據需求更新機群索引設定。
   - 組態

     若要開啟物件索引，請將 Thing indexing (物件索引) 開啟，然後選取要從中建立索引的資料來源。

     若要開啟物件群組索引，請將 Thing group indexing (物件群組索引) 開啟。
   - Custom fields for aggregation - optional (彙總的自訂欄位 – 選用)

     自訂欄位是欄位名稱和欄位類型對的清單。

     若要新增自訂欄位配對，請選擇 Add new field (新增欄位)。請輸入自訂欄位名稱 (例如 `attributes.temperature`)，然後從 Field type (欄位類型) 選單中選取欄位類型。請注意，自訂欄位名稱開頭為 `attributes.` 並會儲存為屬性，以執行物件彙總查詢。

     若要更新並儲存設定，請選擇 Update (更新)。

建立機群指標

1. 在 https：//https://console.aws.amazon.com/iot/ 開啟您的 AWS IoT 主控台。
2. 在 AWS IoT 功能表中，選擇管理，然後選擇機群指標。
3. 在 Fleet metrics (機群指標) 頁面中，選擇 Create fleet metric (建立機群指標) 並完成建立步驟。
4. 在步驟 1 中設定機群指標
   - 在 Query (查詢) 區段中輸入查詢字串，指定要執行彙總搜尋的物件或事物群組。查詢字串由屬性和值組成。對於 Properties (屬性)，選擇所需屬性；如果所需屬性未顯示在清單中，請在欄位中輸入屬性。輸入 `:` 後面的值。範例查詢字串可以是 `thingName:TempSensor*`。每輸入一個查詢字串，就按一次鍵盤上的 Enter 鍵。如果輸入多個查詢字串，請選取 and (和)、or (或)、and not (而非) 或 or not (或非) 來指定其關係。
   - 在 Report properties (報告屬性) 中，從其各自的清單中選擇 Index name (索引名稱)、Aggregation type (彙總類型) 以及 Aggregation field (彙總欄位)。接下來，在 Select data (選取資料) 中選取要彙總的資料，可以在其中選取多個資料值。
   - 選擇下一步。
5. 在步驟 2 中指定機群指標屬性

- 在 Fleet metric name (機群指標名稱) 欄位中，為要建立的機群指標輸入名稱。
- 在 Description - optional (描述 – 選用) 欄位中，為要建立的機群指標輸入描述。此欄位為選用欄位。
- 在 Hours (小時) 和 Minutes (分鐘) 欄位中，輸入希望機群指標向 CloudWatch 發送資料的時間 (頻率)。
- 選擇下一步。

6. 在步驟 3 中檢閱和建立
   - 檢閱步驟 1 和步驟 2 的設定。若要編輯設定，請選擇 Edit (編輯)。
   - 選擇 Create fleet metric (建立機群指標)。

成功建立之後，機群指標會列示在 Fleet metric (機群指標) 頁面上。

更新機群指標

1. 在機群指標頁面上，選擇要更新的機群指標。
2. 在機群指標 Details (詳細資訊) 頁面上，選擇 Edit (編輯)。此操作會開啟建立步驟，讓您可以在這三個步驟中的任一步驟更新機群指標。
3. 完成機群指標的更新後，選擇 Update fleet metric (更新機群指標)。

刪除機群指標

1. 在機群指標頁面上，選擇要刪除的機群指標。
2. 在顯示機群指標詳細資訊的下一個頁面上，選擇 Delete (刪除)。
3. 在對話方塊中，輸入機群指標的名稱以確認刪除。
4. 選擇 刪除。此步驟會永久刪除機群指標。

管理機群指標 (CLI)

下列各節說明如何使用 AWS CLI 來管理您的機群指標。在建立機群指標之前，請確保已啟用機群索引與相關資料來源和組態。若要為物件或物件群組啟用機群索引，請遵循管理物件索引或者管理物件群組索引中的指示。

建立機群指標

您可以使用 create-fleet-metric CLI 命令來建立機群指標。

```
aws iot create-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 60 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum
```

此命令的輸出包含機群指標的名稱和 Amazon Resource Name (ARN)。輸出看起來如下：

```
{
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "metricName": "YourFleetMetricName"
}
```

清單機群指標

您可以使用 list-fleet-metric CLI 命令，列出帳戶中的所有機群指標。

```
aws iot list-fleet-metrics
```

此命令的輸出包含所有機群指標。輸出看起來如下：

```
{
    "fleetMetrics": [
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric1",
            "metricName": "YourFleetMetric1"
        },
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric2",
            "metricName": "YourFleetMetric2"
        }
    ]
}
```

描述機群指標

您可以使用 describe-fleet-metric CLI 命令，顯示有關機群指標的其他詳細資訊。

```
aws iot describe-fleet-metric --metric-name "YourFleetMetricName"
```

此命令的輸出包含與指定機群指標相關的詳細資訊。輸出看起來如下：

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625790642.355,
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "sum"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625790642.355,
    "metricName": "YourFleetMetricName"
}
```

更新機群指標

您可以使用 update-fleet-metric CLI 命令更新機群指標。

```
aws iot update-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 120 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum,count --index-name AWS_Things
```

update-fleet-metric 命令不會產生任何輸出。可以使用 describe-fleet-metric CLI 命令查看結果。

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625792300.881,
    "queryString": "*",
    "period": 120,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 2,
    "aggregationType": {
        "values": [
            "sum",
            "count"
        ],
```

```
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625792300.881,
    "metricName": "YourFleetMetricName"
}
```

刪除機群指標

使用 delete-fleet-metric CLI 命令刪除機群指標。

```
aws iot delete-fleet-metric --metric-name "YourFleetMetricName"
```

如果刪除成功或指定機群指標不存在，則此命令不會產生任何輸出。

如需詳細資訊，請參閱針對機群指標進行疑難排解。

授權對 IoT 資源進行標記

為了更好地控制您可以建立、修改或使用的機群指標，您可以將標籤連接到機群指標。

若要標記您使用 AWS 管理主控台 或 建立的機群指標 AWS CLI，您必須在 IAM 政策中包含 `iot:TagResource`動作，以授予使用者許可。如果您的 IAM 政策不包含 `iot:TagResource`，使用標籤建立機群指標的任何動作都會傳回`AccessDeniedException`錯誤。

如需標記資源的一般資訊，請參閱標記資源 AWS IoT。

IAM 政策範例

當您建立機群指標時，請參閱下列授予標記許可的 IAM 政策範例：

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Action": [
    "iot:TagResource"
   ],
   "Effect": "Allow",
   "Resource": [
    "arn:aws:iot:*:*:fleetmetric/*"
```

```
    ]
   },
   {
    "Action": [
     "iot:CreateFleetMetric"
    ],
    "Effect": "Allow",
    "Resource": [
     "arn:aws:iot:*:*:index/*",
     "arn:aws:iot:*:*:fleetmetric/*"
    ]
   }
  ]
}
```

如需詳細資訊，請參閱[適用於 AWS IoT的動作、資源和條件鍵](#)。

MQTT 型檔案交付

您可以使用 MQTT 型檔案交付來管理檔案並將其傳輸至機群中的 AWS IoT 裝置。透過 AWS 雲端中的此功能，您可以建立包含多個檔案的串流，您可以更新串流資料（檔案清單和描述）、取得串流資料等等。 AWS IoT MQTT 型檔案交付可以使用 MQTT 通訊協定，在 JSON 或 CBOR 中支援請求和回應訊息，以小區塊傳輸資料到您的 IoT 裝置。

如需如何使用 往返 IoT 裝置傳輸資料的詳細資訊 AWS IoT，請參閱 將裝置連接至 AWS IoT。

主題

- 什麼是串流？
- 在 AWS 雲端中管理串流
- 在裝置中使用 AWS IoT MQTT 型檔案交付
- FreeRTOS OTA 中的範例使用案例

什麼是串流？

在 中 AWS IoT，串流是可公開定址的資源，是可傳輸至 IoT 裝置的檔案清單摘要。典型的串流包含下列資訊：

- 一個在特定時間唯一識別串流的 Amazon 資源名稱 (ARN)。此 ARN 的模式為 `arn:partition:iot:region:account-ID:stream/stream ID`。
- 識別串流並在 AWS Command Line Interface () 或 SDK 命令中使用（且通常需要AWS CLI) 的串流 ID。
- 串流說明，提供串流資源的說明。
- 串流版本，識別串流的特定版本。由於串流資料可在裝置開始資料傳輸之前立即修改，因此裝置可使用串流版本來強制執行一致性檢查。
- 檔案清單，可傳輸至裝置。對於清單中的每個檔案，串流會記錄檔案 ID、檔案大小及檔案的地址資訊，這些資訊包含例如 Amazon S3 儲存貯體名稱、物件金鑰和物件版本。
- AWS Identity and Access Management (IAM) 角色，授予 AWS IoT 以 MQTT 為基礎的檔案交付許可，以讀取存放在資料儲存體中的串流檔案。

AWS IoT MQTT 型檔案交付提供下列功能，讓裝置可以從 雲端傳輸資料 AWS ：

- 使用 MQTT 通訊協定傳輸資料。

- 支援 JSON 或 CBOR 格式。
- 說明串流 (`DescribeStream` API) 的能力，以取得串流檔案清單、串流版本及相關資訊。
- 在小區塊形式傳送資料 (`GetStream` API) 的能力，使具有硬體限制的裝置可收到區塊。
- 支援每個請求的動態區塊大小，以支援具有不同記憶體容量的裝置。
- 當多個裝置從同一個串流檔案請求資料區塊時，最佳化並行串流請求。
- Amazon S3 作為串流檔案的資料儲存。
- 支援從 AWS IoT MQTT 型檔案交付到 CloudWatch 的資料傳輸日誌發佈。

如需 MQTT 型檔案交付配額，請參閱 AWS 一般參考 中的 AWS IoT Core Service Quotas。

在 AWS 雲端中管理串流

AWS IoT 提供 AWS SDK 和 AWS CLI 命令，可讓您用來管理 AWS 雲端中的串流。您可使用這些命令來執行下列：

- 建立串流 CLI / SDK
- 說明串流以取得其資訊。CLI / SDK
- 列出 中的串流 AWS 帳戶。CLI / SDK
- 更新串流中的檔案清單或串流說明。CLI / SDK
- 刪除串流。CLI / SDK

> **Note**
>
> 目前，串流不會顯示於 AWS 管理主控台之中。您必須使用 AWS CLI 或 AWS SDK 來管理 中的串流 AWS IoT。此外，嵌入式 C SDK 是唯一支援 MQTT 型檔案傳輸的 SDK。

從裝置使用 AWS IoT MQTT 型檔案交付之前，您必須確保裝置符合下列條件，如下節所示：

- 反映透過 MQTT 傳輸資料所需正確許可的政策。
- 您的裝置可以連線至 AWS IoT Device Gateway。
- 政策陳述式，指出您可以標記資源。如果`CreateStream`使用標籤呼叫 ，則需要 `iot:TagResource` 。

在您從裝置使用 AWS IoT MQTT 型檔案交付之前，必須遵循下一節中的步驟，以確保您的裝置獲得適當授權，並可連線至 AWS IoT 裝置閘道。

授與您的裝置許可

您可遵循建立 AWS IoT 政策中的步驟，來建立裝置政策或使用現有的裝置政策。將政策連接至與您裝置相關聯的憑證，並將下列許可新增至裝置政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        }
    ]
}
```

將您的裝置連線至 AWS IoT

使用 AWS IoT MQTT 型檔案交付的裝置需要連線 AWS IoT。 AWS IoT MQTT 型檔案交付會在 AWS 雲端 AWS IoT 中與 整合，因此您的裝置應該直接連線至AWS IoT 資料平面的端點。

> **Note**
>
> AWS IoT 資料平面的端點專屬於 AWS 帳戶 和 區域。您必須將 端點用於裝置註冊所在的 AWS 帳戶 和 區域 AWS IoT。

如需詳細資訊，請參閱連線至 AWS IoT Core。

TagResource 用量

`CreateStream` API 動作會建立串流，以透過 MQTT 在區塊中交付一或多個大型檔案。

成功的 `CreateStream` API 呼叫需要下列許可：

- `iot:CreateStream`
- `iot:TagResource`（如果 `CreateStream` 具有標籤）

支援這兩個許可的政策如下所示：

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateStream",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:stream/streamId"
    }
}
```

需要iot:TagResource政策陳述式動作，以確保使用者無法在沒有適當許可的情況下在資源上建立或更新標籤。如果沒有 的規格政策陳述式動作iot:TagResource，AccessDeniedException如果請求隨附標籤，CreateStreamAPI 呼叫將傳回 。

如需詳細資訊，請參閱下列連結：

- CreateStream
- TagResource
- 標籤

在裝置中使用 AWS IoT MQTT 型檔案交付

如要啟動資料傳輸程序，裝置必須接收初始資料集，其至少包含串流 ID。在任務文件中含括初始資料集，您可使用 AWS IoT 任務 來排程裝置的資料傳輸任務。當裝置收到初始資料集時，應該開始與以 AWS IoT MQTT 為基礎的檔案交付互動。若要與 AWS IoT MQTT 型檔案交付交換資料，裝置應該：

- 使用 MQTT 通訊協定來訂閱 MQTT 型檔案交付主題。
- 使用 MQTT 訊息傳送請求，然後等待接收回應。

您可選擇性地在初始資料集中包含串流檔案 ID 和串流版本。將串流檔案 ID 傳送至裝置可簡化裝置韌體/軟體的程式設計，因為其不需要提出 DescribeStream 請求來取得此 ID。裝置可指定 GetStream 請求中的串流版本，強制執行一致性檢查，以防串流意外更新。

使用 DescribeStream 以取得串流資料

AWS IoT MQTT 型檔案交付提供 DescribeStream API，可將串流資料傳送至裝置。此 API 傳回的串流資料包括串流 ID、串流版本、串流說明和串流檔案清單，每個檔案都有一個檔案 ID 和檔案大小 (以位元組為單位)。裝置可利用此資訊來選取任意檔案以啟動資料傳輸程序。

> **Note**
>
> 若您的裝置在初始資料集中收到所有必要的串流檔案 ID，您無須使用 DescribeStream API。

請依照下列步驟提出 DescribeStream 請求。

1. 訂閱「已接受」主題篩選條件 $aws/things/*ThingName*/streams/*StreamId*/description/json。
2. 訂閱「已拒絕」主題篩選條件 $aws/things/*ThingName*/streams/*StreamId*/rejected/json。
3. 將訊息傳送至 $aws/things/*ThingName*/streams/*StreamId*/describe/json 來發佈 DescribeStream 請求。
4. 若接受該請求，您的裝置將會在「已接受」主題篩選條件上收到 DescribeStream 回應。
5. 若請求遭到拒絕，您的裝置將會在「已拒絕」主題篩選條件上收到錯誤回應。

> **Note**
>
> 若您在顯示的主題和主題篩選條件中將 json 替換為 cbor，您的裝置將會收到 CBOR 格式的訊息，這比 JSON 更精簡。

DescribeStream 請求

JSON 中的典型 DescribeStream 請求看起來會如下列範例所示。

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- (選用) "c" 為用戶端字符欄位。

  用戶端字符不可超過 64 個位元組。長於 64 個位元組的用戶端字符會造成錯誤回應及 InvalidRequest 錯誤訊息。

DescribeStream 回應

JSON 中的 DescribeStream 回應看起來會如下列範例所示。

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039",
    "s": 1,
    "d": "This is the description of stream ABC.",
    "r": [
```

```
        {
            "f": 0,
            "z": 131072
        },
        {
            "f": 1,
            "z": 51200
        }
    ]
}
```

- "c" 為用戶端字符欄位。若其提供於 DescribeStream 請求中，則會傳回。使用用戶端字符，使回應與其請求產生關聯。
- "s" 是整數形式的串流版本。您可使用此版本，對您的 GetStream 請求執行一致性檢查。
- "r" 包含串流中的一份檔案清單。
  - "f" 是整數形式的串流檔案 ID。
  - "z" 是以位元組數為單位的串流檔案大小。
- "d" 包含串流的說明。

從串流檔案取得資料區塊

您可使用 GetStream API，裝置便可接收小型資料區塊形式的串流檔案，因此其可供處理大型區塊大小有限制的裝置使用。如要接收整個資料檔案，裝置可能需要傳送或接收多個請求和回應，直至所有資料區塊已接收並加以處理為止。

GetStream 請求

請依照下列步驟提出 GetStream 請求。

1. 訂閱「已接受」主題篩選條件 $aws/things/*ThingName*/streams/*StreamId*/data/json。
2. 訂閱「已拒絕」主題篩選條件 $aws/things/*ThingName*/streams/*StreamId*/rejected/json。
3. 將 GetStream 請求發佈至主題 $aws/things/*ThingName*/streams/*StreamId*/get/json。
4. 若接受該請求，您的裝置將會在「已接受」主題篩選條件上收到一個或多個 GetStream 回應。每個回應訊息包含一個單一區塊的基本資訊和資料承載。

5. 重複步驟 3 和 4，以接收所有資料區塊。若請求的資料量大於 128 KB，您必須重複這些步驟。您必須設定裝置的程式，來使用多個 GetStream 請求，以接收所有請求的資料。
6. 若請求遭到拒絕，您的裝置將會在「已拒絕」主題篩選條件上收到錯誤回應。

> **Note**
>
> - 若您在顯示的主題和主題篩選條件中將 "json" 替換為 "cbor"，您的裝置將會收到 CBOR 格式的訊息，這比 JSON 更精簡。
> - AWS IoT 以 MQTT 為基礎的檔案交付會將區塊的大小限制為 128 KB。若您對超過 128 KB 的區塊提出請求，請求將會失敗。
> - 您可對總大小大於 128 KB 的多個區塊提出請求 (例如，若您對總量為 160 KB 的資料提出 5 個區塊 (每個區塊 32 KB) 的請求)。在這種情況下，請求不會失敗，但您的裝置必須提出多個請求才能接收所有請求的資料。當您的裝置提出其他請求時，服務會傳送其他區塊。我們建議您僅於正確接收並處理先前的回應之後，才可繼續進行新的請求。
> - 無論所請求的資料大小總量為何，您都應該將裝置設定為在未收到區塊或未正確收到區塊時啟動重試。

JSON 中的典型 GetStream 請求看起來會如下列範例所示。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4096,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [選用] "c" 為用戶端字符欄位。

  用戶端字符不能超過 64 位元組。長於 64 個位元組的用戶端字符會造成錯誤回應及 InvalidRequest 錯誤訊息。
- [選用] "s" 為串流版本欄位 (一個整數)。

MQTT 型檔案交付會依據此請求的版本和雲端中的最新串流版本套用一致性檢查。若從 GetStream 請求中的裝置傳送的串流版本與雲端中最新串流版本不相符，則服務會傳送錯誤回應及 VersionMismatch 錯誤訊息。一般而言，裝置會在初始資料集或 DescribeStream 的回應中接收預期的 (最新的) 串流版本。

- "f" 是串流檔案 ID (範圍為 0 到 255 的整數)。

  當您使用 或 SDK 建立 AWS CLI 或更新串流時，需要串流檔案 ID。若裝置請求 ID 不存在的串流檔案，則服務會傳送錯誤回應和 ResourceNotFound 錯誤訊息。

- "l" 為以位元組為單位的資料區塊大小 (範圍為 256 到 131,072 的整數)。

  請參閱 建置 GetStream 請求的點陣圖，以取得有關如何使用點陣圖欄位來指定 GetStream 回應中將會傳回之串流檔案部分的說明。若裝置指定一個超出範圍的區塊大小，則服務會傳送錯誤回應和 BlockSizeOutOfBounds 錯誤訊息。

- [選用] "o" 為串流檔案中區塊的偏移量 (範圍為 0 到 98,304 的整數)。

  請參閱 建置 GetStream 請求的點陣圖，以取得有關如何使用點陣圖欄位來指定 GetStream 回應中將會傳回之串流檔案部分的說明。98,304 的最大值是以 24 MB 串流檔案大小限制為基礎，最小區塊大小為 256 個位元組。若未指定，則預設值為 0。

- [選用] "n" 為請求的區塊數量 (範圍為 0 到 98,304 的整數)。

  "n" 欄位指定 (1) 請求的區塊數量，或 (2) 使用點陣圖欄位 ("b") 時，點陣圖請求將會傳回區塊數量的限制。此第二次使用是可選的。若未定義，其預設為 131072/*DataBlockSize*。

- [選用] "b" 是個代表所請求區塊的點陣圖。

  使用點陣圖，您的裝置可以請求不連續的區塊，這使得在錯誤後處理重試更為方便。請參閱 建置 GetStream 請求的點陣圖，以取得有關如何使用點陣圖欄位來指定 GetStream 回應中將會傳回之串流檔案部分的說明。對此欄位，請將點陣圖轉換為以十六進記法表示點陣圖值的字串。點陣圖必須小於 12,288 個位元組。

> **Important**
>
> 應指定 "n" 或 "b"。如果未指定，當檔案大小小於 131072 位元組 (128 KB) 時，GetStream 請求可能無效。

GetStream 回應

對於請求的每個資料區塊，JSON 中的 GetStream 回應看似此範例。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4096,
    "i": 2,
    "p": "..."
}
```

- "c" 為用戶端字符欄位。若其提供於 GetStream 請求中，則會傳回。使用用戶端字符，使回應與其請求產生關聯。
- "f" 為目前資料區塊承載所屬串流檔案的 ID。
- "l" 為資料區塊承載的大小 (以位元組為單位)。
- "i" 為包含於承載中的資料區塊 ID。資料區塊的編號從 0 開始。
- "p" 包含資料區塊承載。此欄位是一個字串，表示以 Base64 編碼的資料區塊值。

建置 GetStream 請求的點陣圖

您可使用 GetStream 請求中的點陣圖欄位 (b)，從串流檔案獲取非連續區塊。此有助於 RAM 容量有限的裝置處理網路交付問題。裝置僅能請求那些未接收或未正確接收的區塊。點陣圖決定將會傳回的串流檔案區塊。對於點陣圖中設定為 1 的每個位元，會傳回串流檔案相對應的區塊。

下列範例說明如何在 GetStream 請求中指定點陣圖及其支援欄位。例如，您想要以 256 個位元組 (區塊大小) 的區塊接收串流檔案。將每個 256 個位元組的區塊視為具有一個數字，指定其在檔案中的位置，從 0 開始。所以區塊 0 是檔案中第一個 256 個位元組的區塊，區塊 1 是第二個區塊，依此類推。您想要從該檔案請求區塊 20、21、24 和 43。

區塊偏移

因為第一個區塊為數字 20，請指定偏移 (欄位 o) 為 20 以節省點陣圖中的空間。

區塊的數量

為確保您的裝置不會收到超過其在有限記憶體資源下可處理的區塊數，您可指定 MQTT 型檔案交付傳送的每則訊息中應傳回的區塊數量上限。請注意，若點陣圖本身指定的區塊數量小於此數，或者其讓 MQTT 型檔案交付傳送的回應訊息總大小大於每個 GetStream 請求 128 KB 的服務限制，則會忽略此值。

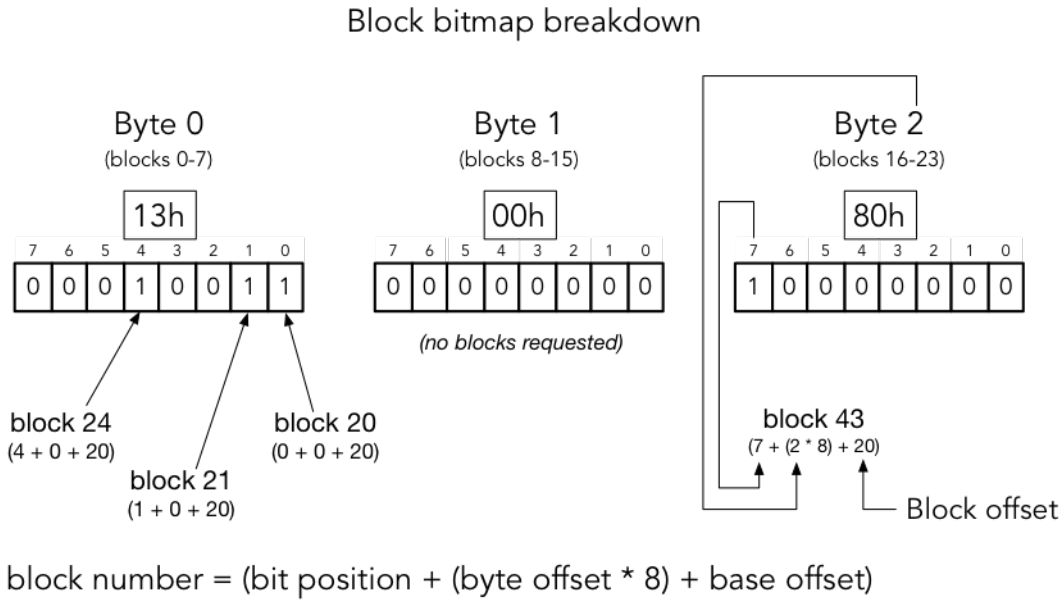
區塊點陣圖

點陣圖本身是一個以十六進記法表示的無符號位元組陣列，包含於 `GetStream` 請求中作為數字的字串表示。但若要建構此字串，讓我們先將點陣圖視為一長串的位元 (二進制數)。若在此序列中的某個位元設定為 1，則串流檔案相對應的區塊將會傳送回裝置。對於我們的範例，我們想要接收區塊 20、21、24 和 43，因此我們必須在點陣圖中設定位元 20、21、24 和 43。我們可以使用區塊偏移來節省空間，因此在我們從每個區塊減去偏移量之後，我們要設定位元 0、1、4 和 23，如下列範例所示。

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

一次取一個位元組 (8 位元)，這通常會寫成："0b00010011"、"0b00000000" 和 "0b10000000"。位元 0 出現在我們的二進製表示中的第一個位元組末尾，位元 23 出現在最後一個位元組的開頭。除非您知道慣例，否則這可能會令人困擾。第一個位元組包含位元 7-0 (依該順序)，第二個位元組包含位元 15-8，第三個位元組包含位元 23-16，依此類推。於十六進記法中，此會轉換為 "0x130080"。

> **Tip**
>
> 您可將標準二進制轉換為十六進記法。一次取四個二進制數字，並將其轉換為相對應的十六進制。例如，"0001" 變成 "1"，"0011" 變成 "3" 等等。



綜上所述，我們 `GetStream` 請求的 JSON 如下所示。

```
{
    "c" : "1",
    "s" : 1,
    "l" : 256,
    "f" : 1,
    "o" : 20,
    "n" : 32,
    "b" : "130080"
}
```

- "c" 為用戶端字符欄位。
- 「s」是預期串流版本。
- "l" 為資料區塊承載的大小 (以位元組為單位)。
- 「f」是來源檔案索引的 ID。
- 「o」是區塊位移。
- 「n」是區塊數量。
- 「b」是從位移開始遺失的區塊點陣圖。此值必須是 based64 編碼。

處理 AWS IoT MQTT 型檔案交付的錯誤

對 DescribeStream 和 GetStream 兩個 API 傳送至裝置的錯誤回應包含用戶端字符、錯誤碼及錯誤訊息。典型的錯誤回應看似下列範例所示。

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- "o" 為指出錯誤發生原因的錯誤代碼。如需詳細資訊，請參閱本節稍後的錯誤碼。
- "m" 為包含錯誤詳細資料的錯誤訊息。
- "c" 為用戶端字符欄位。若其提供於 DescribeStream 請求中，則可能會傳回。您可使用用戶端字符，使回應與其請求產生關聯。

  用戶端字符欄位不一定會包含在錯誤回應中。當提供於請求中的用戶端字符無效或格式錯誤時，其不會傳回錯誤回應。

> **Note**
>
> 若為回溯相容性，錯誤回應中的欄位可能為非縮寫形式。例如，錯誤代碼可能由「代碼」或 "o" 欄位進行指定，用戶端字符欄位可由 "clientToken" 或 "c" 欄位進行指定。我們建議您使用上述縮寫形式。

InvalidTopic

串流訊息的 MQTT 主題是無效的。

InvalidJson

串流請求不是一個有效的 JSON 文件。

InvalidCbor

串流請求不是有效的 CBOR 文件。

InvalidRequest

該請求通常被標識為格式錯誤。如需詳細資訊，請參閱錯誤訊息。

未經授權

該請求無權存取儲存媒體 (例如 Amazon S3) 中的串流資料檔案。如需詳細資訊，請參閱錯誤訊息。

BlockSizeOutOfBounds

區塊大小超出範圍。請參閱 AWS IoT Core Service Quotas 中的「MQTT 型檔案交付」一節。

OffsetOutOfBounds

偏移超出範圍。請參閱 AWS IoT Core Service Quotas 中的「MQTT 型檔案交付」一節。

BlockCountLimitExceeded

請求區塊的數量超出範圍。請參閱 AWS IoT Core Service Quotas 中的「MQTT 型檔案交付」一節。

BlockBitmapLimitExceeded

請求點陣圖的大小超出範圍。請參閱 AWS IoT Core Service Quotas 中的「MQTT 型檔案交付」一節。

ResourceNotFound

找不到請求的串流、檔案、檔案版本或區塊。如需詳細資訊，請參閱錯誤訊息。

VersionMismatch

請求中的串流版本與 MQTT 型檔案交付功能中的串流版本不相符。這表示自裝置最初接收串流版本以來，串流資料已遭修改。

ETagMismatch

串流中的 S3 ETag 與最新 S3 物件版本的 ETag 不相符。

InternalError

於 MQTT 型檔案交付時發生內部錯誤。

FreeRTOS OTA 中的範例使用案例

FreeRTOS OTA (over-the-air代理程式使用 AWS IoT MQTT 型檔案交付，將 FreeRTOS 韌體映像傳輸至 FreeRTOS 裝置。若要將初始資料集傳送至裝置，它會使用 AWS IoT 任務服務將 OTA 更新任務排程至 FreeRTOS 裝置。

如需 MQTT 型檔案交互用戶端的參考實作，請參閱 FreeRTOS 文件中的 FreeRTOS OTA 代理程式程式碼。

Device Advisor

Device Advisor 是一種以雲端為基礎的全受管測試功能，可在裝置軟體開發期間驗證 IoT 裝置。Device Advisor 提供預先建置的測試，可讓您在將裝置部署到生產環境之前 AWS IoT Core，用來驗證 IoT 裝置是否有可靠且安全的連線能力。Device Advisor 的預先建置測試可協助您針對最佳實務驗證您的裝置軟體，是否可以使用 TLS、MQTT、Device Shadow，以及 IoT 任務。您也可以下載已簽署的資格報告，提交給 AWS 合作夥伴網路，讓您的裝置符合 AWS Partner Device Catalog 的資格，無需將裝置送入其中並等待裝置進行測試。

> **Note**
>
> us-east-1、us-west-2、ap-northeast-1 和 eu-west-1 區域中支援 Device Advisor。
> Device Advisor 支援使用 MQTT 和透過 WebSocket Secure (WSS) 的 MQTT 通訊協定發佈和訂閱訊息的裝置和用戶端。所有通訊協定皆支援 IPv4 和 IPv6。
> Device Advisor 支援 RSA 伺服器憑證。

任何已建置為連接到 的裝置 AWS IoT Core 都可以利用 Device Advisor。您可以從 AWS IoT 主控台或使用 AWS CLI 或 SDK 存取 Device Advisor。當您準備好測試裝置時，請向 註冊裝置， AWS IoT Core 並使用 Device Advisor 端點設定裝置軟體。然後選擇預先建置的測試、設定它們、在您裝置上執行這些測試，並取得測試結果以及詳細的日誌或資格報告。

Device Advisor 是 AWS 雲端中的測試端點。您可以測試裝置，方法為將它們設定為連接至 Device Advisor 提供的測試端點。將裝置設定為連線至測試端點後，您可以造訪 Device Advisor 的主控台，或使用 AWS SDK 選擇您想要在裝置上執行的測試。Device Advisor 接著會管理測試的完整生命週期，包括佈建資源、排程測試程序、管理狀態機器、記錄裝置行為、記錄結果，以及以測試報告的形式提供最終結果。

TLS 通訊協定

Transport Layer Security (TLS) 通訊協定用來加密通過不安全網路 (例如網際網路) 的機密資料。TLS 通訊協定的前身是 Secure Sockets Layer (SSL) 通訊協定。

Device Advisor 支援下列 TLS 通訊協定：

- TLS 1.3 (建議)
- TLS 1.2

通訊協定、連接埠映射和身分驗證

裝置通訊協定是由裝置或用戶端用來使用裝置端點連線至訊息代理程式。下表列出 Device Advisor 端點支援的通訊協定，以及它們使用的身分驗證方法和連接埠。

通訊協定、身分驗證和連接埠對應

通訊協定	支援的操作	身分驗證	站點	ALPN 通訊協定名稱
MQTT over WebSocket	發佈、訂閱	Signature 第 4 版	443	N/A
MQTT	發佈、訂閱	X.509 用戶端憑證	8883	`x-amzn-mqtt-ca`
MQTT	發佈、訂閱	X.509 用戶端憑證	443	N/A

本章包含下列部分：

- 設定
- 在主控台中開始使用 Device Advisor
- Device Advisor 工作流程
- Device Advisor 詳細主控台工作流程
- 長時間測試主控台工作流程
- Device Advisor VPC 端點 (AWS PrivateLink)
- Device Advisor 測試案例

設定

第一次使用 Device Advisor 之前，請先完成以下任務：

建立 IoT 物件

首先，建立 IoT 物件並將憑證連接至該物件。如需如何建立物件的教學課程，請參閱建立物件。

建立要用作裝置角色的 IAM 角色

> **Note**
>
> 您可以使用 Device Advisor 主控台快速建立裝置角色。若要了解如何使用 Device Advisor 主控台設定裝置角色，請參閱在主控台中開始使用 Device Advisor。

1. 前往 AWS Identity and Access Management 主控台，並登入 AWS 帳戶 您用於 Device Advisor 測試的 。
2. 在左側導覽窗格上，選擇 Policies (政策)。
3. 選擇 Create policy (建立政策)。
4. 在 Create policy (建立政策) 下，執行下列操作：
   a. 針對 Service (服務)，選擇 IoT。
   b. 在動作下，執行下列其中一個動作：
      - (建議) 根據連接至您在上一節中建立之 IoT 物件或憑證的政策選取動作。
      - 在篩選動作方塊中搜尋下列動作，並選取它們：
        - `Connect`
        - `Publish`
        - `Subscribe`
        - `Receive`
        - `RetainPublish`
   c. 在資源下，限制用戶端、主題和主題資源。限制這些資源是安全最佳實務。若要限制資源，請執行下列動作：
      i. 選擇 Specify client resource ARN for the Connect action (指定 Connect 動作的用戶端資源 ARN)。
      ii. 選擇新增 ARN，然後執行下列任一動作：

      > **Note**
      >
      > 此 clientId 是您裝置用來與 Device Advisor 互動的 MQTT 用戶端 ID。

- 在視覺化 ARN 編輯器中指定區域、accountID 和 clientID。
- 手動輸入您想要透過其執行測試案例之 IoT 主題的 Amazon Resource Name (ARN)。

iii. 選擇新增。

iv. 選擇指定 Receive 動作和另外 1 個動作的主題資源 ARN。

v. 選擇新增 ARN，然後執行下列任一動作：

> **Note**
>
> 主題名稱是您裝置將訊息發佈至其中的 MQTT 主題。

- 在視覺化 ARN 編輯器中指定區域、accountID 和主題名稱。
- 手動輸入您想要透過其執行測試案例之 IoT 主題的 ARN。

vi. 選擇 Add (新增)。

vii. 選擇指定 Subscribe 動作的 topicFilter 資源 ARN。

viii. 選擇新增 ARN，然後執行下列任一動作：

> **Note**
>
> 主題名稱是您裝置訂閱的 MQTT 主題。

- 在視覺化 ARN 編輯器中指定區域、accountID 和主題名稱。
- 手動輸入您想要透過其執行測試案例之 IoT 主題的 ARN。

ix. 選擇新增。

5. 選擇下一步：標籤。
6. 選擇下一步：檢閱。
7. 在檢閱政策下，針對您的政策輸入一個名稱。
8. 選擇建立政策。
9. 在左側導覽窗格上，選擇 Roles (角色)。
10. 選擇建立角色。

11. 在選取信任的實體下，選擇自訂信任政策。
12. 在自訂信任政策方塊中輸入下列信任政策。建議您新增全域條件內容金鑰 aws:SourceArn 和 aws:SourceAccount 至政策，保護自己免受混淆代理人問題的困擾。

> **Important**
>
> 您的 aws:SourceArn 必須與 format: arn:aws:iotdeviceadvisor:*region:account-id*:*. 相符。請確認 *region* 與您的 AWS IoT 區域符合，且 *account-id* 與您的客戶帳戶 ID 相符。如需詳細資訊，請參閱預防跨服務混淆代理人。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAwsIoTCoreDeviceAdvisor",
            "Effect": "Allow",
            "Principal": {
                "Service": "iotdeviceadvisor.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:iotdeviceadvisor:*:123456789012:suitedefinition/*"
            }
        }
        }
    ]
}
```

13. 選擇下一步。
14. 選擇您在步驟 4 所建立的政策。
15. (選用) 在設定許可界限下，選擇使用許可界限來控制角色許可上限，然後選取您已建立的政策。
16. 選擇下一步。

17. 輸入 Role name (角色名稱) 和 Role description (角色描述)。
18. 選擇 Create Role (建立角色)。

為要使用 Device Advisor 的 IAM 使用者建立自訂受管政策

1. 導覽至 IAM 主控台，網址為 https://console.aws.amazon.com/iam/。若出現提示，請輸入 AWS 憑證以進行登入。
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 選擇 Create Policy (建立政策)，然後選擇 JSON 標籤。
4. 新增必要的許可以使用 Device Advisor。政策文件可在安全最佳實務主題中找到。
5. 選擇檢閱政策。
6. 輸入 Name (名稱) 和 Description (描述)。
7. 選擇 Create Policy (建立政策)。

建立要使用 Device Advisor 的 IAM 使用者

> **Note**
>
> 我們建議您建立要在執行 Device Advisor 測試時使用的 IAM 使用者。從管理員使用者執行 Device Advisor 測試可能會造成安全風險，因此不建議此操作。

1. 導覽至位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台；若出現提示，請輸入 AWS 憑證以進行登入。
2. 在左側導覽窗格中，選擇 Users (使用者)。
3. 選擇 Add User (新增使用者)。
4. 輸入 User name (使用者名稱)。
5. 如果使用者想要與 AWS 外部互動，則需要程式設計存取 AWS 管理主控台。授予程式設計存取權的方式取決於正在存取的使用者類型 AWS。

   若要授予使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	根據
IAM	（建議）使用主控台登入資料做為臨時登入資料，以簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的登入以進行 AWS 本機開發。 • AWS SDKs，請參閱 AWS SDKs 和工具參考指南中的登入以進行 AWS 本機開發。
人力資源身分 (IAM Identity Center 中管理的使用者)	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的設定 AWS CLI 要使用 AWS IAM Identity Center的 。 • AWS SDKs、工具和 AWS APIs，請參閱 AWS SDKs 和工具參考指南中的 IAM Identity Center 身分驗證。
IAM	使用暫時登入資料簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	遵循《IAM 使用者指南》中將臨時登入資料與 AWS 資源搭配使用的指示。

哪個使用者需要程式設計存取權？	到	根據
IAM	(不建議使用) 使用長期憑證簽署對 AWS CLI、 AWS SDKs程式設計請求。 AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《 使用者指南》中的使用 IAM 使用者憑證進行身分驗證。 • AWS SDKs和工具，請參閱 AWS SDKs和工具參考指南中的使用長期憑證進行身分驗證。 • 對於 AWS APIs，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

6. 選擇下一步：許可。
7. 若要提供存取權，請新增權限至您的使用者、群組或角色：
   - 中的使用者和群組 AWS IAM Identity Center：

     建立權限合集。請按照《AWS IAM Identity Center 使用者指南》中的建立權限合集說明進行操作。
   - 透過身分提供者在 IAM 中管理的使用者：

     建立聯合身分的角色。遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。
   - IAM 使用者：
     - 建立您的使用者可擔任的角色。請按照《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
     - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台) 中的指示。
8. 在搜尋方塊中，輸入您已建立之客戶管理政策的名稱。然後，選取政策名稱的核取方塊。

9. 選擇下一步：標籤。
10. 選擇 Next:Review (下一步：檢閱)。
11. 選擇 Create user (建立使用者)。
12. 選擇關閉。

Device Advisor 需要代表您存取您的 AWS 資源 （ 物件、憑證和端點 ）。您的 IAM 使用者必須擁有必要的許可。如果您將必要的許可政策連接到 IAM 使用者，Device Advisor 也會將記錄發佈到 Amazon CloudWatch。

設定您的裝置

Device Advisor 會使用伺服器名稱指示 (SNI) TLS 延伸來套用 TLS 組態。在裝置連線並傳遞與 Device Advisor 測試端點相同的伺服器名稱時，它們必須使用此延伸。

當測試處於 Running 狀態時，Device Advisor 允許 TLS 連線。它會在每次測試執行的前後拒絕 TLS 連線。基於這個原因，我們建議您使用裝置連線重試機制，以取得 Device Advisor 的全自動化測試體驗。您可以執行測試套件，其中包含多個測試案例，例如 TLS 連線、MQTT 連線和 MQTT 發佈。如果您執行多個測試案例，我們建議您的裝置每五秒嘗試連線到我們的測試端點。然後，您可以使多個測試案例依序自動執行。

> **Note**
>
> 若要準備好您的裝置軟體進行測試，我們建議您使用可以連線至 AWS IoT Core的 SDK。然後，您應該使用為 AWS 帳戶提供的 Device Advisor 測試端點來更新 SDK。

Device Advisor 支援兩種類型的端點：帳戶層級和裝置層級端點。選擇最適合您使用案例的端點。若要針對不同裝置同時執行多個測試套件，請使用裝置層級端點。

請執行下列命令來取得裝置層級端點：

若為使用 X.509 用戶端憑證的 MQTT 客戶：

```
aws iotdeviceadvisor get-endpoint --thing-arn your-thing-arn
```

或

```
aws iotdeviceadvisor get-endpoint --certificate-arn your-certificate-arn
```

若為使用簽章第 4 版的 MQTT over WebSocket 客戶：

```
aws iotdeviceadvisor get-endpoint --device-role-arn your-device-role-arn --
authentication-method SignatureVersion4
```

若要一次執行一個測試套件，請選擇 Account-level endpoint (帳戶層級端點)。請執行下列命令來取得帳戶層級端點：

```
aws iotdeviceadvisor get-endpoint
```

在主控台中開始使用 Device Advisor

本教學課程可協助您 AWS IoT Core Device Advisor 在 主控台上開始使用 。Device Advisor 提供必要測試和已簽署的資格報告等功能。您可以使用這些測試和報告，來限定並列出 AWS Partner Device Catalog 中的裝置，如 AWS IoT Core 資格計畫所詳述。

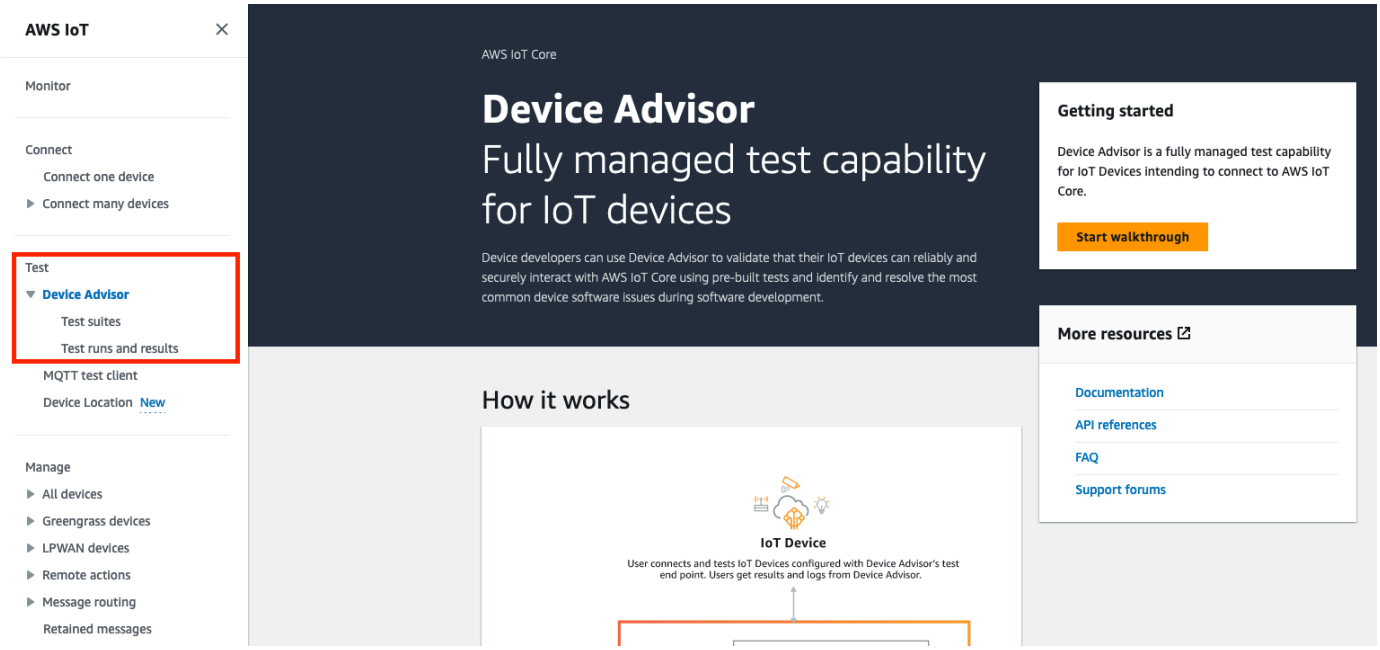
如需使用 Device Advisor 的詳細資訊，請參閱 Device Advisor 工作流程 和 Device Advisor 詳細主控台工作流程。

若要完成此教學課程，請遵循 設定 中概述的步驟。

> **Note**
>
> 下列支援 Device Advisor AWS 區域：
>
> - 美國東部 (維吉尼亞北部)
> - 美國西部 (奧勒岡)
> - 亞太地區 (東京)
> - 歐洲 (愛爾蘭)

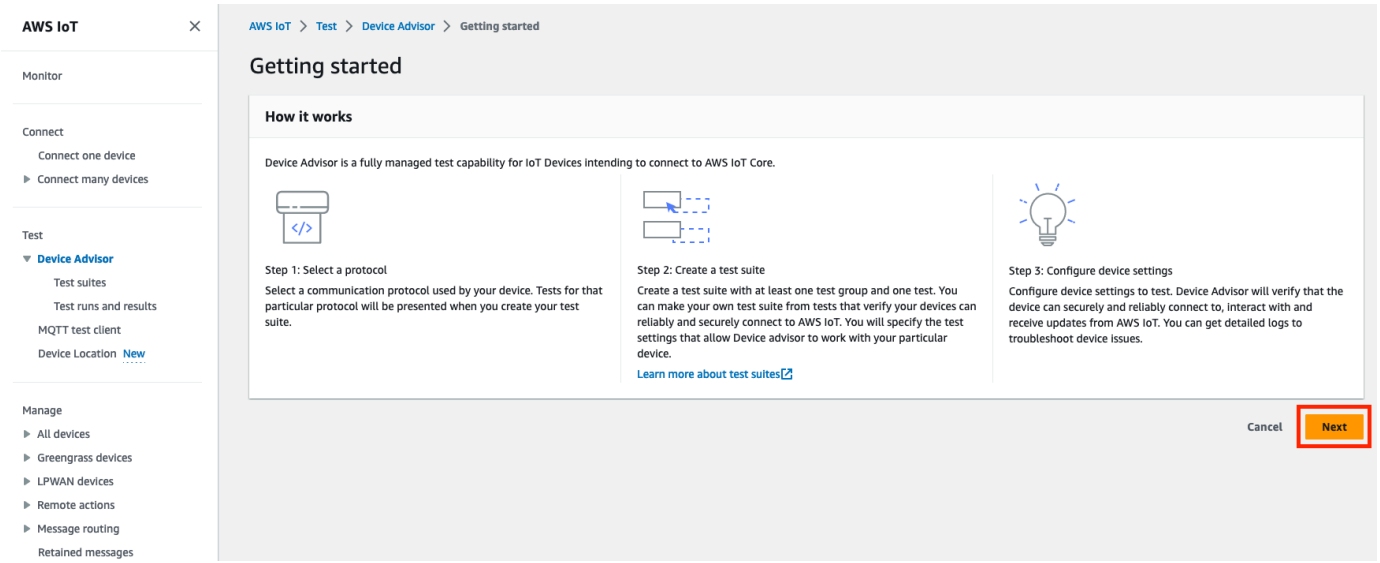
開始使用

1. 在 AWS IoT 主控台導覽窗格的測試下，選擇 Device Advisor。然後，選擇主控台上的開始演練按鈕。

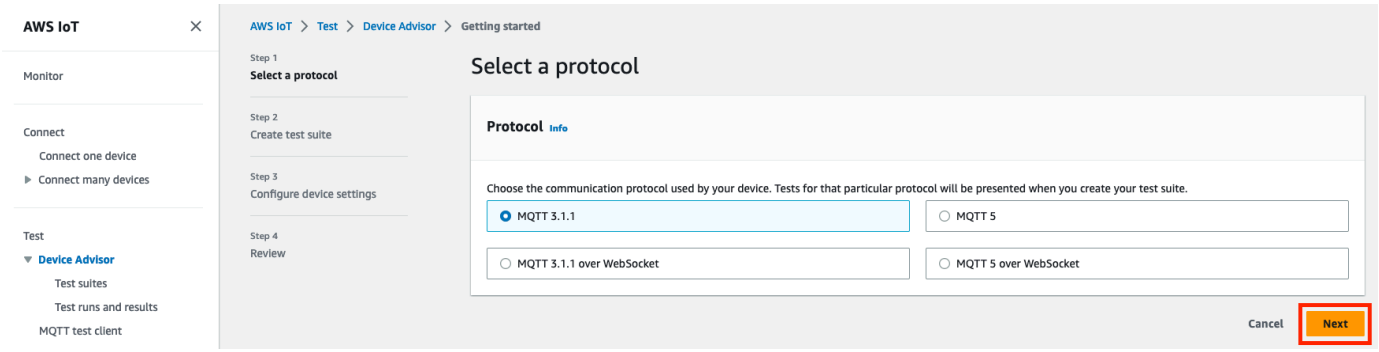


2. 開始使用 Device Advisor 頁面概述建立測試套件和針對您裝置執行測試所需的步驟。您也可以在這裡針對您的帳戶尋找 Device Advisor 測試端點。您必須在用於測試的裝置上設定韌體或軟體，才能連線到此測試端點。

   若要完成本教學課程，請先建立物件和憑證。在您檢閱運作方式下的資訊之後，請選擇下一步。

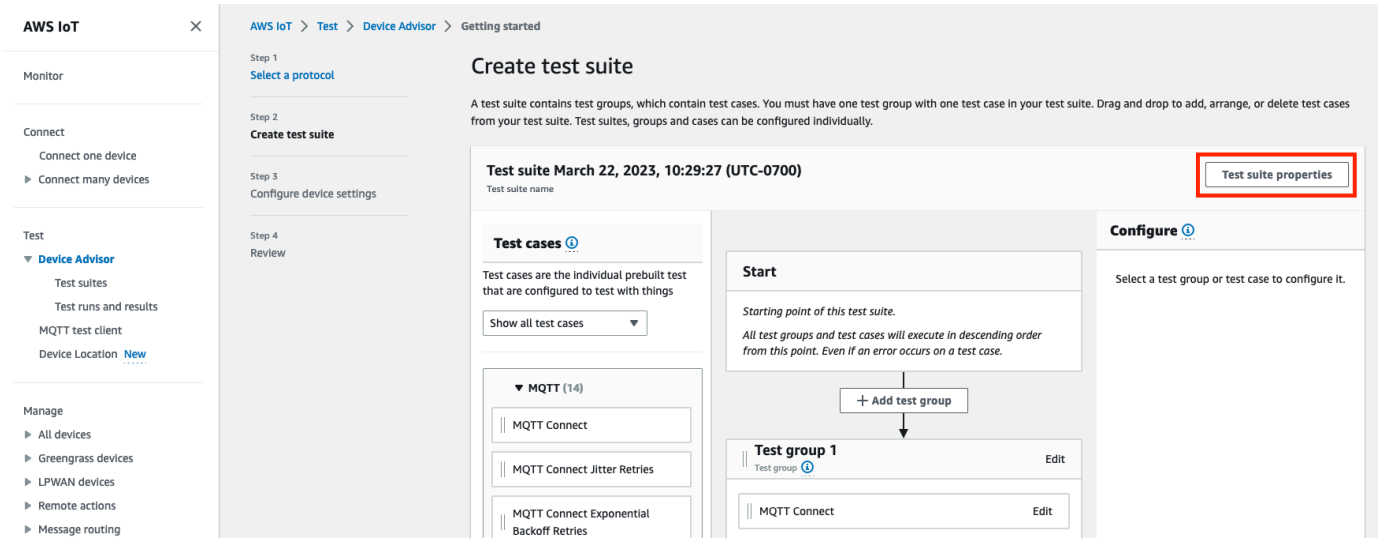


3. 在步驟 1：選取通訊協定中，從列出的選項中選取通訊協定。然後選擇下一步。



4. 在 Step 2 (步驟 2) 中，您可以建立和設定自訂測試套件。一個自訂測試套件必須至少具有一個測試群組，而且每個測試群組必須至少具有一個測試案例。我們已新增 MQTT Connect 測試案例，供您開始使用。

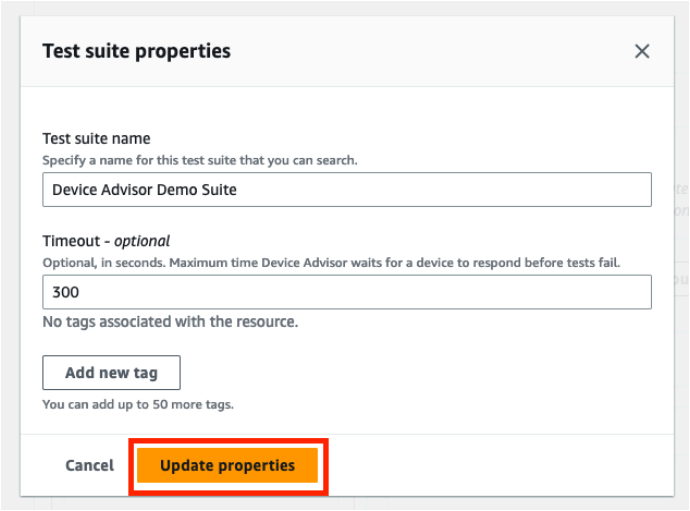
   選擇 Test suite properties (測試套件屬性)。



   當您建立測試套件時，請提供測試套件屬性。您可以設定以下套件層級屬性：

   - Test suite name (測試套件名稱)：輸入測試套件名稱。
   - 逾時 (選用)：目前測試套件中每個測試案例的逾時 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。
   - Tags (標籤) (選用)：將標籤新增至測試套件。

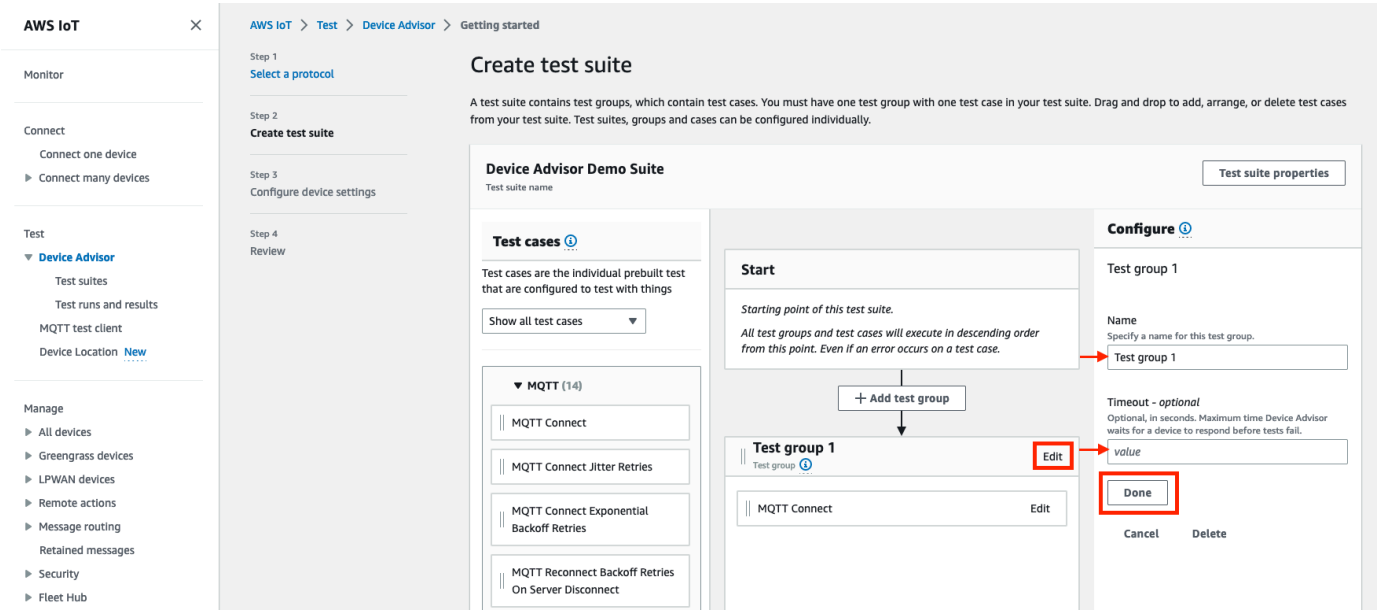
   完成後，請選擇 Update properties (更新屬性)。



5. (選用) 若要更新測試套件群組組態，請選擇測試群組名稱旁邊的編輯按鈕。

   - Name (名稱)：輸入測試套件群組的自訂名稱。
   - 逾時 (選用)：目前測試套件中每個測試案例的逾時 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。

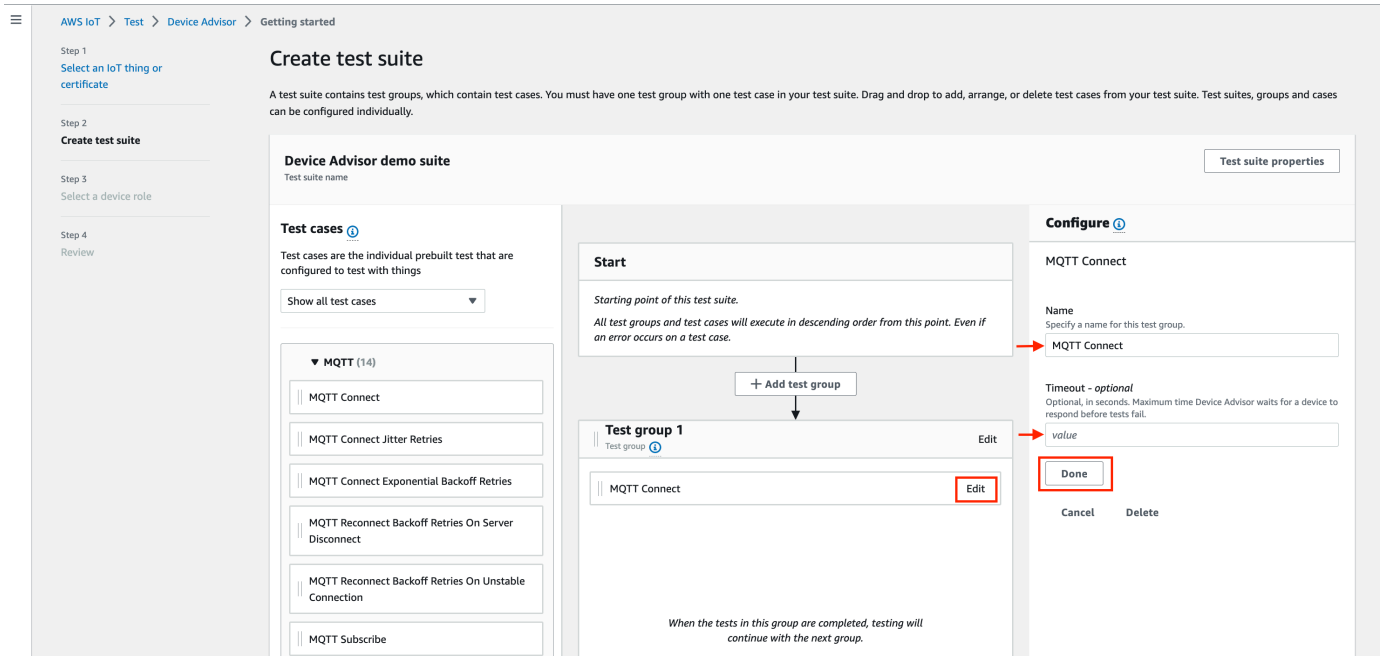
   完成時，請選擇完成以繼續。



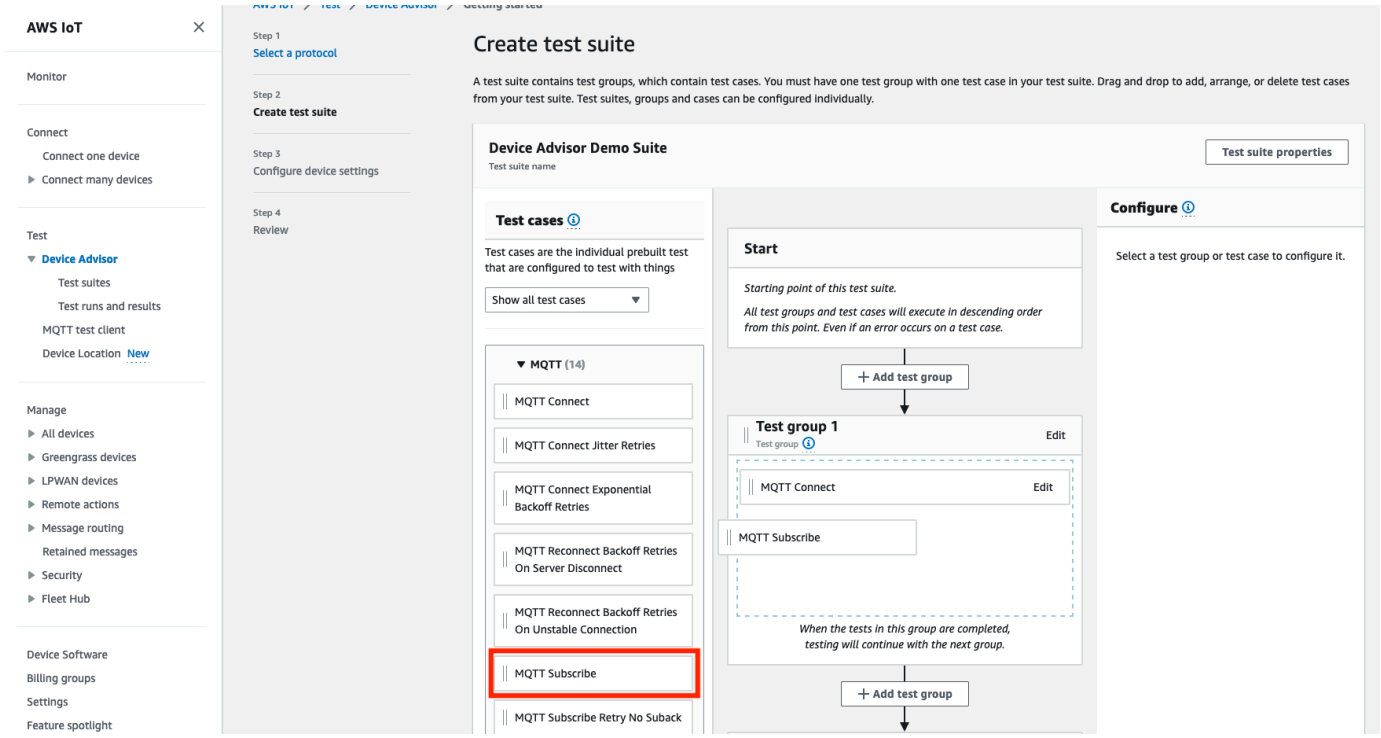
6. (選用) 若要更新測試案例的測試案例組態，請選擇測試案例名稱旁邊的編輯按鈕。

   - Name (名稱)：輸入測試套件群組的自訂名稱。
   - 逾時 (選用)：所選測試案例的逾時 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。

完成時，請選擇完成以繼續。



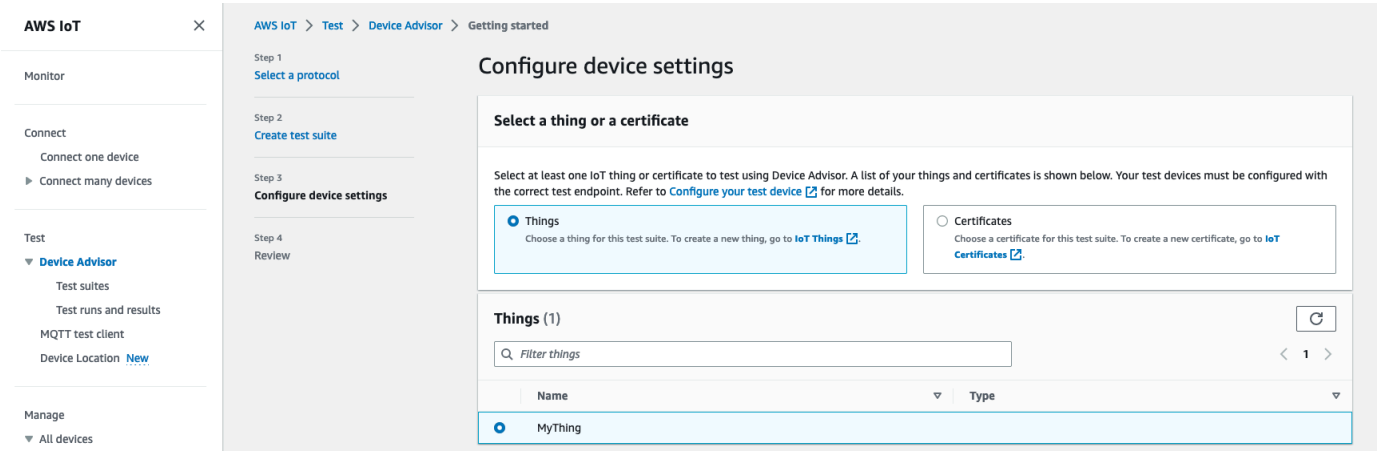
7. (選用) 若要將更多測試群組新增至測試套件，請選擇新增測試群組，然後遵循步驟 5 中的指示。
8. (選用) 若要新增更多測試案例，請將測試案例區段中的測試案例拖曳至到您的任何測試群組。



9. 您可以變更測試群組和測試案例的順序。若要進行變更，請在清單中將列出的測試案例向上或向下拖曳。Device Advisor 會依照您列出測試的順序執行這些測試。

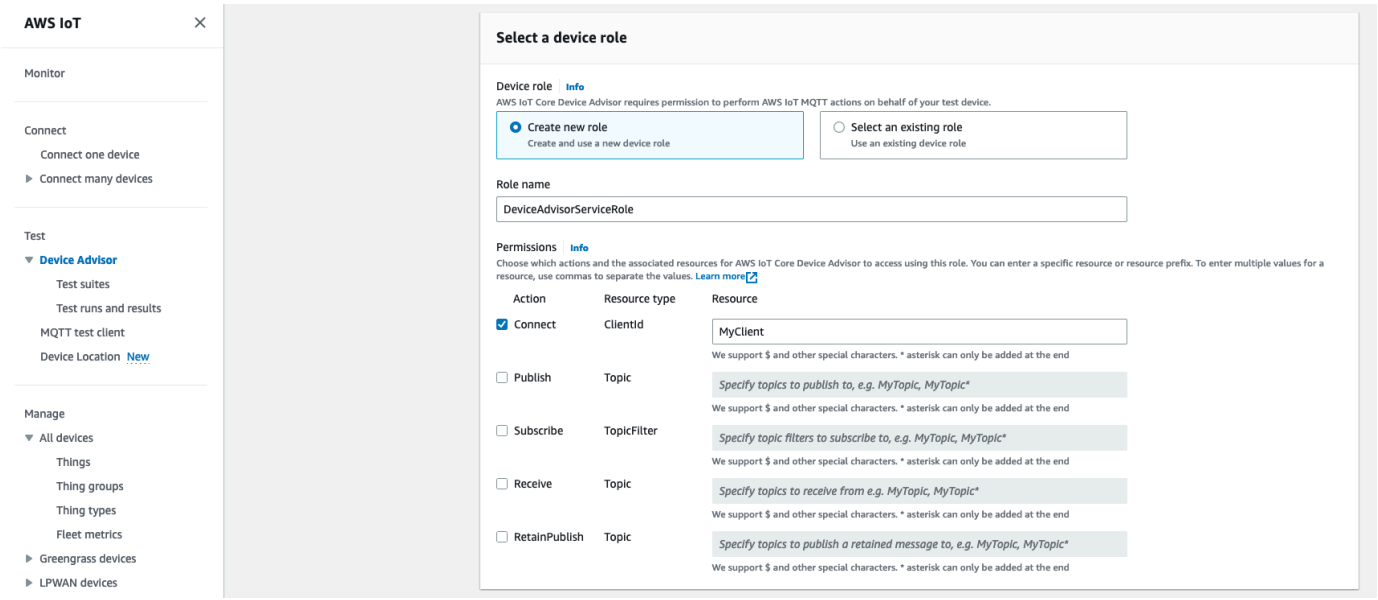
在設定了您的測試套件之後，請選擇 Next (下一步)。

10. 在步驟 3 中，選取要使用 Device Advisor 測試的 AWS IoT 物件或憑證。如果沒有任何現有物件或憑證，請參閱設定。

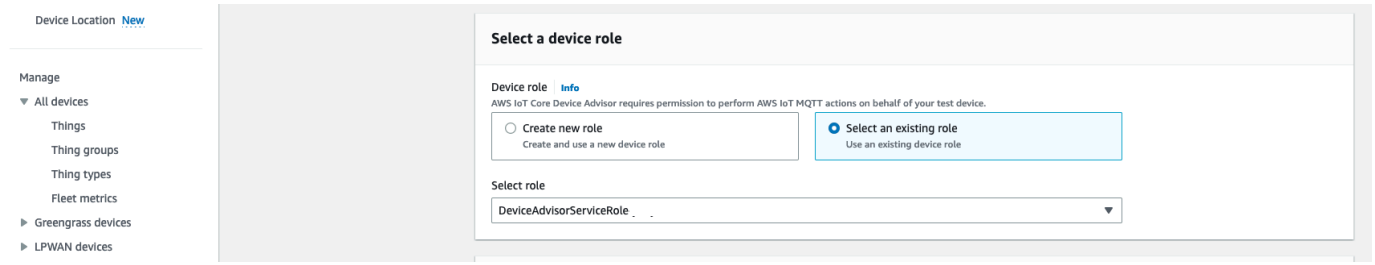


11. 您可以設定 Device Advisor 用來代表測試裝置執行 AWS IoT MQTT 動作的裝置角色。僅針對 MQTT Connect 測試案例，才會自動選取 Connect 動作。這是因為裝置角色需要此許可才能執行測試套件。對於其他測試案例，系統會選取對應的動作。

    提供每個選取動作的資源值。例如，對於 Connect 動作，請提供裝置用來連線至 Device Advisor 端點的用戶端 ID。您可以使用逗號分隔值提供多個值，並以萬用字元 (*) 作為這些值的字首。例如，若要對以 `MyTopic` 為開頭的任何主題提供發佈許可，請輸入 **`MyTopic*`** 作為資源值。

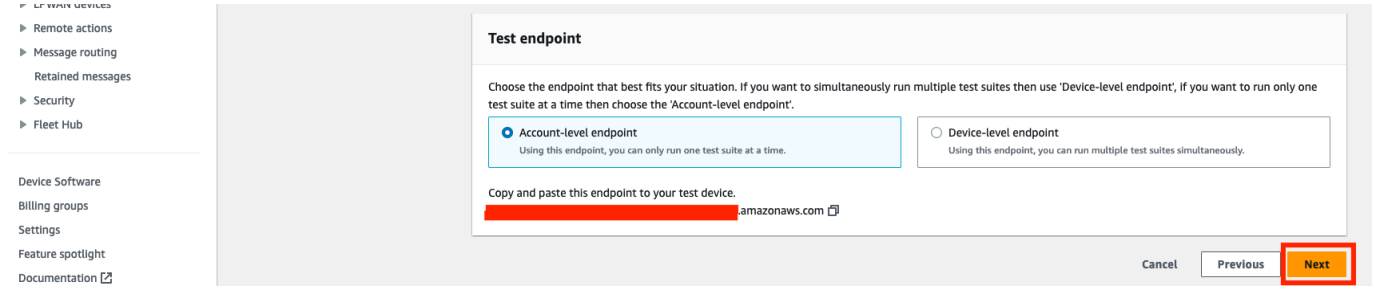


    若要從設定使用先前建立的裝置角色，請選擇選取現有角色。然後在選取角色下選擇您的裝置角色。



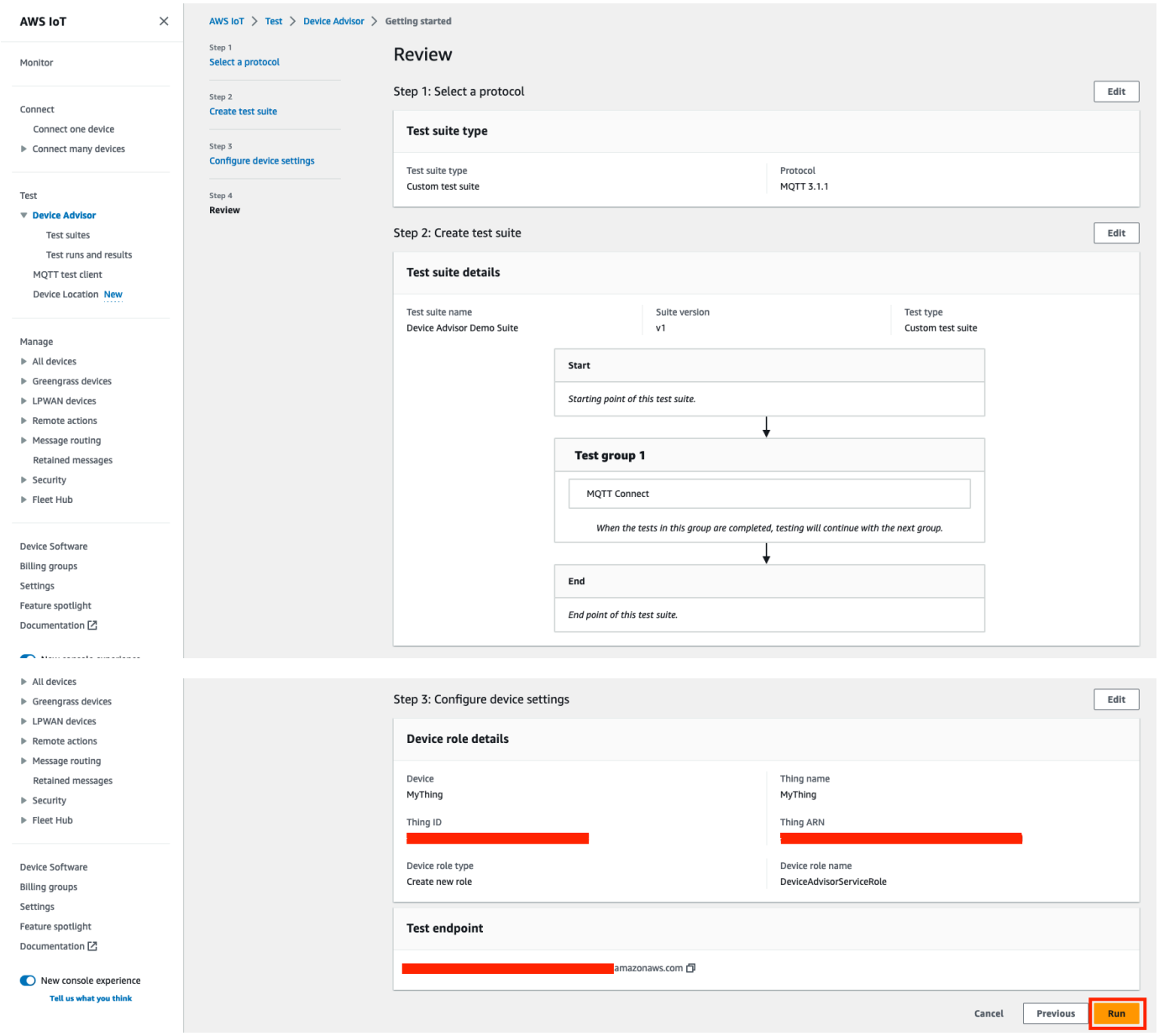
使用提供的兩個選項之一來設定裝置角色，然後選擇下一步。

12. 在測試端點區段中，選取最適合使用案例的端點。若要使用相同的 同時執行多個測試套件 AWS 帳戶，請選取裝置層級端點。若要一次執行一個測試套件，請選取帳戶層級端點。

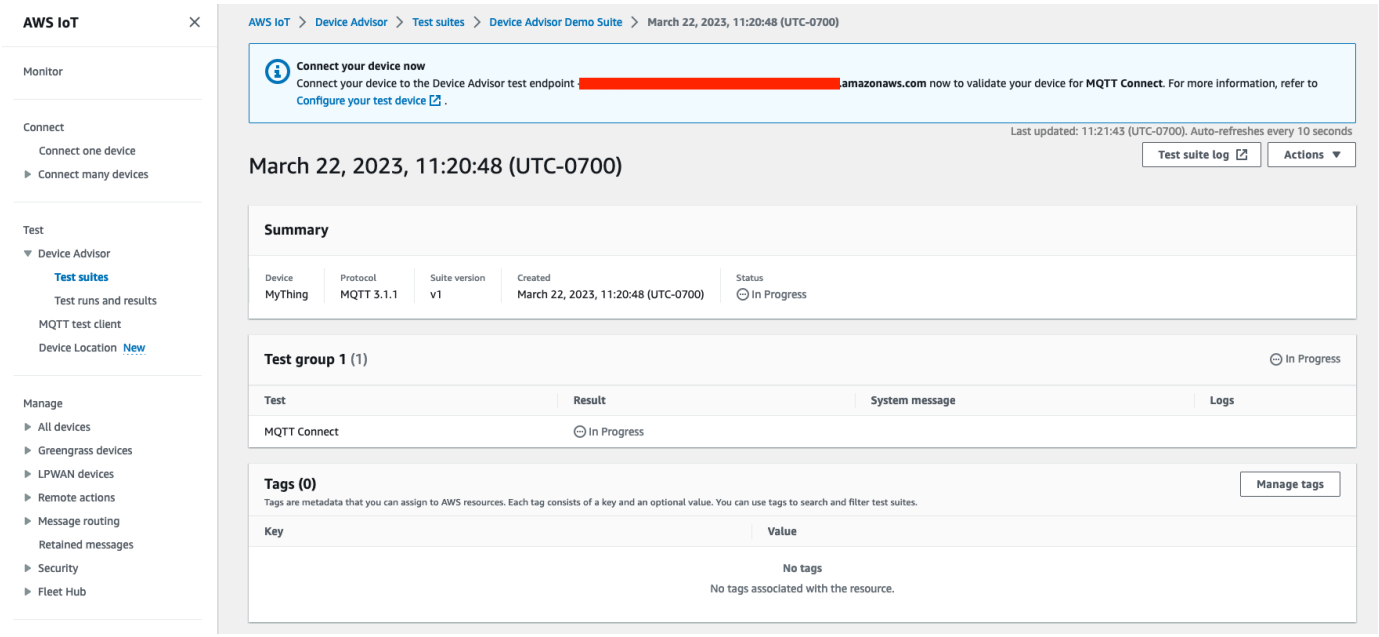


13. 步驟 4 顯示所選測試裝置、測試端點、測試套件，以及所設定之測試裝置角色的概觀。若要對區段進行變更，請為要編輯的區段選擇編輯按鈕。一旦確認了您的測試組態，請選擇執行來建立測試套件並執行您的測試。

> **Note**
>
> 如需最佳結果，您可以在開始執行測試套件之前，將選取的測試裝置連線至 Device Advisor 測試端點。我們建議您為裝置建置一個機制，每隔五秒嘗試連接到我們的測試端點，最多持續一到兩分鐘。



14. 在導覽窗格的測試下，選擇 Device Advisor，然後選擇測試執行和結果。選取測試套件執行，以檢視其執行詳細資訊和記錄。



15. 若要存取套件執行的 Amazon CloudWatch Logs：

    - 選擇 Test suite log (測試套件記錄)，以檢視測試套件執行的 CloudWatch 記錄。
    - 選擇任何測試案例的 Test case log (測試案例記錄)，以檢視測試案例特定的 CloudWatch 記錄。

16. 根據您的測試結果，疑難排解您的裝置，直到所有測試都通過。

Device Advisor 工作流程

本教學課程說明如何建立自訂測試套件，以及針對您要在主控台中測試的裝置執行測試。在測試完成之後，您可以檢視測試結果和詳細記錄。

先決條件

開始本教學課程之前，請先完成設定中概述的步驟。

建立測試套件定義

首先，安裝 AWS SDK。

`rootGroup` 語法

根群組是 JSON 字串，指定哪些測試案例包含在您的測試套件中。它也會針對這些測試案例指定任何必要的組態。使用根群組根據您的需求建構和排序您的測試套件。測試套件的階層如下：

```
test suite # test group(s) # test case(s)
```

一個測試套件必須至少具有一個測試群組，而且每個測試群組必須至少具有一個測試案例。Device Advisor 會依您定義測試群組和測試案例的順序執行測試。

每個根群組都遵循這個基本結構：

```
{
    "configuration": {  // for all tests in the test suite
        "": ""
    }
    "tests": [{
        "name": ""
        "configuration": {  // for all sub-groups in this test group
            "": ""
        },
        "tests": [{
            "name": ""
            "configuration": {  // for all test cases in this test group
                "": ""
            },
            "test": {
                "id": ""
                "version": ""
            }
        }]
    }]
}
```

在根群組中，您可以使用 `name`、`configuration`，以及群組包含的 `tests` 來定義測試套件。`tests` 群組包含個別測試的定義。您可以使用 `name`、`configuration`，以及針對該測試定義測試案例的 `test` 區塊來定義每個測試。最後，每個測試案例都是使用 `id` 和 `version` 定義的。

如需如何為每個測試案例 (`test` 區塊) 使用 `"id"` 和 `"version"` 欄位的相關資訊，請參閱 Device Advisor 測試案例。該節還包含可用 `configuration` 設定的相關資訊。

下列區塊是根群組組態的範例。此組態指定 MQTT Connect Happy Case 和 MQTT Connect Exponential Backoff Retries 測試案例，以及組態欄位的描述。

```
{
```

```
    "configuration": {},  // Suite-level configuration
    "tests": [            // Group definitions should be provided here
      {
        "name": "My_MQTT_Connect_Group",  // Group definition name
        "configuration": {}               // Group definition-level configuration,
        "tests": [                        // Test case definitions should be provided
 here
        {
            "name": "My_MQTT_Connect_Happy_Case",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 300        // Test case definition-level
 configuration, in seconds
            },
            "test": {
                "id": "MQTT_Connect",              // test case id
                "version": "0.0.0"                 // test case version
            }
        },
        {
            "name": "My_MQTT_Connect_Jitter_Backoff_Retries",  // Test case definition
 name
            "configuration": {
                "EXECUTION_TIMEOUT": 600                // Test case definition-level
 configuration,  in seconds
            },
            "test": {
                "id": "MQTT_Connect_Jitter_Backoff_Retries",  // test case id
                "version": "0.0.0"                                  // test case version
            }
        }]
    }]
}
```

您必須在建立測試套件定義時提供根群組組態。儲存回應物件中傳回的 suiteDefinitionId。您可以使用此 ID 擷取測試套件定義資訊，以及執行測試套件。

以下是一個 Java 開發套件範例：

```
response = iotDeviceAdvisorClient.createSuiteDefinition(
        CreateSuiteDefinitionRequest.builder()
            .suiteDefinitionConfiguration(SuiteDefinitionConfiguration.builder()
                .suiteDefinitionName("your-suite-definition-name")
                .devices(
```

```
                    DeviceUnderTest.builder()
                        .thingArn("your-test-device-thing-arn")
                        .certificateArn("your-test-device-certificate-arn")
                        .deviceRoleArn("your-device-role-arn") //if using SigV4 for
 MQTT over WebSocket
                        .build()
                )
                .rootGroup("your-root-group-configuration")
                .devicePermissionRoleArn("your-device-permission-role-arn")
                .protocol("MqttV3_1_1 || MqttV5 || MqttV3_1_1_OverWebSocket ||
 MqttV5_OverWebSocket")
                .build()
            )
            .build()
)
```

取得測試套件定義

在建立您的測試套件定義之後，您會在 CreateSuiteDefinition API 操作的回應物件中收到 suiteDefinitionId。

當此操作傳回 suiteDefinitionId 時，您可能會在每個群組內看到新的 id 欄位，以及在根群組內看到測試案例定義。您可以使用這些 ID 來執行測試套件定義的子集。

Java 開發套件範例：

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
)
```

取得測試端點

使用 GetEndpoint API 操作取得裝置所使用的測試端點。選取最適合您測試的端點。若要同時執行多個測試套件，請透過提供 thing ARN、certificate ARN 或 device role ARN 來使用裝置層級端點。若要執行單一測試套件，請不提供引數給 GetEndpoint 操作，以選擇帳戶層級端點。

開發套件範例：

```
response = iotDeviceAdvisorClient.getEndpoint(GetEndpointRequest.builder()
```

```
.certificateArn("your-test-device-certificate-arn")
.thingArn("your-test-device-thing-arn")
.deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

.build())
```

啟動測試套件執行

在建立了測試套件定義，並將測試裝置設定為連線至 Device Advisor 測試端點之後，請使用 StartSuiteRun API 執行您的測試套件。

對於 MQTT 客戶，請使用 certificateArn 或 thingArn 執行測試套件。當設定這兩者時，如果憑證屬於物件，則會使用此憑證。

對於 MQTT over WebSocket 客戶，請使用 deviceRoleArn 來執行測試套件。如果指定的角色與測試套件定義中指定的角色不同，則指定的角色會覆寫定義的角色。

對於 .parallelRun()，如果使用裝置層級端點，來使用一個 AWS 帳戶平行執行多個測試套件，請使用 true。

開發套件範例：

```
response = iotDeviceAdvisorClient.startSuiteRun(StartSuiteRunRequest.builder()
.suiteDefinitionId("your-suite-definition-id")
.suiteRunConfiguration(SuiteRunConfiguration.builder()
    .primaryDevice(DeviceUnderTest.builder()
        .certificateArn("your-test-device-certificate-arn")
        .thingArn("your-test-device-thing-arn")
        .deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

        .build())
    .parallelRun(true | false)
    .build())
.build())
```

從回應儲存 suiteRunId。您將使用此資訊來擷取此測試套件執行的結果。

取得測試套件執行

在您開始測試套件執行之後，您可以使用 GetSuiteRun API 來檢查其進度及結果。

開發套件範例：

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

停止測試套件執行

若要停止仍在進行的測試套件執行，您可以呼叫 StopSuiteRun API 操作。在您呼叫 StopSuiteRun API 操作之後，服務將會啟動清除程序。當服務執行清除程序時，測試套件執行狀態會更新為 Stopping。清除程序需要幾分鐘的時間來完成。一旦此程序完成，測試套件執行狀態就會更新為 Stopped。在測試執行完全停止之後，您可以啟動另一個測試套件執行。您可以使用 GetSuiteRun API 操作定期檢查套件執行狀態，如上節所示。

開發套件範例：

```
// Using the SDK, call the StopSuiteRun API.

response = iotDeviceAdvisorClient.StopSuiteRun(
StopSuiteRun.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

取得成功資格測試套件執行的資格報告

如果執行成功完成的資格測試套件，則可以使用 GetSuiteRunReport API 操作擷取資格報告。您可以使用此資格報告，搭配 AWS IoT Core 資格計畫限定您的裝置。若要判斷您的測試套件是否為資格測試套件，請檢查 intendedForQualification 參數是否設定為 true。在呼叫 GetSuiteRunReport API 操作之後，您可以從傳回的 URL 下載報告，最長可達 90 秒。如果上次呼叫 GetSuiteRunReport API 操作的經歷時間超過 90 秒，請再次呼叫該操作來擷取新的有效 URL。

開發套件範例：

```
// Using the SDK, call the getSuiteRunReport API.
```

```
response = iotDeviceAdvisorClient.getSuiteRunReport(
    GetSuiteRunReportRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .suiteRunId("your-suite-run-id")
        .build()
)
```

Device Advisor 詳細主控台工作流程

在此教學課程中，您已建立自訂測試套件，並針對您要在主控台中測試的裝置執行測試。在測試完成之後，您可以檢視測試結果和詳細記錄。

教學

- 先決條件
- 建立測試套件定義
- 啟動測試套件執行
- 停止測試套件執行 (選用)
- 檢視測試套件執行詳細資訊和記錄
- 下載 AWS IoT 資格報告

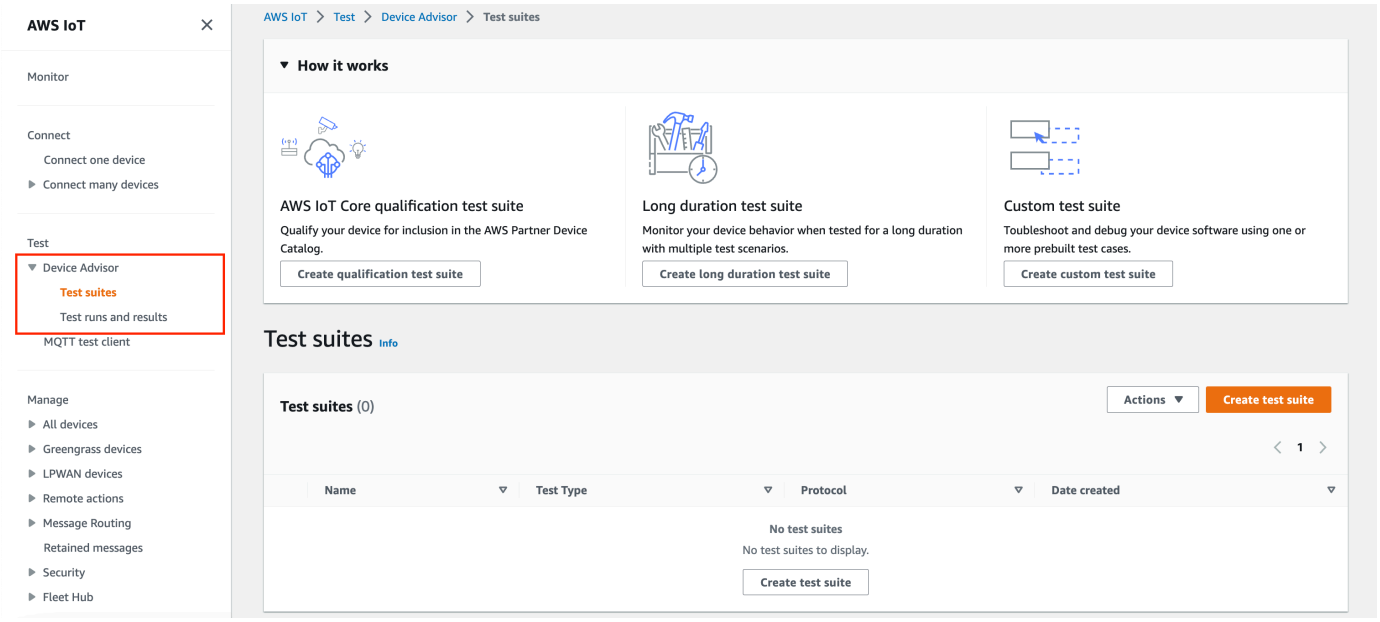
先決條件

為完成此教學課程，您需要建立物件和憑證。

建立測試套件定義

建立測試套件套件，以便您可以為裝置執行它並執行驗證。

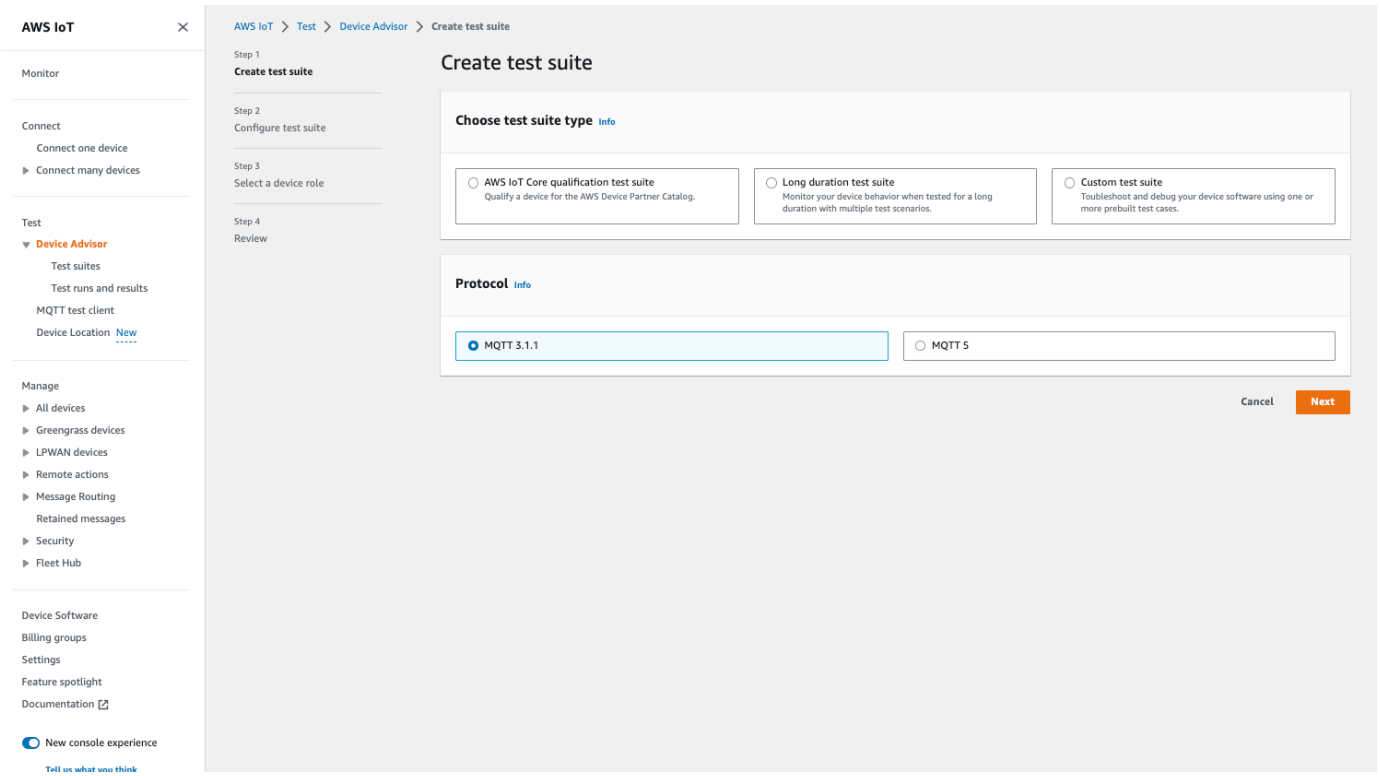
1. 在 AWS IoT 主控台的導覽窗格中，展開 Test (測試)、Device Advisor，然後選擇 Test suites (測試套件)。



選擇 Create Test Suite (建立測試套件)。

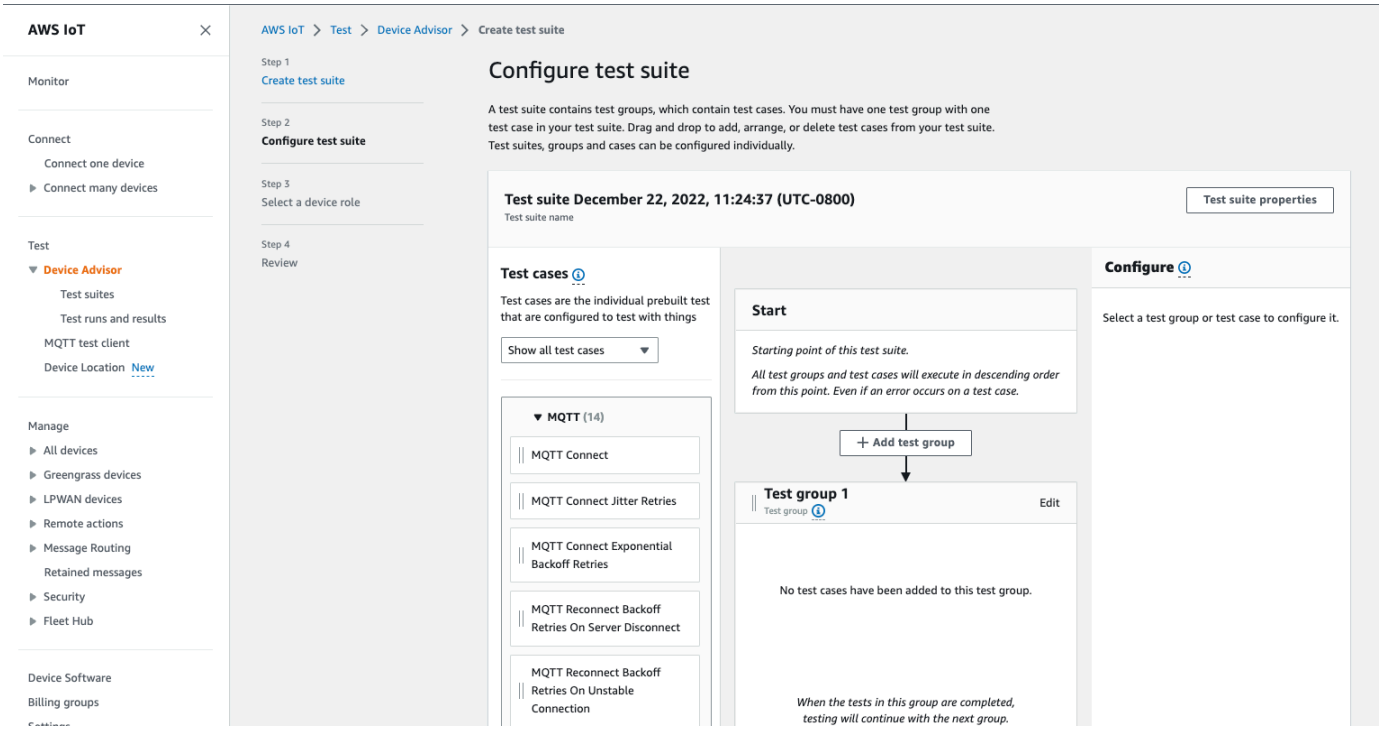
2. 擇一選取 `Use the AWS Qualification test suite` 或者 `Create a new test suite`。

對於通訊協定，請選擇 MQTT 3.1.1 或 MQTT 5。

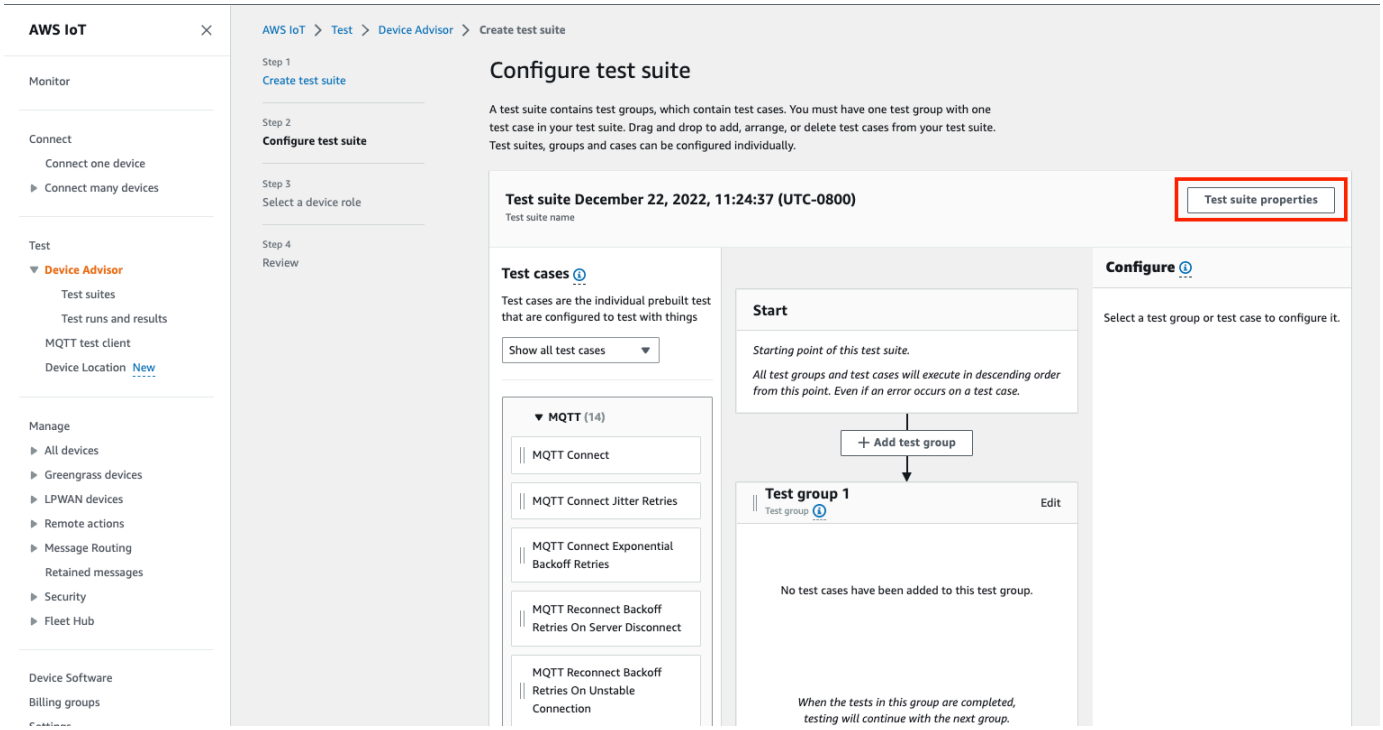


選取 `Use the AWS Qualification test suite`以符合資格，並將您的裝置列出至 AWS 合作夥伴裝置目錄。透過選擇此選項，根據 AWS IoT Core 資格計劃限定您裝置所需的測試案例是預先選取的。無法新增或移除測試群組和測試案例。不過，您仍然需要設定測試套件屬性。

選取 `Create a new test suite` 來建立和設定自訂測試套件。我們建議您從這個選項開始進行初始測試和疑難排解。一個自訂測試套件必須至少具有一個測試群組，而且每個測試群組必須至少具有一個測試案例。基於本教學課程的目的，我們將選取此選項並選擇 Next (下一步)。

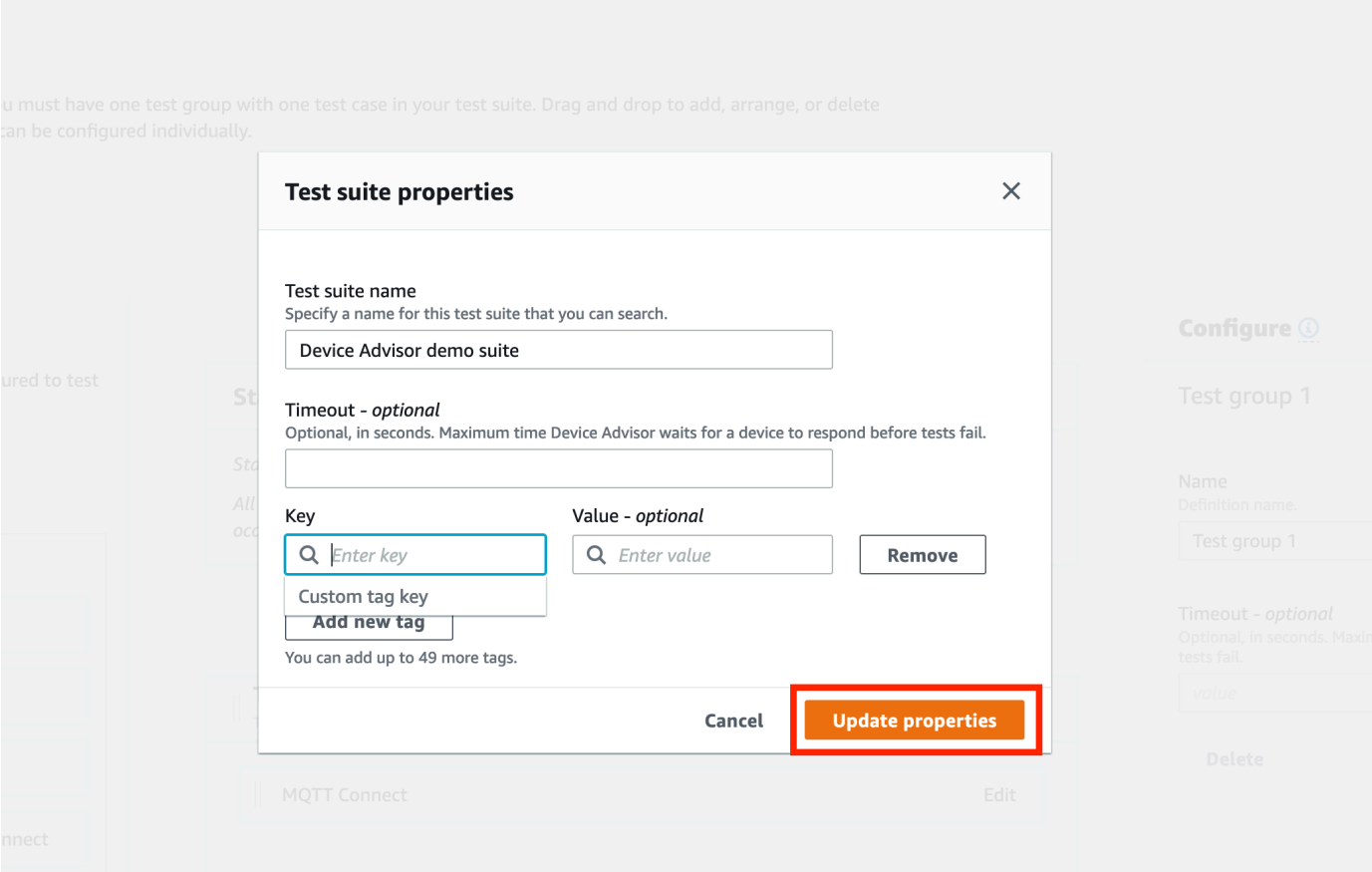


3. 選擇 Test suite properties (測試套件屬性)。必須在建立測試套件時建立測試套件屬性。



在 Test suite properties (測試套件屬性) 下，請填寫下列資訊：

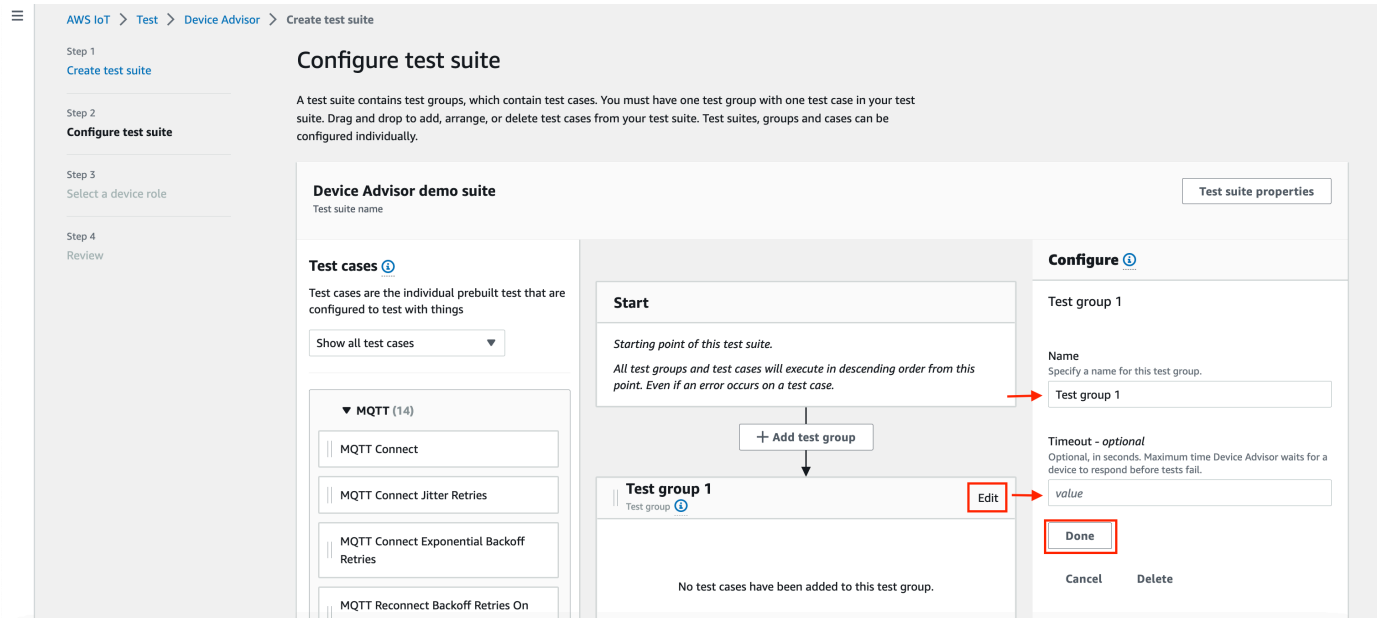
- Test suite name (測試套件名稱)：您可以使用自訂名稱建立套件。
- Timeout (逾時) (選用)：目前測試套件中每個測試案例的逾時 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。
- Tags (標籤) (選用)：將標籤新增至測試套件。



完成後，請選擇 Update properties (更新屬性)。

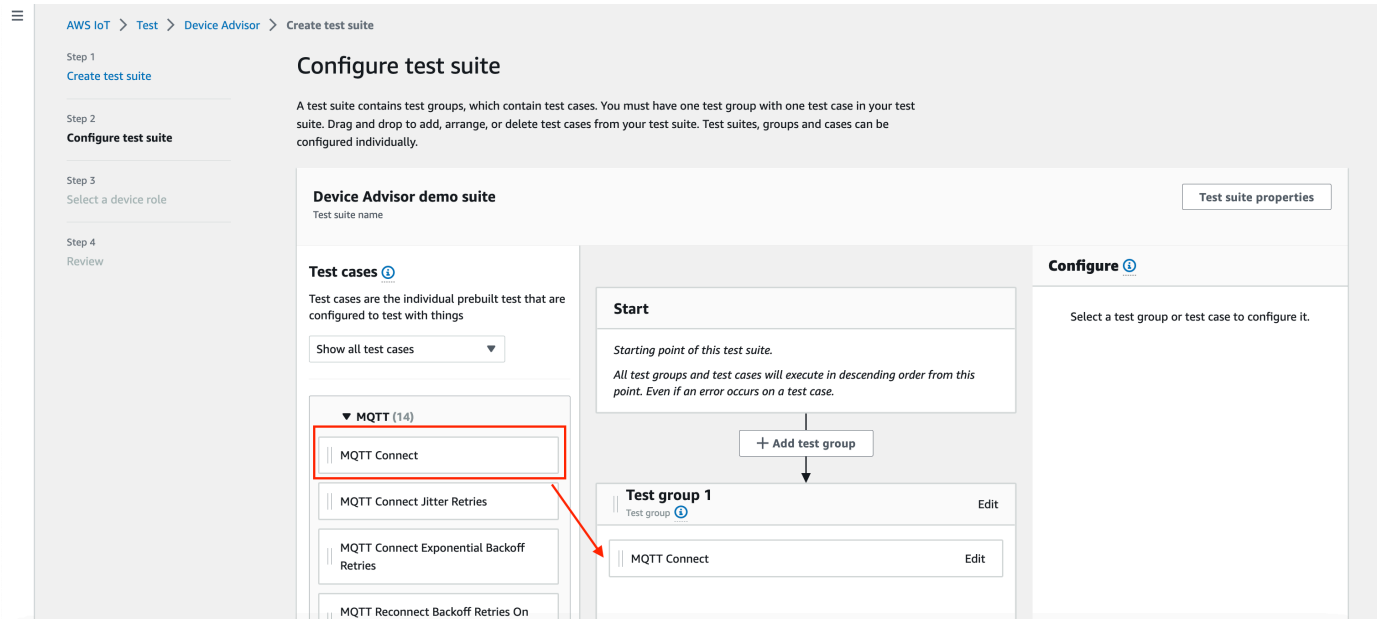
4. 若要修改群組層級組態，請在 `Test group 1` 下，選擇 Edit (編輯)。接著，輸入 Name (名稱)，為群組提供自訂名稱。

   您也可以選擇性地在選取的測試群組下輸入 Timeout (逾時) 值 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。



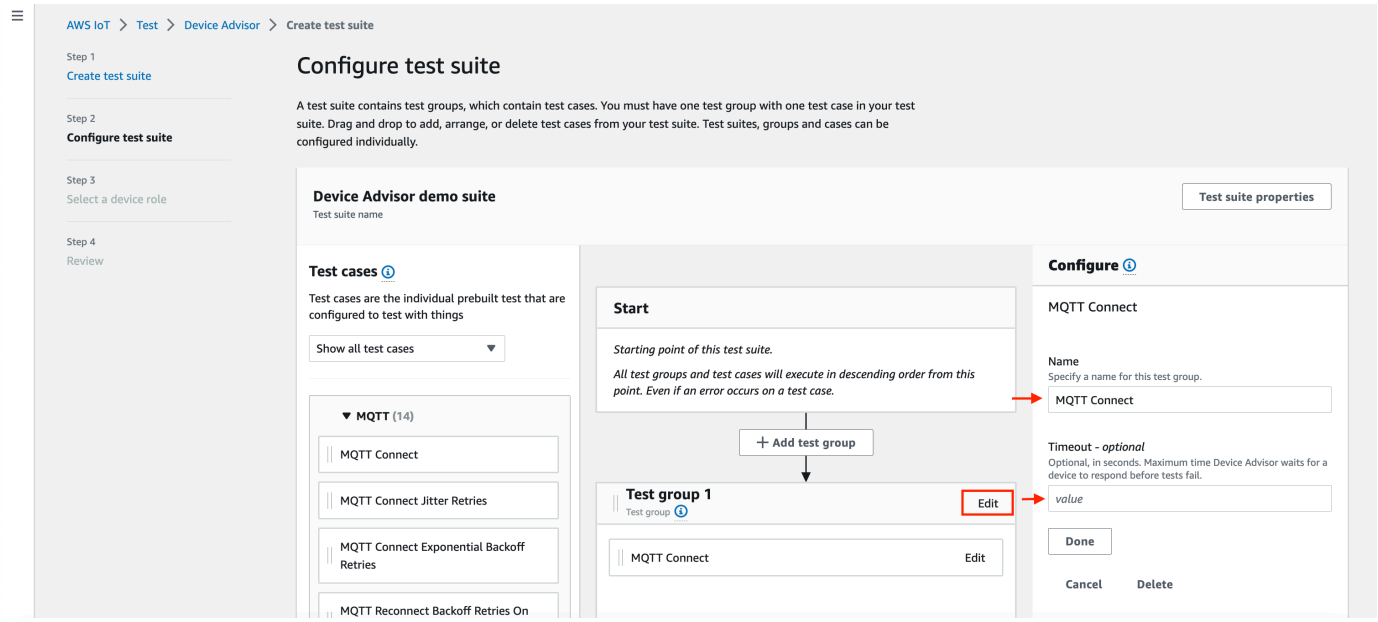
選擇 Done (完成)。

5. 將其中一個可用的測試案例從 Test cases (測試案例) 拖曳至測試群組。



6. 若要為新增至測試群組的測試案例修改測試案例層級組態，請選擇 Edit (編輯)。接著，輸入 Name (名稱)，為群組提供自訂名稱。

   您也可以選擇性地在選取的測試群組下輸入 Timeout (逾時) 值 (以秒為單位)。如果您未指定逾時值，則系統會使用預設值。

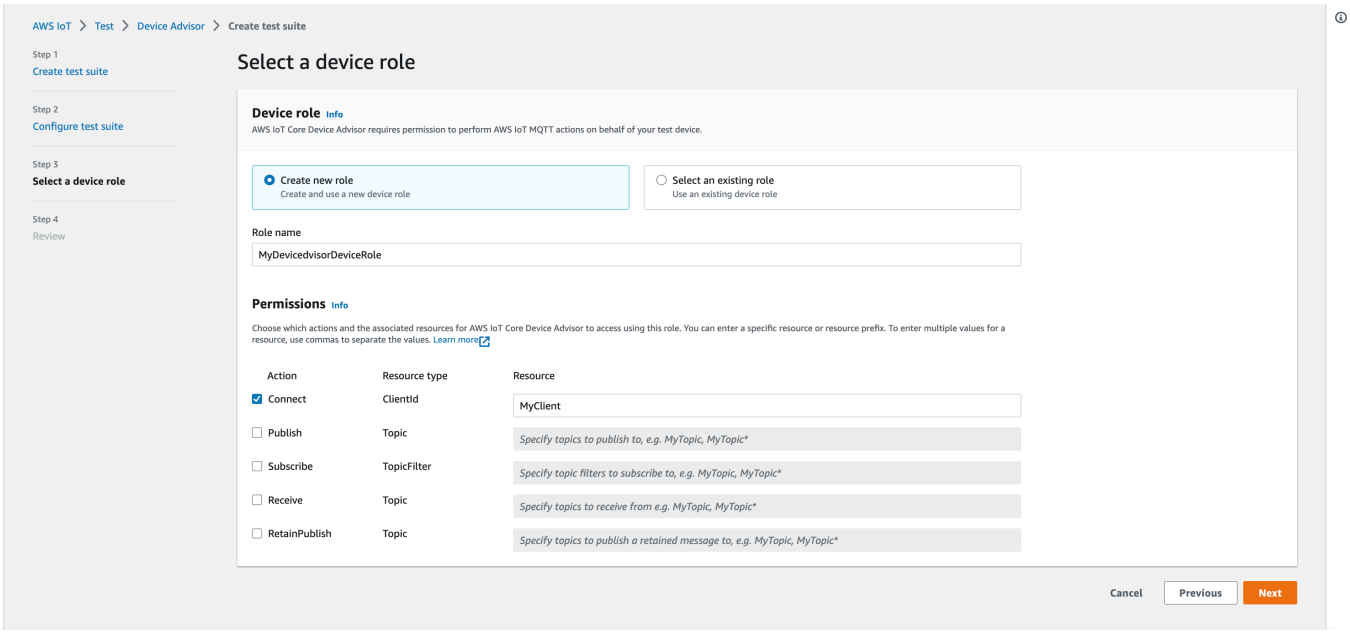


選擇 Done (完成)。

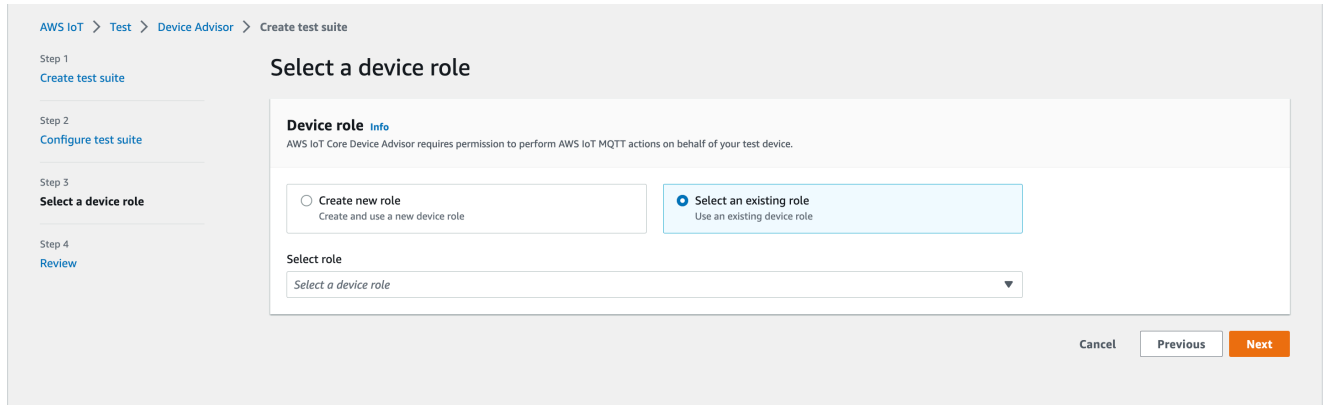
> **Note**
>
> 若要將更多測試群組新增至測試套件，請選擇 Add test group (新增測試群組)。依照上述步驟建立和設定更多測試群組，或將更多測試案例新增至一或多個測試群組。選擇測試案例並將其拖曳至所需位置，即可將測試群組和測試案例重新排序。Device Advisor 會依您定義測試群組和測試案例的順序執行測試。

7. 選擇下一步。
8. 在步驟 3 中，設定 Device Advisor 將用來代表測試裝置執行 AWS IoT MQTT 動作的裝置角色。

   如果僅在 Step 2 (步驟 2) 中選取 MQTT Connect 測試案例，系統會自動檢查 Connect (連接) 動作，因為裝置角色需要該許可才能執行此測試套件。如果選取了其他測試案例，系統會檢查相應的必要動作。確保已為每個動作提供資源值。例如，對於 Connect (連接) 動作，請提供裝置在連接至 Device Advisor 端點所用的用戶端 ID。可以使用逗號分隔值的方式提供多個值，也可以使用萬用字元 (*) 字元來提供字首值。若要提供以 `MyTopic` 為開頭的主題發佈許可，則可提供 `MyTopic*` 作為資源值。



如果先前已建立裝置角色，而且希望使用該角色，請選擇 Select an existing role (選取現有角色)，然後在 Select role (選取角色) 下選擇裝置角色。



使用提供的兩個選項之一來設定裝置角色，然後選擇 Next (下一步)。

9. 在 Step 4 (步驟 4)，請確保每個步驟中提供的組態正確無誤。若要編輯針對特定步驟提供的組態，請選擇 Edit (編輯) 來取得相應的步驟。

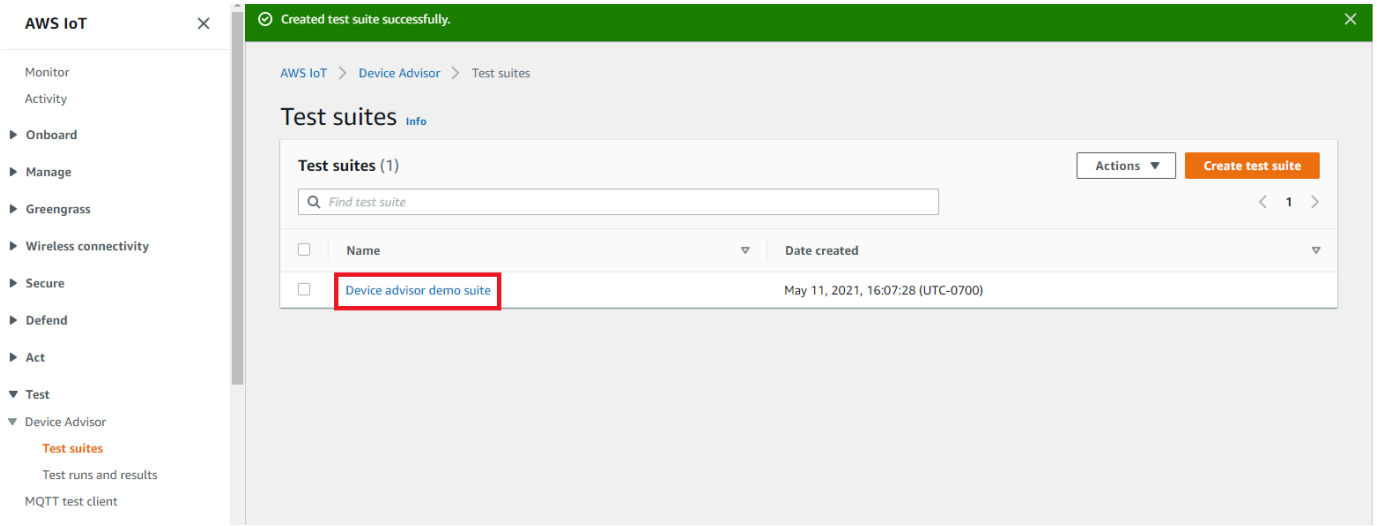
   確認組態之後，請選擇 Create test suite (建立測試套件)。

   應該成功建立測試套件，並且您將被重新導向到 Test suites (測試套件) 頁面，在這裡您可以檢視所有已建立的測試套件。

   如果建立測試套件失敗，請確保已根據先前的指示，設定測試套件、測試群組、測試案例和裝置角色。

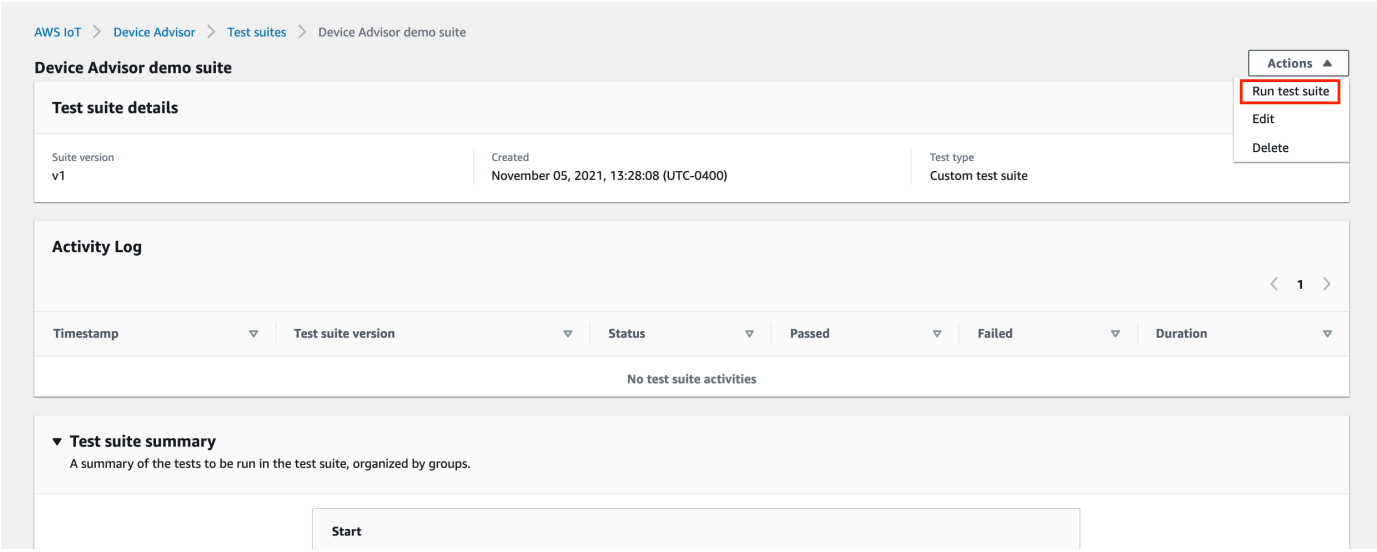
啟動測試套件執行

1. 在 AWS IoT 主控台的導覽窗格中，展開 Test (測試)、Device Advisor，然後選擇 Test suites (測試套件)。
2. 選擇您要檢視哪個測試套件的詳細資訊。



測試套件詳細資訊頁面會顯示與測試套件相關的所有資訊。

3. 選擇 Actions (動作)，然後 Run test suite (執行測試套件)。

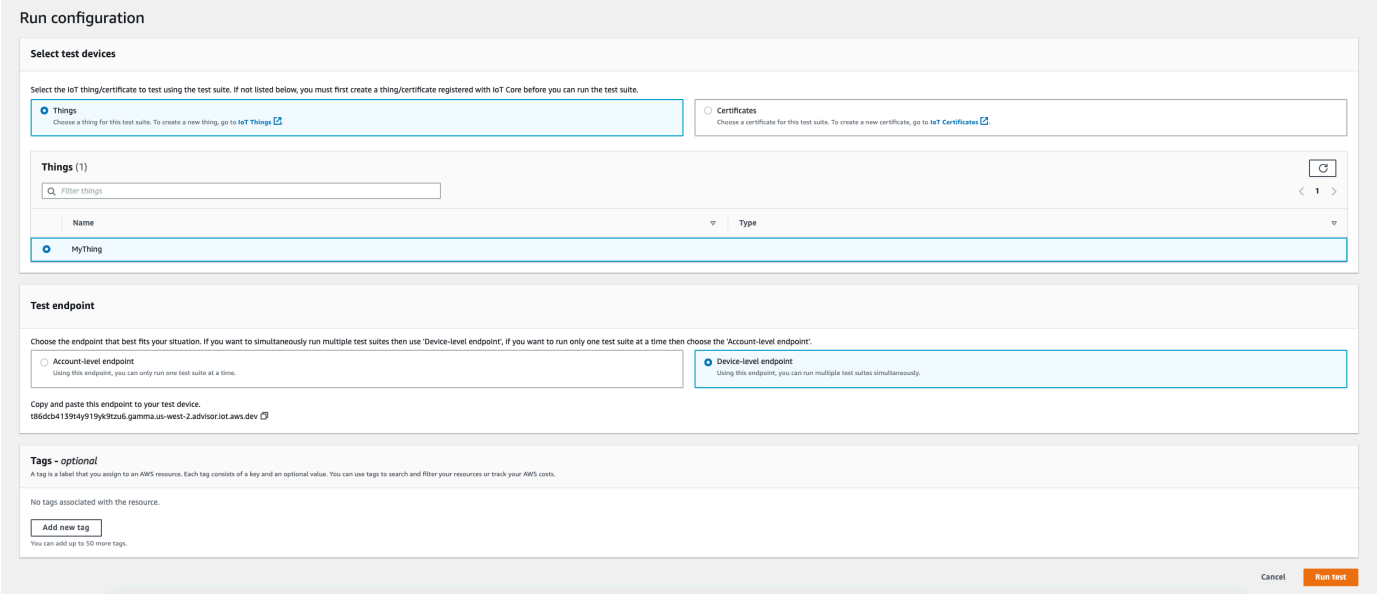


4. 在執行組態下，您將需要選取要使用 Device Advisor 測試的 AWS IoT 物件或憑證。如果您沒有任何現有的實物或憑證，請先建立 AWS IoT Core 資源。

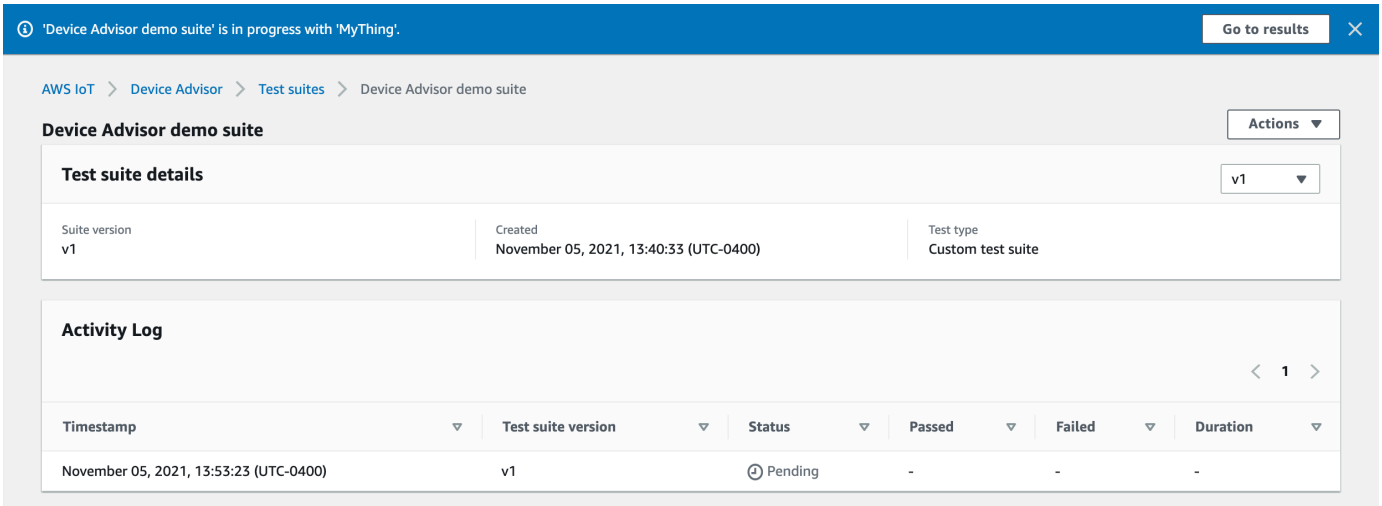
在 Test endpoint (測試端點) 區段中，選取最適合案例的端點。如果您計劃在未來使用相同的 AWS 帳戶同時執行多個測試套件，請選取裝置層級端點。如果打算一次只執行一個測試套件，請選取 Account-level endpoint (帳戶層級端點)。

使用選取的 Device Advisor 測試端點來設定測試裝置。

選取物件或憑證並選擇 Device Advisor 端點後，請選擇 Run test (執行測試)。

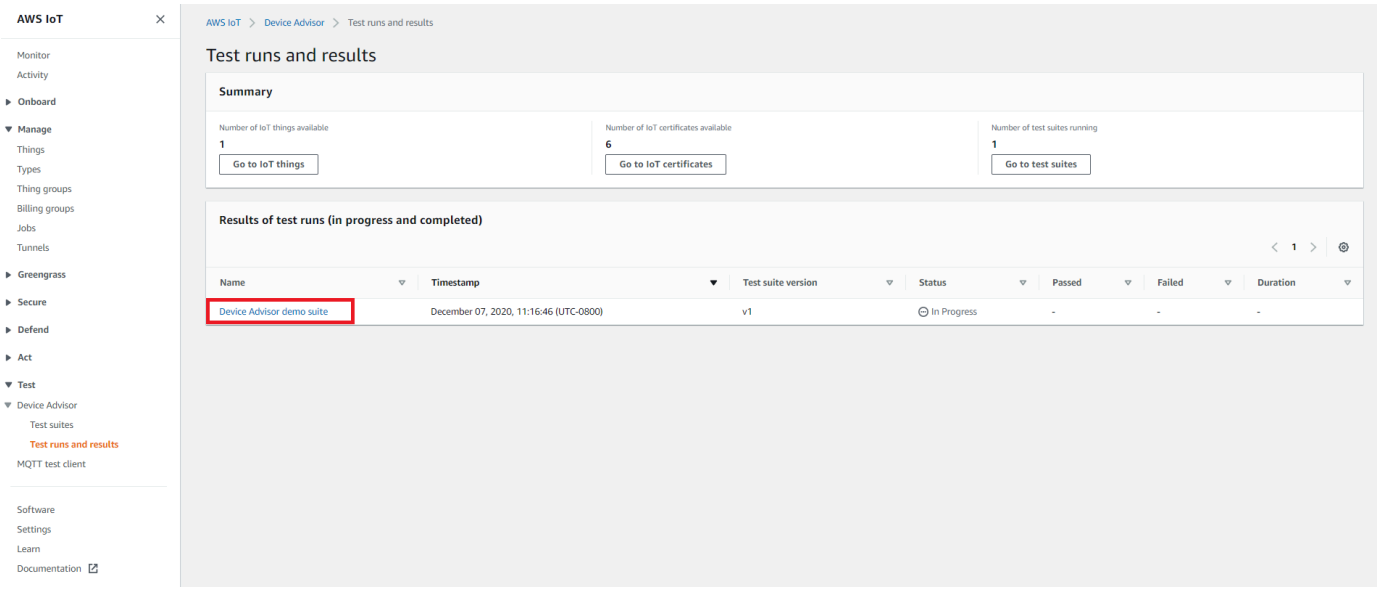


5. 選擇頂端橫幅上的 Go to results (前往結果)，以檢視測試執行詳細資訊。

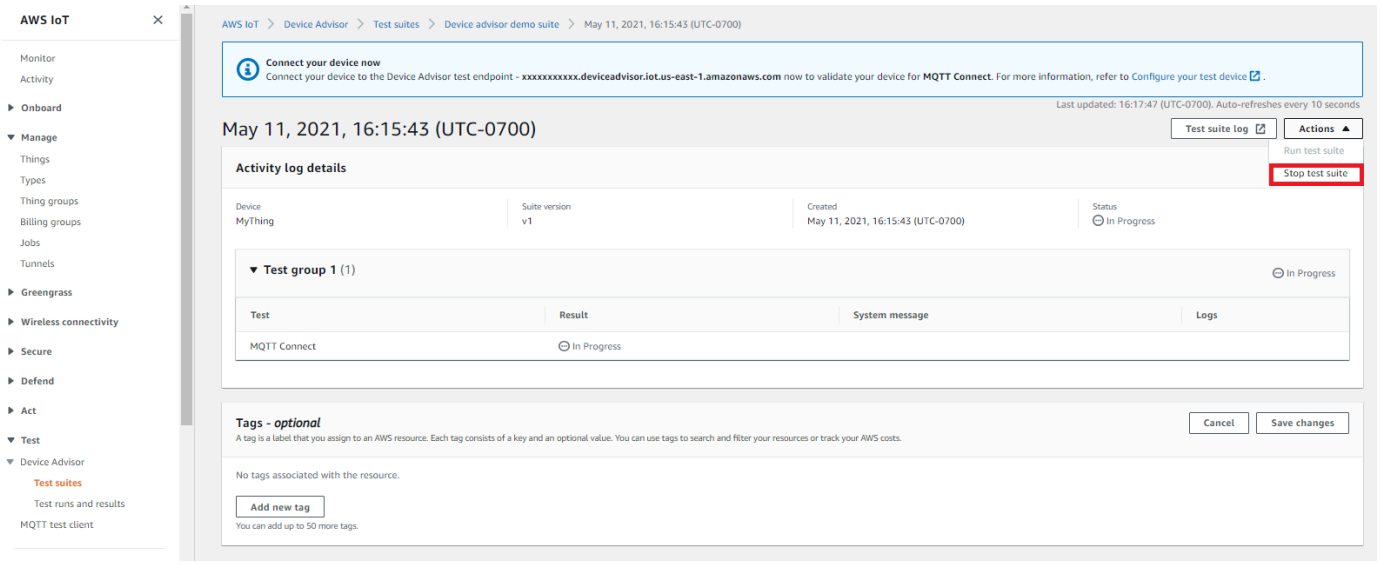


停止測試套件執行 (選用)

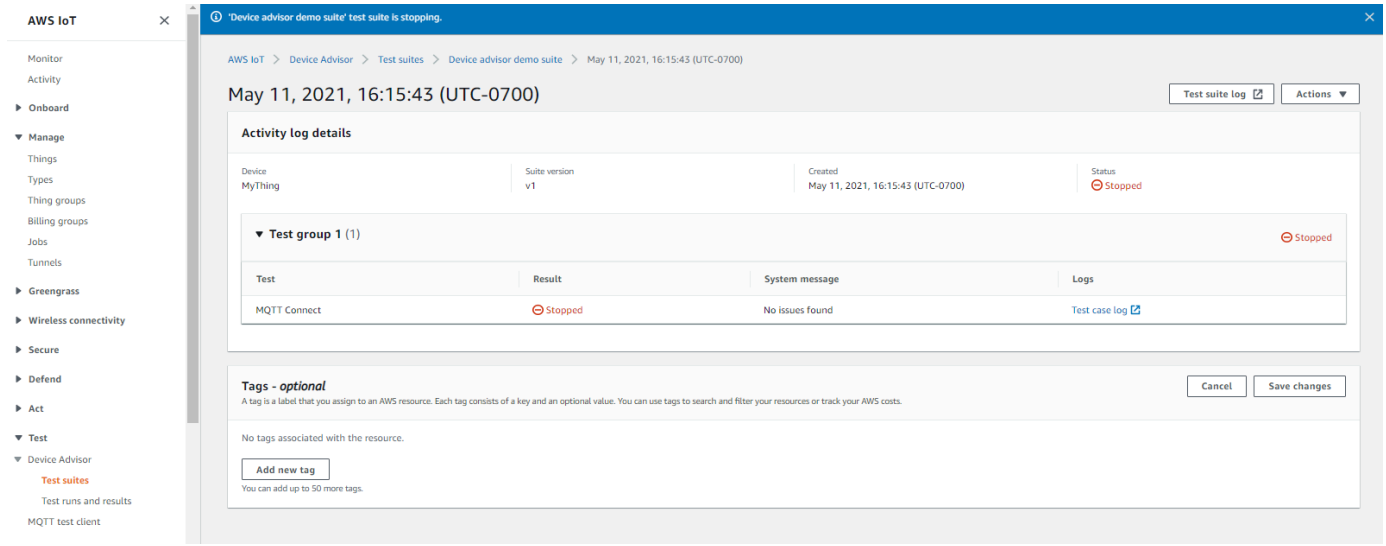
1. 在 AWS IoT 主控台的導覽窗格中，展開 Test (測試)、Device Advisor，然後選擇 Test runs and results (測試執行和結果)。
2. 選擇您要停止的進行中測試套件。



3. 選擇 Actions (動作)，然後 Stop test suite (停止測試套件)。



4. 此清除程序需要幾分鐘的時間來完成。當清除程序執行時，測試執行狀態將為 `STOPPING`。等待清除程序完成，且測試套件狀態變更為 `STOPPED` 狀態，再開始新的套件執行。



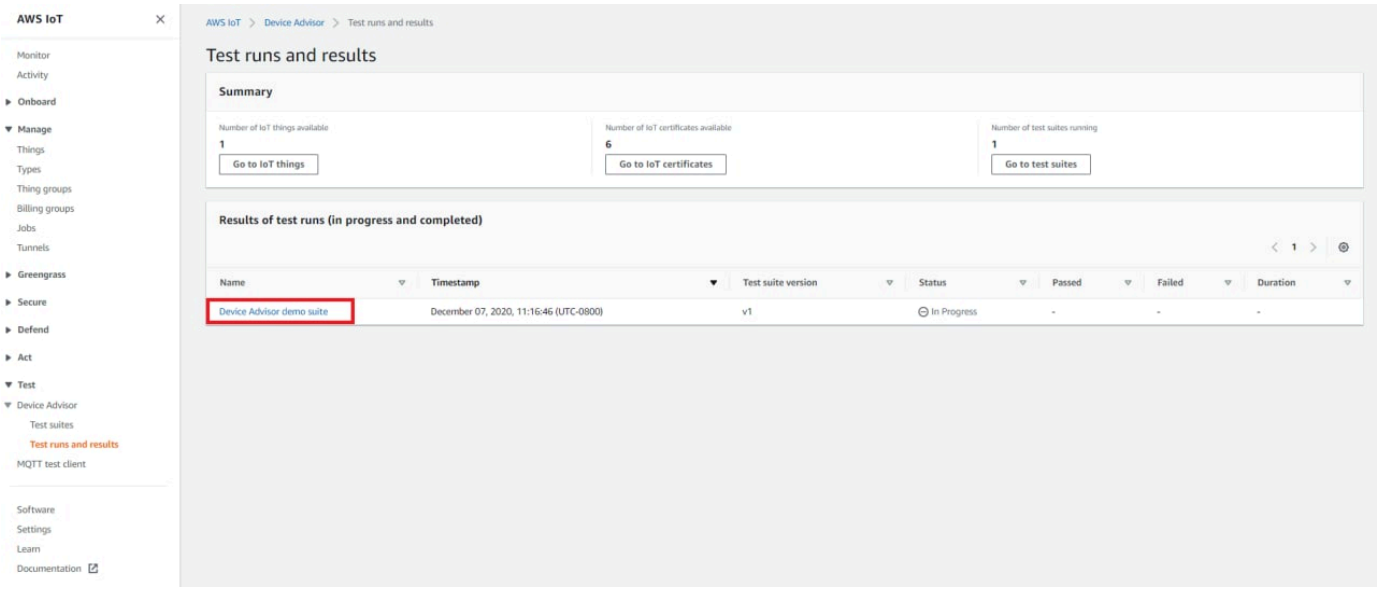
檢視測試套件執行詳細資訊和記錄

1. 在 AWS IoT 主控台的導覽窗格中，展開 Test (測試)、Device Advisor，然後選擇 Test runs and results (測試執行和結果)。

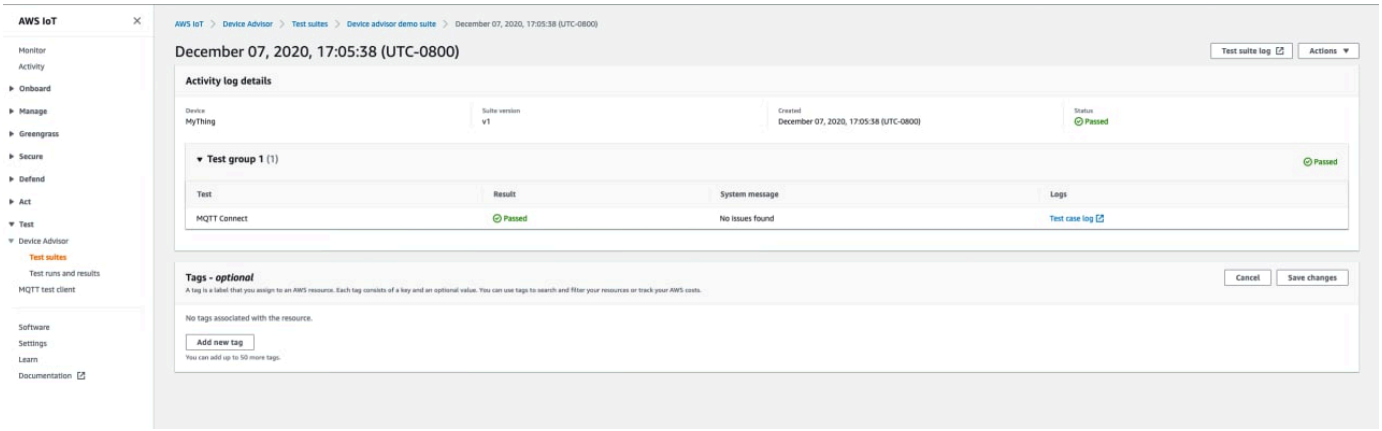
   此頁面顯示：

   - IoT 物件的數目
   - IoT 憑證的數目
   - 目前執行的測試套件數目
   - 所有已建立的測試套件執行

2. 選擇您要檢視哪個測試套件的執行詳細資訊和記錄。



執行摘要頁面會顯示目前測試套件執行的狀態。此頁面每隔 10 秒自動重新整理一次。我們建議您為裝置建置一個機制，每隔五秒嘗試連接到我們的測試端點，持續一到兩分鐘。然後，您可以採取自動方式依序執行多個測試用例。



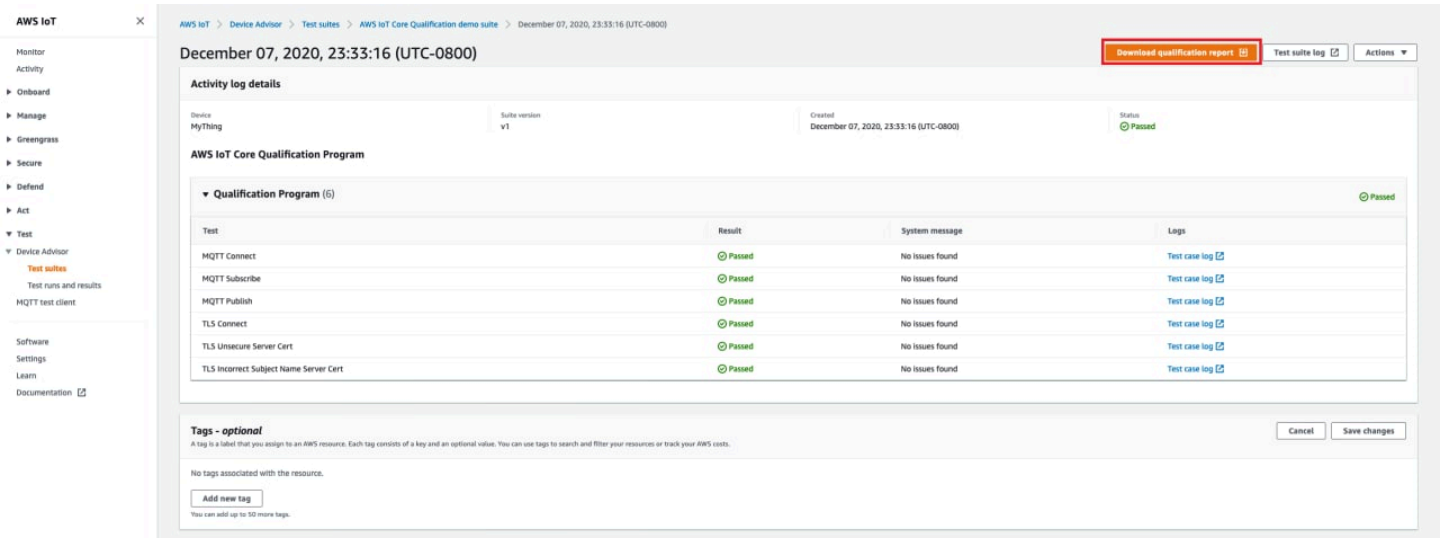
3. 若要存取測試套件執行的 CloudWatch 記錄，請選擇 Test suite log (測試套件記錄)。

   若要存取任何測試案例的 CloudWatch 記錄，請選擇 Test case log (測試案例記錄)。

4. 根據您的測試結果，疑難排解您的裝置，直到所有測試都通過。

下載 AWS IoT 資格報告

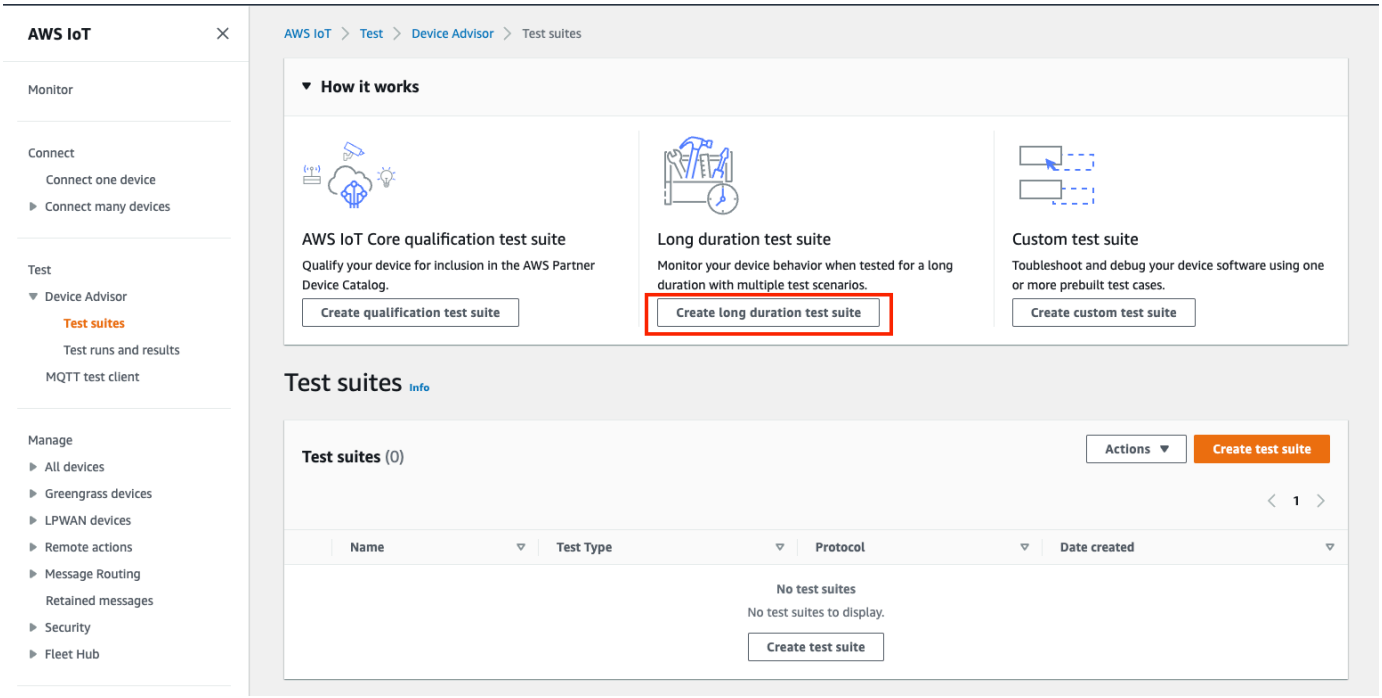
如果您在建立測試套件時選擇使用 AWS IoT 資格測試套件選項，並且能夠執行資格測試套件，則可以在測試執行摘要頁面中選擇下載資格報告來下載資格報告。



長時間測試主控台工作流程

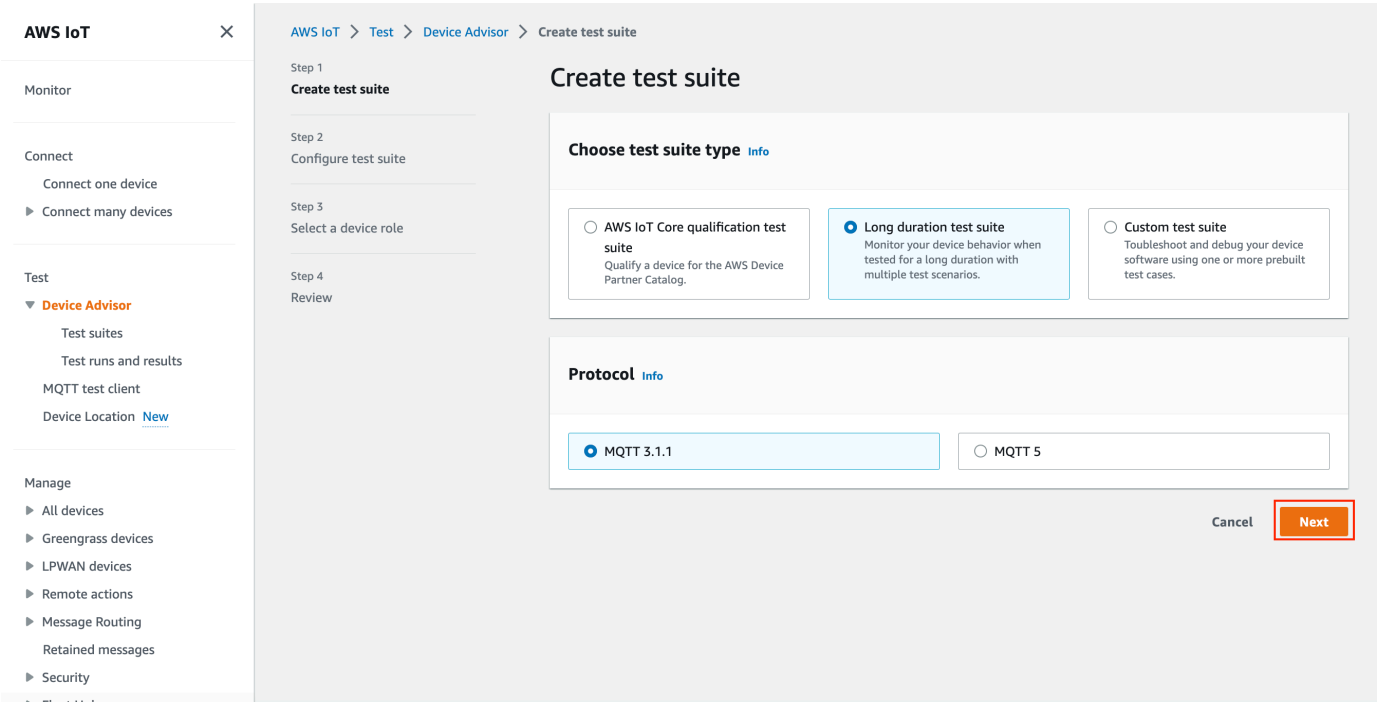
本教學課程可協助您使用主控台對 Device Advisor 進行長時間測試。若要完成教學課程，請遵循 設定 中的步驟。

1. 在 AWS IoT 主控台的導覽窗格中，展開 Test (測試)、Device Advisor，然後選擇 Test suites (測試套件)。在頁面上，選取 Create long duration test suite (建立長時間測試套件)。



2. 在 Create test suite (建立測試套件) 頁面，選取 Long duration test suite (長時間測試套件)，然後選擇Next (下一步)。

對於通訊協定，請選擇 MQTT 3.1.1 或 MQTT 5。



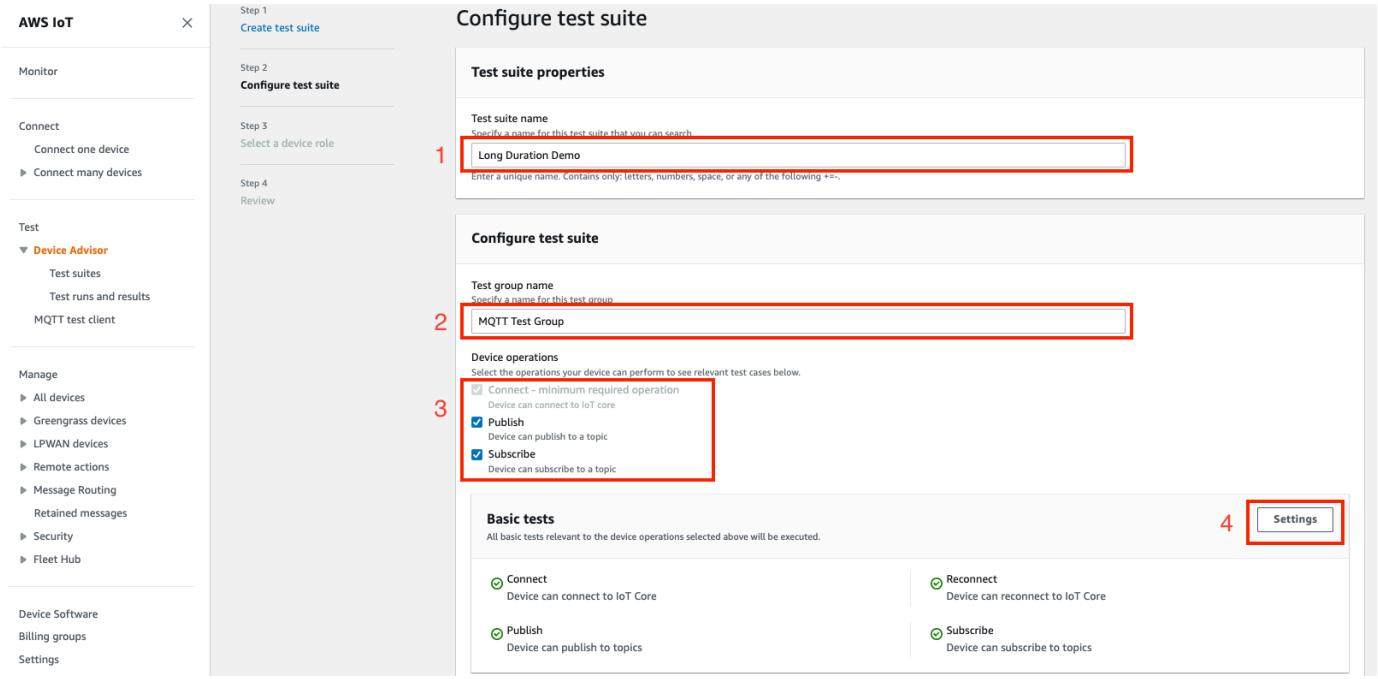
3. 在 Configure test suite (設定測試套件) 頁面上，執行以下操作：

   a. 更新 Test suite name (測試套件名稱) 欄位。

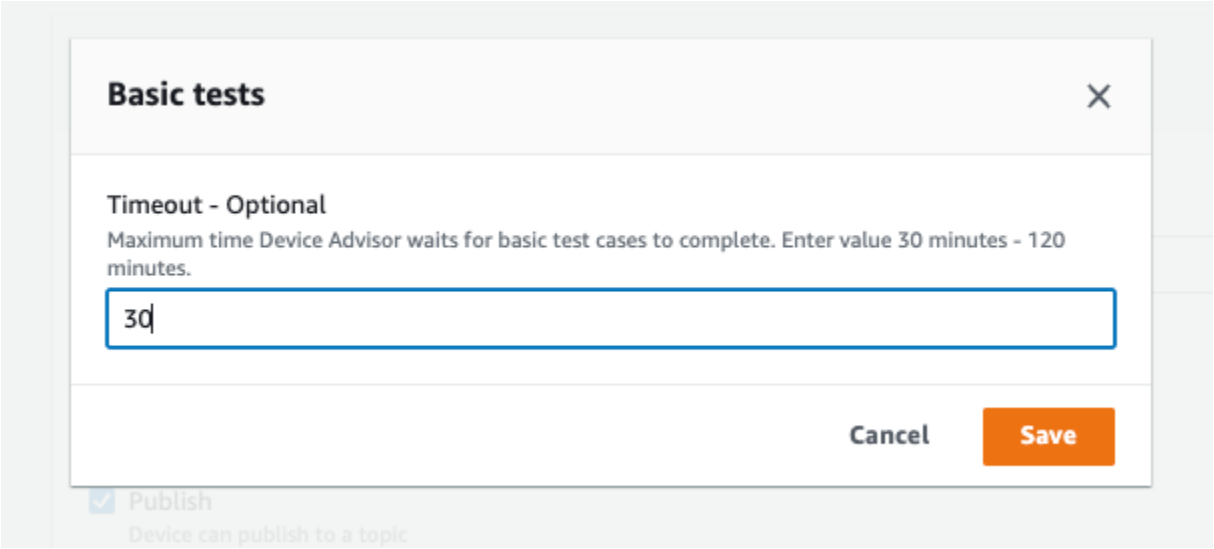
   b. 更新 Test group name (測試群組名稱) 欄位。

   c. 選擇裝置可以執行的 Device operations (裝置操作)。此步驟將會選擇要執行的測試。

   d. 選取 Settings (設定) 選項。



4. (選擇性) 輸入 Device Advisor 等待基本測試完成所需的時間上限。選取儲存。



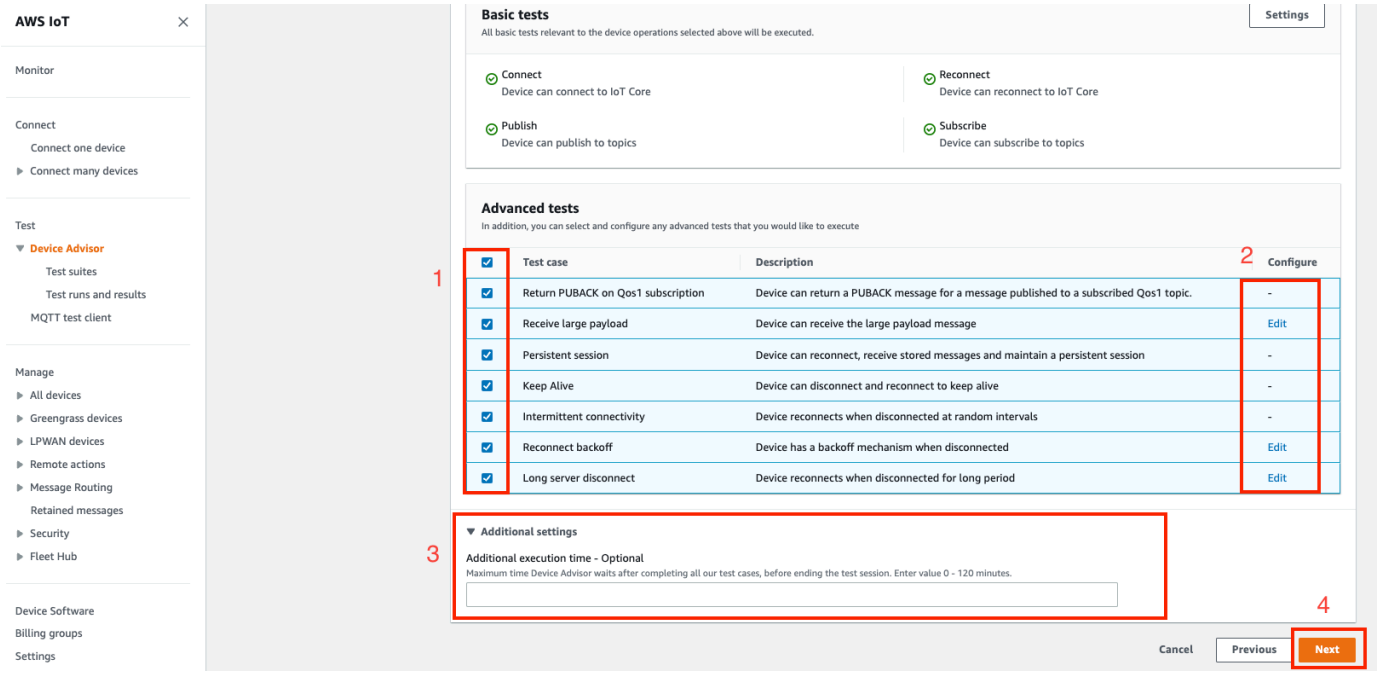
5. 在 Advanced tests (進階測試) 和 Additional settings (其他設定) 區段中執行下列動作。

   a. 選取或取消選取您要在此測試中執行的 Advanced tests (進階測試)。

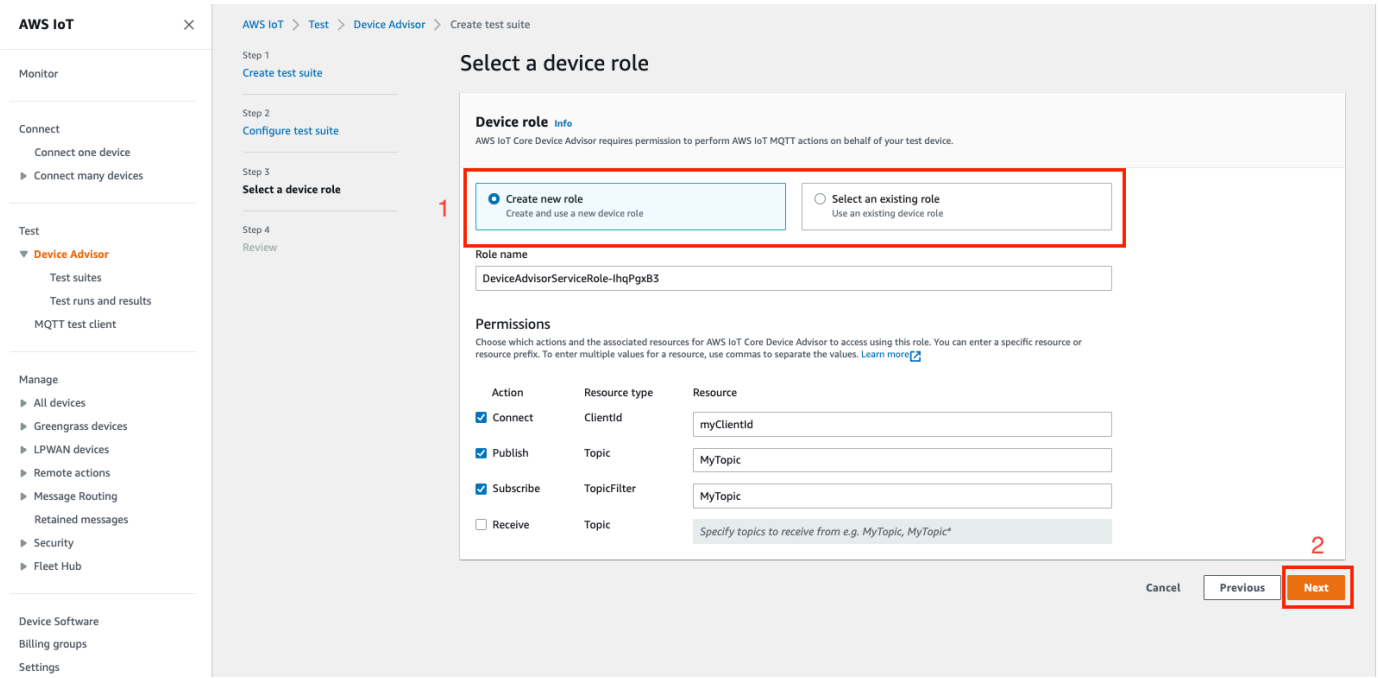
   b. Edit (編輯) 測試組態（如果適用）。

   c. 在 Additional settings (其他設定) 區段下設定 Additional execution time (其他執行時間)。

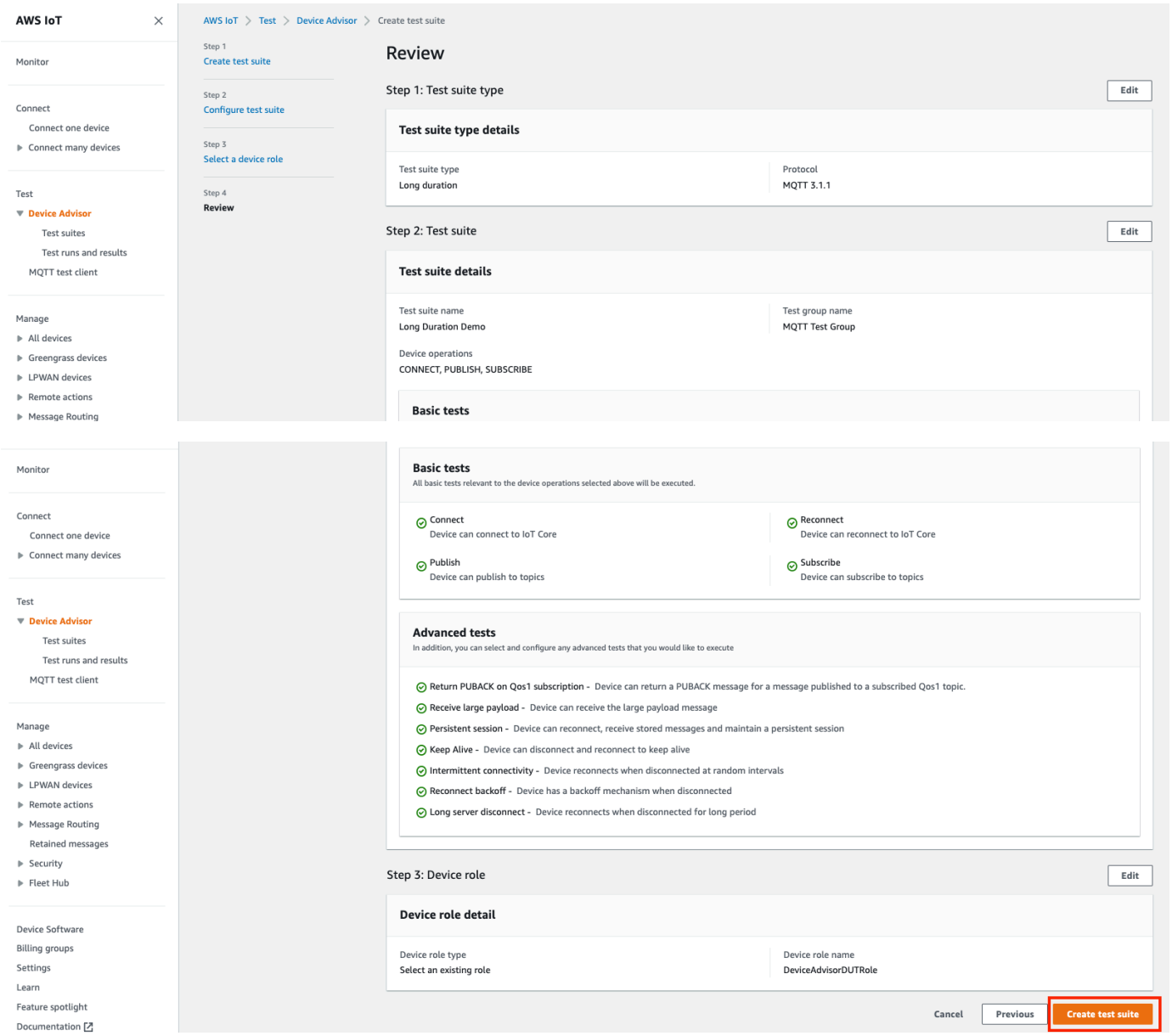
   d. 選擇 Next (下一步)，並進行下一個步驟。



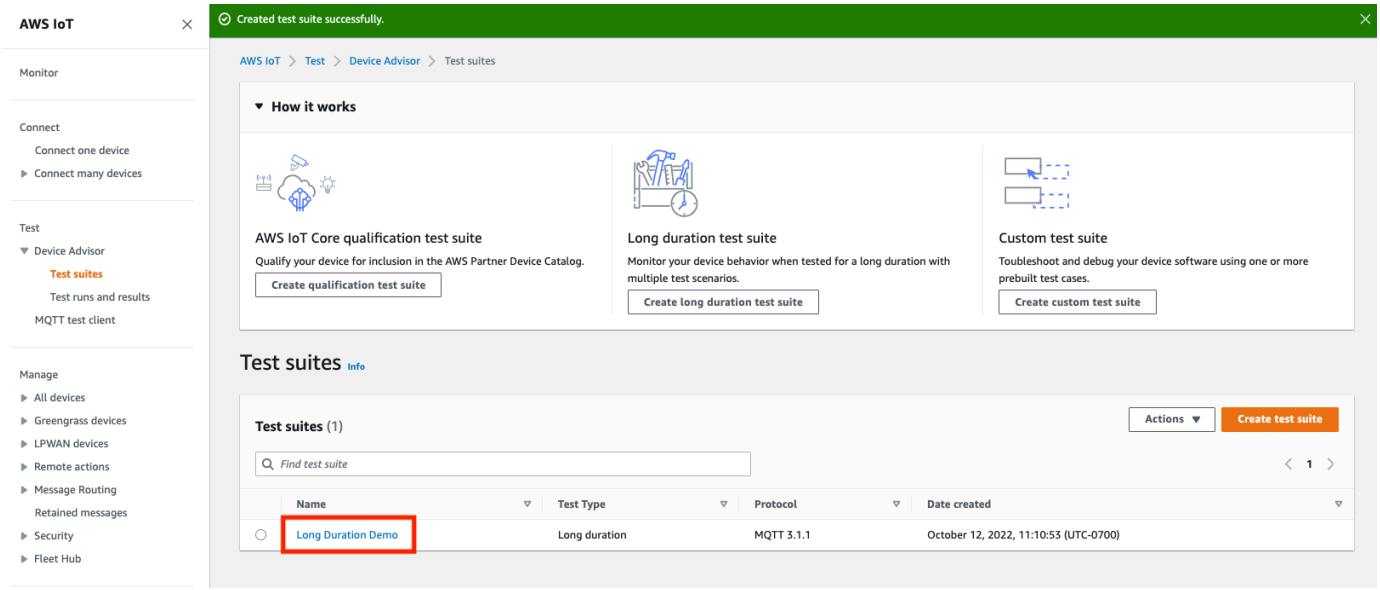
6. 在此步驟中選擇 Create a new role (建立新角色) 或 Select an existing role (選取現有角色)。如需詳細資訊，請參閱 [建立要用作裝置角色的 IAM 角色](#)。



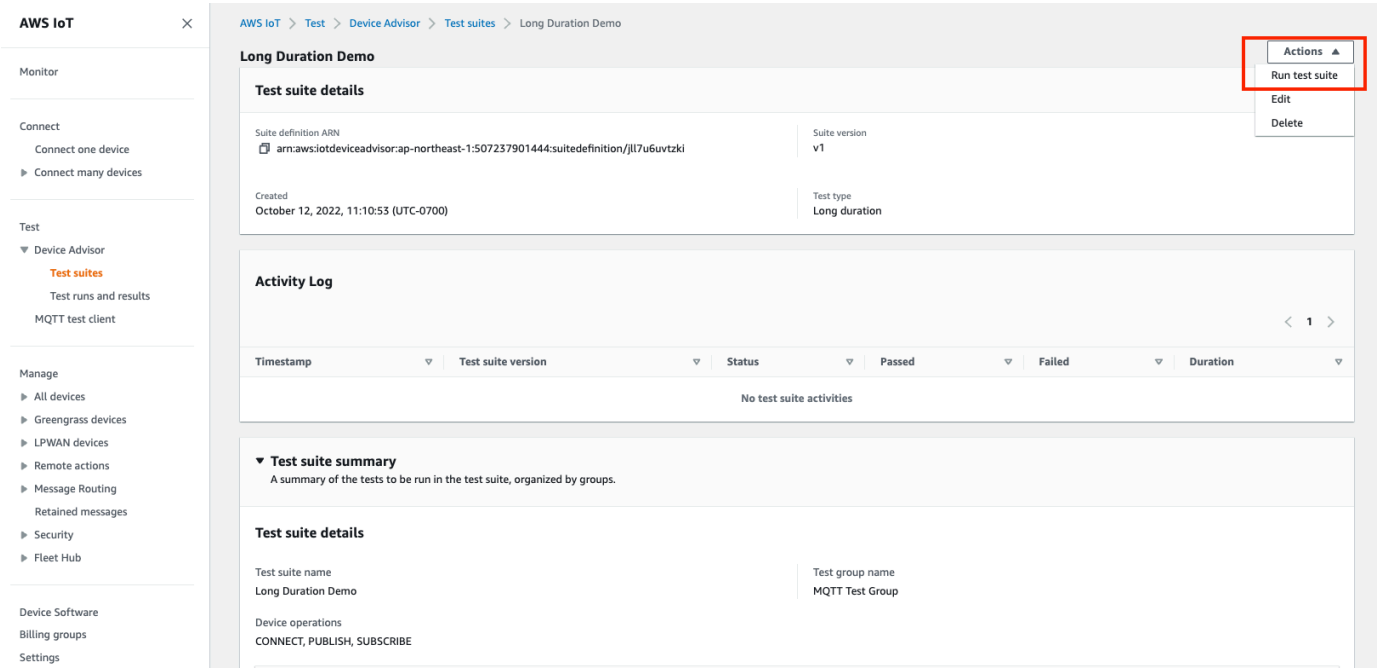
7. 檢閱所有在此步驟之前建立的組態，然後選取 Create test suite (建立測試套件)。



8. 建立的測試套件位於 Test suites (測試套件) 區段下。選取套件以檢視詳細資訊。



9. 要執行建立的測試套件，請選取 Actions (動作)，然後選擇 Run test suite (執行測試套件)。

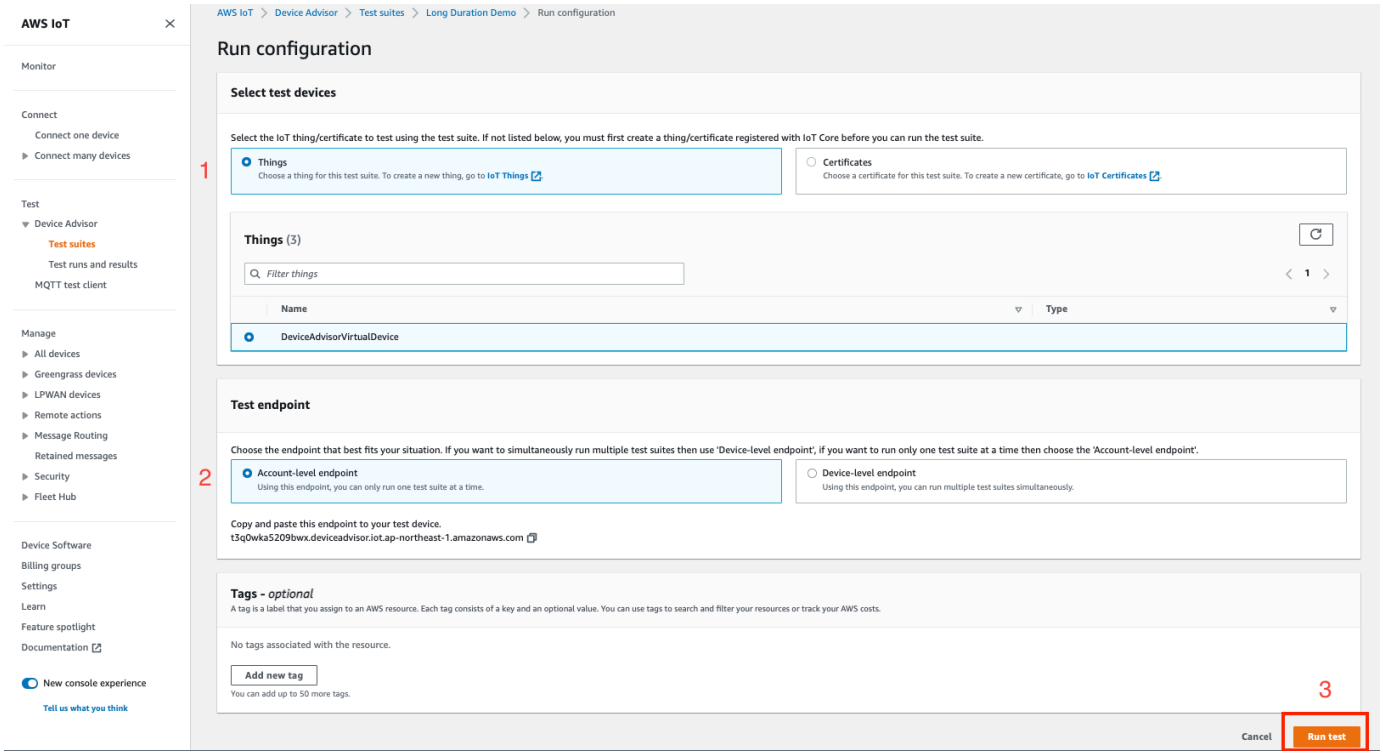


10. 在 Run configuration (執行組態) 頁面選擇組態選項。

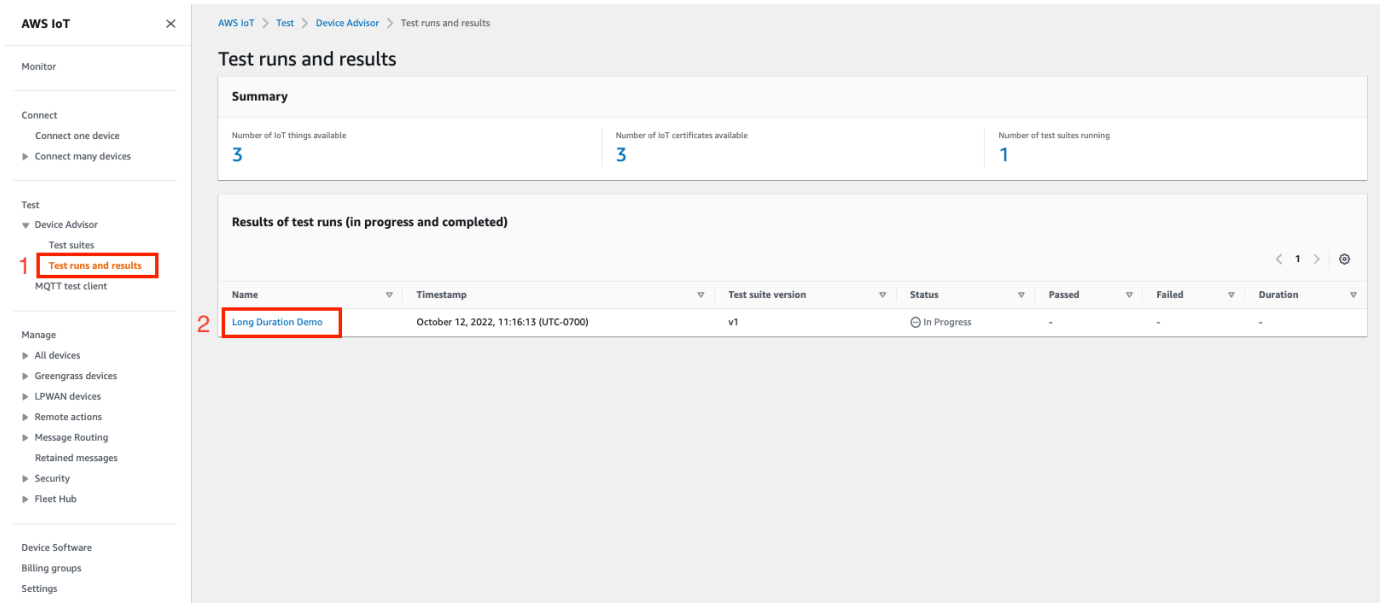
    a. 選取要執行測試的 Things (物件) 或 Certificate (憑證)。

    b. 選取 Account-level endpoint (帳戶層級端點) 或 Device-level endpoint (裝置層級端點)。

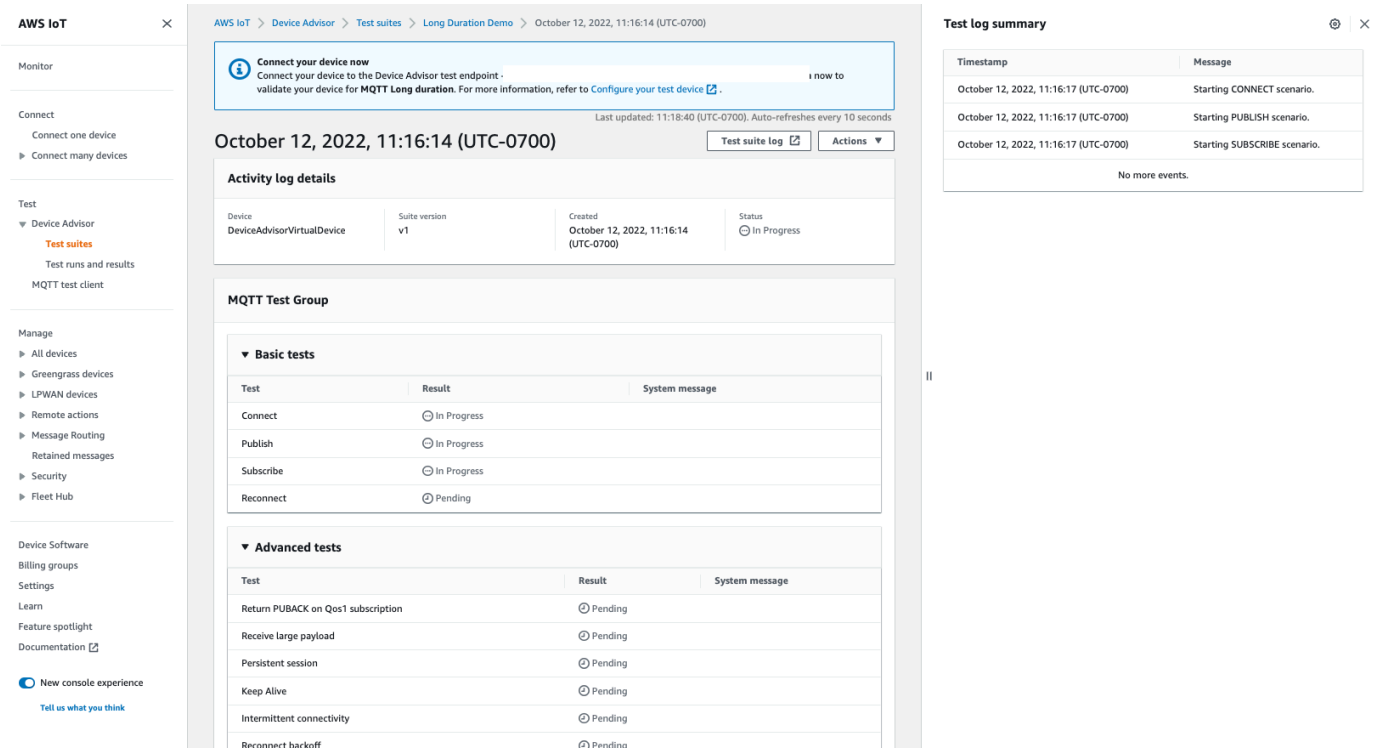
    c. 選擇 Test (測試) 以執行測試。



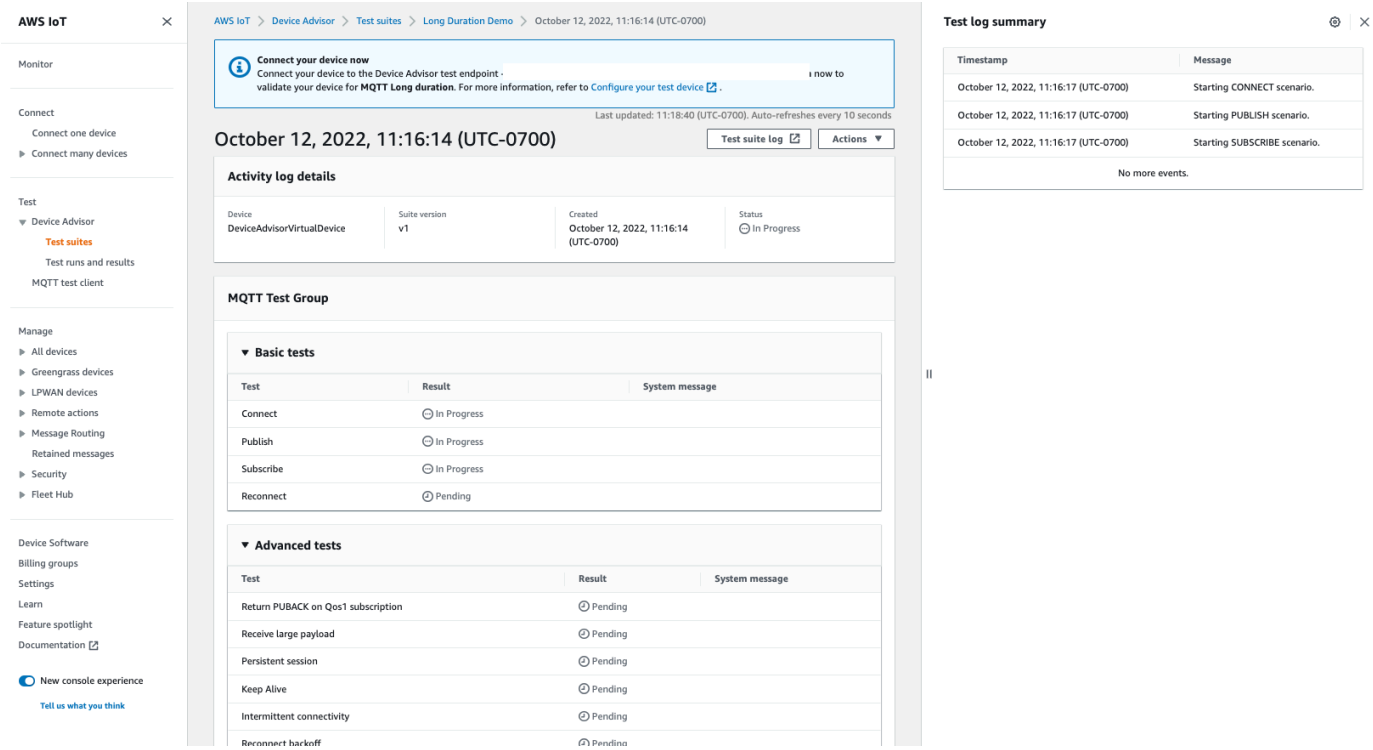
11. 要查看測試套件執行的結果，請選擇左側導航窗格中的 Test runs and results (測試執行和結果)。選擇已執行的測試套件，以檢視結果的詳細資訊。



12. 上一個步驟會顯示測試摘要頁面。所有關於測試執行的詳細資訊都會顯示在此頁面中。當主控台提示您開始進行裝置連線時，請將您的裝置連線至提供的端點。測試的進度會顯示在此頁面。



13. 長時間測試會在側邊面板列出額外的 Test log summary (測試日誌摘要)，以近乎即時的速度顯示所有發生於裝置與代理程式之間的重要事件。要查看更深入的詳細日誌，請按一下 Test case log (測試案例日誌)。



Device Advisor VPC 端點 (AWS PrivateLink)

您可以建立介面 VPC 端點，在 VPC 與 AWS IoT Core Device Advisor 測試端點 （資料平面） 之間建立私有連線。您可以在將 AWS IoT 裝置部署到生產環境 AWS IoT Core 之前，使用此端點驗證裝置與 的可靠且安全連線。Device Advisor 的預先建置測試可協助依據最佳實務驗證您的裝置軟體對於 TLS、MQTT、Device Shadow 和 AWS IoT 工作 的使用。

AWS PrivateLink 支援與 IoT 裝置搭配使用的介面端點。此服務可協助您在不使用網際網路閘道、NAT 裝置、VPC 連線或 Direct Connect 連接的情況下，以私有方式存取 AWS IoT Core Device Advisor 測試端點。您 VPC 中傳送 TCP 和 MQTT 封包的執行個體不需要公有 IP 地址，即可與 AWS IoT Core Device Advisor 測試端點通訊。VPC 與 AWS IoT Core Device Advisor 之間的流量不會離開 AWS 雲端。IoT 裝置與 Device Advisor 測試案例之間的任何 TLS 和 MQTT 通訊都會保留在您 AWS 帳戶的資源中。

每個介面端點都由子網路中的一個或多個彈性網路介面表示。

如需 VPC 端點的詳細資訊，請參閱《Amazon VPC 使用者指南》中的界面 VPC 端點 (AWS PrivateLink)。

AWS IoT Core Device Advisor VPC 端點的考量事項

設定界面 VPC 端點之前，請務必檢閱《Amazon VPC 使用者指南》中的界面端點屬性和限制。繼續之前，請考慮以下項目：

- AWS IoT Core Device Advisor 目前支援從您的 VPC 呼叫 Device Advisor 測試端點 （資料平面）。訊息代理程式使用資料平面通訊來傳送和接收資料。它藉由 TLS 和 MQTT 封包的協助做到這一點。將 AWS IoT 裝置 AWS IoT Core Device Advisor 連線至 Device Advisor 測試端點的 VPC 端點。此 VPC 端點不會使用控制平面 API 動作。若要建立或執行測試套件或其他控制平面 APIs，請透過公有網際網路使用 主控台、 AWS 開發套件或 AWS 命令列界面。
- 下列 AWS 區域 支援 的 VPC 端點 AWS IoT Core Device Advisor：
  - 美國東部 (維吉尼亞北部)
  - 美國西部 (奧勒岡)
  - 亞太地區 (東京)
  - 歐洲 (愛爾蘭)
- Device Advisor 支援具有 X.509 用戶端憑證和 RSA 伺服器的 MQTT。
- 目前不支援 VPC 端點政策。

- 如需如何建立資源 (連接 VPC 端點) 的指示，請檢查 VPC 端點的必要條件。您必須建立 VPC 和私有子網路，才能使用 AWS IoT Core Device Advisor VPC 端點。
- 資源上有配額 AWS PrivateLink 。如需詳細資訊，請參閱 AWS PrivateLink 配額。
- VPC 端點僅支援 IPv4 流量。

建立 的介面 VPC 端點 AWS IoT Core Device Advisor

若要開始使用 VPC 端點，請建立介面 VPC 端點。接著，選取 AWS IoT Core Device Advisor 做為 AWS 服務。如果您使用的是 AWS CLI，請呼叫 describe-vpc-endpoint-services 以確認 AWS IoT Core Device Advisor 存在於您 中的可用區域中 AWS 區域。確認連接到端點的安全群組允許 MQTT 和 TLS 流量進行 TCP 協定通訊。例如，在美國東部 (維吉尼亞北部) 區域，使用下列命令：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.deviceadvisor.iot
```

您可以使用 AWS IoT Core 下列服務名稱為 建立 VPC 端點：

- com.amazonaws.region.deviceadvisor.iot

根據預設，端點的私有 DNS 為開啟。這可確保使用預設測試端點會保留在您的私有子網路中。若要取得您的帳戶或裝置層級端點，請使用 主控台 AWS CLI 或 AWS SDK。例如，如果您在公有子網路或公用網際網路上執行 get-point，您可以取得端點並使用它來連線至 Device Advisor。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的透過介面端點存取服務。

若要將 MQTT 用戶端連線至 VPC 端點介面， AWS PrivateLink 服務會在連接到 VPC 的私有託管區域中建立 DNS 記錄。這些 DNS 記錄會將 AWS IoT 裝置的要求導向至 VPC 端點。

透過 VPC AWS IoT Core Device Advisor 端點控制對 的存取

您可以使用 VPC 條件內容金鑰來限制裝置對 的存取， AWS IoT Core Device Advisor 並僅允許透過 VPC 端點存取。 AWS IoT Core 支援下列 VPC 相關內容金鑰：

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core Device Advisor 目前不支援 VPC 端點政策。

下列政策授予許可，以 AWS IoT Core Device Advisor 使用符合物件名稱的用戶端 ID 連線至 。它也會發佈至以物件名稱為前置詞的任何主題。該政策取決於連接到具有特定 VPC 端點 ID 的 VPC 端點的裝置。此政策會拒絕與公有 AWS IoT Core Device Advisor 測試端點的連線嘗試。

```
{
"Version":"2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
"StringEquals": {
"aws:SourceVpce": "vpce-1a2b3c4d"
            }
        }

        },
        {
"Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

Device Advisor 測試案例

Device Advisor 提供六種類別的預先建置測試。

- TLS
- MQTT
- 影子
- 任務執行
- 許可和政策
- 長時間測試

Device Advisor 測試案例以符合 AWS Device Qualification Program 的資格。

根據 AWS 裝置資格計畫，您的裝置必須通過下列測試才能符合資格。

> **Note**
>
> 這是資格測試修訂清單。

- TLS Connect (TLS 連接) (「TLS 連接」)
- TLS Incorrect Subject Name Server Cert (TLS 不正確的主體名稱伺服器憑證) (「不正確的主體通用名稱 (CN) / 主體別名 (SAN)」)
- TLS Unsecure Server Cert (TLS 不安全的伺服器憑證) (「未由認可的 CA 簽署」)
- TLS 裝置支援 AWS IoT 密碼套件 (「TLS 裝置支援 AWS IoT 建議的密碼套件」)
- TLS 接收最大大小的片段 (「TLS 接收最大大小的片段」)
- TLS 過期伺服器憑證 (「過期伺服器憑證」)
- TLS 大型伺服器憑證 (「TLS 大型伺服器憑證」)
- MQTT Connect ("Device send CONNECT to AWS IoT Core (Happy case)")

- MQTT Subscribe (MQTT 訂閱) (「可訂閱 (Happy 案例)」)
- MQTT Publish (MQTT 發佈) (「QOS0 (Happy 案例)」)
- MQTT 連線抖動重試 (「裝置在抖動退避的情況下重試連接：沒有 CONNACK 回應」)

TLS

使用這些測試來判斷裝置與 之間的傳輸層安全通訊協定 (TLS) AWS IoT 是否安全。

> **Note**
>
> Device Advisor 現在支援 TLS 1.3。

Happy Path

TLS Connect

驗證測試中的裝置是否可以完成 TLS 交握 AWS IoT。此測試不會驗證用戶端裝置的 MQTT 實作。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。為了獲得最佳結果，我們建議使用 2 分鐘的逾時值。

```
"tests":[
   {
      "name":"my_tls_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
```

```
]
```

Example測試案例輸出：

- 通過 — 測試下的裝置已完成 TLS 交握 AWS IoT。
- 傳遞警告 — 測試下的裝置已完成 TLS 交握 AWS IoT，但有來自裝置或 的 TLS 警告訊息 AWS IoT。
- 失敗 — 由於交握錯誤 AWS IoT ，測試中的裝置無法完成與 的 TLS 交握。

TLS 接收最大大小的片段

此測試案例驗證您的裝置是否可以接收和處理 TLS 最大大小的片段。您的測試裝置必須訂閱 QoS 1 的預先設定主題，才能接收大型承載。您可以使用組態 `${payload}` 自訂承載。

Example API 測試案例定義：

> **Note**
>
> `EXECUTION_TIMEOUT` 的預設值為 5 分鐘。為了獲得最佳結果，我們建議使用 2 分鐘的逾時值。

```
"tests":[
   {
      "name":"TLS Receive Maximum Size Fragments",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
         "PAYLOAD_FORMAT":"{"message":"${payload}"}", // A string with a placeholder
 ${payload}, or leave it empty to receive a plain string.
         "TRIGGER_TOPIC": "test_1" // A topic to which a device will subscribe, and
 to which a test case will publish a large payload.
      },
      "test":{
         "id":"TLS_Receive_Maximum_Size_Fragments",
         "version":"0.0.0"
      }
   }
]
```

密碼套件

TLS 裝置支援 AWS IoT 建議的密碼套件

驗證來自待測裝置的 TLS 用戶端 Hello 訊息中的密碼套件是否包含建議的 AWS IoT 密碼套件。它提供裝置支援的密碼套件的更多洞見。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_tls_support_aws_iot_cipher_suites_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Support_AWS_IoT_Cipher_Suites",
         "version":"0.0.0"
      }
   }
]
```

Example測試案例輸出：

- 通過 — 測試密碼套件下的裝置至少包含其中一個建議的 AWS IoT 密碼套件，且不包含任何不支援的密碼套件。
- 通過但有警告 - 裝置密碼套件至少包含其中一個 AWS IoT 加密套件，但是：

  1. 它不包含任何建議的密碼套件
  2. 它包含 不支援的密碼套件 AWS IoT。

  我們建議您驗證任何不支援的密碼套件是否安全。
- 失敗 — 測試密碼套件下的裝置不包含任何 AWS IoT 支援的密碼套件。

較大型的伺服器憑證

TLS 大型伺服器憑證

接收和處理較大型伺服器憑證時，驗證您的裝置是否可以與 AWS IoT 完成 TLS 交握。此測試使用的伺服器憑證大小（以位元組為單位）大於目前在 TLS Connect 測試案例和 IoT Core 中使用的大小，截至 20。在此測試案例期間， 會 AWS IoT 測試裝置的 TLS 緩衝空間。如果緩衝空間足夠大，則 TLS 交握會完成，而不會發生錯誤。此測試不會驗證裝置的 MQTT 實作。TLS 交握程序完成後，測試案例即會停止。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。為了獲得最佳結果，我們建議使用 2 分鐘的逾時值。如果此測試案例失敗，但 TLS Connect 測試案例通過，我們建議您增加裝置的 TLS 緩衝區空間限制。增加緩衝區空間限制，確保您的裝置可以在大小增加的情況下處理更大的伺服器憑證。

```
"tests":[
   {
      "name":"my_tls_large_size_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Large_Size_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example測試案例輸出：

- 通過：待測裝置已完成與 AWS IoT的 TLS 交握。
- 傳遞警告 — 待測裝置已完成 TLS 交握 AWS IoT，但有來自裝置或 的 TLS 警告訊息 AWS IoT。
- 失敗 — 由於交握程序期間 AWS IoT 發生錯誤，測試中的裝置無法完成 TLS 交握。

TLS 不安全伺服器憑證

未由認可的 CA 簽署

驗證若提供的伺服器憑證沒有來自 ATS CA 的有效簽章，待測裝置是否會關閉連線。裝置應該只會連接到提供有效憑證的端點。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_tls_unsecure_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Unsecure_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example測試案例輸出：

- 通過：待測裝置已關閉連線。
- 失敗 — 正在測試的裝置已完成 TLS 交握 AWS IoT。

TLS 不正確的主體名稱伺服器憑證 / 不正確的主體通用名稱 (CN) / 主體別名 (SAN)

驗證若為網域名稱提供的伺服器憑證不同於所請求的伺服器憑證，測試下的裝置是否會關閉連線。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_tls_incorrect_subject_name_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"TLS_Incorrect_Subject_Name_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example測試案例輸出：

- 通過：待測裝置已關閉連線。
- 失敗 — 待測裝置已完成 TLS 交握 AWS IoT。

TLS 過期伺服器憑證

過期的伺服器憑證

驗證如果所提供伺服器憑證已過期，測試下的裝置是否會關閉連線。

Example API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_tls_expired_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Expired_Server_Cert",
```

```
         "version":"0.0.0"
      }
   }
]
```

Example測試案例輸出：

- 通過 — 待測裝置拒絕完成 TLS 交握 AWS IoT。裝置會在關閉連線前傳送 TLS 警示訊息。
- 通過但有警告 - 待測裝置拒絕完成與 AWS IoT的 TLS 交握。不過，它不會在關閉連線之前傳送 TLS 警示訊息。
- 失敗 — 測試中的裝置完成 TLS 交握 AWS IoT。

MQTT

CONNECT、DISCONNECT 和 RECONNECT

「裝置將 CONNECT 傳送至 AWS IoT Core (Happy case)」

驗證測試下的裝置是否傳送 CONNECT 請求。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

「裝置可將 PUBACK 傳回到 QOS1 的任意主題」

此測試案例將檢查裝置（用戶端）是否可在訂閱 QoS1 主題後收到代理程式的發佈訊息時傳回 PUBACK 訊息。

此測試案例的承載內容和承載大小是可設定的。如果已設定承載大小，Device Advisor 會覆寫承載內容的值，並將預先定義的承載傳送至具有所需大小的裝置。承載大小是介於 0 到 128 之間的值，且不能超過 128 KB。 AWS IoT Core 會拒絕發佈和連接大於 128 KB 的請求，如 AWS IoT Core 訊息代理程式和通訊協定限制和配額頁面所示。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。PAYLOAD_SIZE 可以設定為介於 0 到 128 KB 之間的值。定義承載大小會覆寫承載內容，因為 Device Advisor 會將具有指定大小的預先定義承載傳送回裝置。

```
"tests":[
{
        "name":"my_mqtt_client_puback_qos1",
        "configuration": {
            // optional:"TRIGGER_TOPIC": "myTopic",
            "EXECUTION_TIMEOUT":"300", // in seconds
            "PAYLOAD_FOR_PUBLISH_VALIDATION":"custom payload",
            "PAYLOAD_SIZE":"100" // in kilobytes
        },
        "test": {
            "id": "MQTT_Client_Puback_QoS1",
            "version": "0.0.0"
        }
    }
]
```

「裝置在抖動退避的情況下重試連接：沒有 CONNACK 回應」

驗證測試下的裝置在與代理程式重新連線至少五次時，是否使用適當的抖動退避。代理程式記錄測試下裝置的 CONNECT 請求的時間戳記、執行封包驗證、暫停而不傳送 CONNACK 到測試下的裝置，以及等待測試下的裝置重新傳送請求。允許第六次連線嘗試通過，並允許 CONNACK 回流到測試下的裝置。

再次執行上述程序。此測試案例需要裝置總共連接至少 12 次。收集的時間戳記用來驗證測試下的裝置是否使用抖動退避。如果待測裝置具有嚴格的指數退避延遲，則此測試案例會通過但有警告。

建議對待測裝置實作指數退避和抖動機制，以通過此測試案例。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 4 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_jitter_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",    // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Jitter_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

「裝置在指數退避的情況下重試連接：沒有 CONNACK 回應」

驗證測試下的裝置在與代理程式重新連線至少五次時，是否使用適當的指數退避。代理程式記錄測試下裝置的 CONNECT 請求的時間戳記、執行封包驗證、暫停而不傳送 CONNACK 到用戶端裝置，以及等待測試下的裝置重新傳送請求。收集的時間戳記用來驗證待測裝置是否使用指數退避。

建議對待測裝置實作指數退避和抖動機制，以通過此測試案例。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 4 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_exponential_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"600",  // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Exponential_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

「裝置在抖動退避的情況下重新連線：在伺服器中斷連線後」

驗證待測裝置在與伺服器中斷連線後重新連接時，是否使用必要的抖動和退避。Device Advisor 將裝置與伺服器的連線中斷至少五次，並觀察裝置在 MQTT 重新連線時的行為。Device Advisor 記錄待測裝置的 CONNECT 請求的時間戳記、執行封包驗證、暫停而不將 CONNACK 傳送到用戶端裝置，以及等待待測裝置重新傳送請求。收集的時間戳記用來驗證在重新連接時，待測裝置是否使用抖動和退避。如果待測裝置具有嚴格的指數退避，或未實作適當的抖動退避機制，則此測試案例會通過但有警告。如果待測裝置已實作線性退避或常數退避機制，則測試會失敗。

若要通過這個測試案例，建議對待測裝置實作指數退避和抖動機制。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 4 分鐘。
> 可以透過指定 RECONNECTION_ATTEMPTS 來變更驗證退避的重新連線嘗試次數。號碼必須為介於 5 到 10 之間的數字。預設值為 5。

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_server_disconnect",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
```

```
            "RECONNECTION_ATTEMPTS": 5
         },
         "test":{
            "id":"MQTT_Reconnect_Backoff_Retries_On_Server_Disconnect",
            "version":"0.0.0"
         }
      }
]
```

「裝置在抖動退避的情況下重新連線 - 連線不穩定」

驗證待測裝置在連線不穩定時，是否使用必要的抖動和退避。Device Advisor 會在 5 次成功連線後中斷裝置與伺服器的連線，並觀察裝置在 MQTT 重新連線時的行為。Device Advisor 會記錄待測裝置的 CONNECT 請求的時間戳記、執行封包驗證、傳送回 CONNACK、中斷連線、記錄中斷連線的時間戳記，以及等待待測裝置重新傳送請求。收集的時間戳記用來驗證在成功但不穩定的連線後重新連線時，待測裝置是否使用抖動和退避。如果待測裝置具有嚴格的指數退避，或未實作適當的抖動退避機制，則此測試案例會通過但有警告。如果待測裝置已實作線性退避或常數退避機制，則測試會失敗。

若要通過這個測試案例，建議對待測裝置實作指數退避和抖動機制。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 4 分鐘。
> 可以透過指定 RECONNECTION_ATTEMPTS 來變更驗證退避的重新連線嘗試次數。號碼必須為介於 5 到 10 之間的數字。預設值為 5。

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_unstable_connection",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Unstable_Connection",
         "version":"0.0.0"
```

```
      }
   }
]
```

發布

「QOS0 (Happy 案例)」

驗證測試中的裝置是否使用 QoS0 或 QoS1 發佈訊息。您也可以在測試設定中設定主題值和承載，來驗證訊息和承載的主題。

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_publish_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish",
         "version":"0.0.0"
      }
   }
]
```

「QOS1 發佈重試：沒有 PUBACK」

驗證若代理程式未傳送 PUBACK，測試下的裝置是否重新發佈與 QOS1 一起傳送的訊息。您也可以在測試設定中指定此主題，來驗證訊息的主題。用戶端裝置在重新發佈訊息之前不得中斷連線。此測試也會驗證重新發佈的訊息是否具有與原始訊息相同的封包識別符。測試執行期間，如果裝置失去連線並重新連接，則測試案例將重置而不會故障，並且裝置必須再次執行測試案例步驟。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。建議至少 4 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_publish_retry_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retry_No_Puback",
         "version":"0.0.0"
      }
   }
]
```

「發佈保留的訊息」

驗證受測裝置是否會發佈 retainFlag 設定為 true 的訊息。您可以在測試設定中設定主題值和承載，來驗證訊息和承載的主題。如果在 PUBLISH 封包中傳送的 retainFlag 未設定為 true，測試案例將會失敗。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。若要執行此測試案例，請在 device role (裝置角色) 中新增 iot:RetainPublish 動作。

```
"tests":[
   {
      "name":"my_mqtt_publish_retained_messages_test",
      "configuration":{
         // optional:
```

```
         "EXECUTION_TIMEOUT":"300",  // in seconds

 "TOPIC_FOR_PUBLISH_RETAINED_VALIDATION": "my_TOPIC_FOR_PUBLISH_RETAINED_VALIDATION",

 "PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retained_Messages",
         "version":"0.0.0"
      }
   }
]
```

「以使用者屬性進行發佈」

驗證受測裝置是否發佈含有正確使用者屬性的訊息。您可以在測試設定中設定名稱/值對來驗證使用者屬性。如果未提供使用者屬性或不相符，測試案例將會失敗。

API 測試案例定義：

> **Note**
>
> 這是僅限 MQTT5 的測試案例。
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_user_property_test",
      "test":{
        "USER_PROPERTIES": [
            {"name": "name1", "value":"value1"},
            {"name": "name2", "value":"value2"}
        ],
        "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"MQTT_Publish_User_Property",
         "version":"0.0.0"
      }
   }
]
```

訂閱

「可以訂閱 (Happy 案例)」

驗證測試下的裝置是否訂閱 MQTT 主題。您也可以在測試設定中指定此主題，來驗證測試下裝置所訂閱的主題。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_subscribe_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["my_TOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe",
         "version":"0.0.0"
      }
   }
]
```

「訂閱重試：沒有 SUBACK」

驗證測試下的裝置是否重試失敗的 MQTT 主題訂閱。伺服器接著會等待，而且不會傳送 SUBACK。如果用戶端裝置沒有重試訂閱，測試會失敗。用戶端裝置必須使用相同的封包 ID 重試失敗的訂閱。您也可以在測試設定中指定此主題，來驗證測試下裝置所訂閱的主題。測試執行期間，如果裝置失去連線並重新連接，則測試案例將重置而不會故障，並且裝置必須再次執行測試案例步驟。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 4 分鐘。

```
"tests":[
   {
      "name":"my_mqtt_subscribe_retry_test",
      "configuration":{
         "EXECUTION_TIMEOUT":"300",  // in seconds
         // optional:
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["myTOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe_Retry_No_Suback",
         "version":"0.0.0"
      }
   }
]
```

Keep-Alive

「Mqtt No Ack PingResp」

這個測試案例會驗證，如果待測裝置在未收到 Ping 回應時是否會中斷連線。在此測試案例中，Device Advisor 會封鎖從 傳送的回應， AWS IoT Core 以進行發佈、訂閱和 ping 請求。其也會驗證待測裝置是否中斷與 MQTT 的連線。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。建議使用的逾時值大於 keepAliveTime 值的 1.5 倍。
> 此測試的最大值keepAliveTime不得超過 230 秒。

```
"tests":[
```

```
    {
       "name":"Mqtt No Ack PingResp",
       "configuration":
          //optional:
          "EXECUTION_TIMEOUT":"306",   // in seconds
       },
       "test":{
          "id":"MQTT_No_Ack_PingResp",
          "version":"0.0.0"
       }
    }
]
```

持久性工作階段

「持久性階段作業 (Happy 案例)」

此測試案例會驗證裝置與持久性階段作業中斷連結時的行為。此測試案例會檢查裝置是否可以重新連接、恢復其觸發程序主題訂閱而不需明確重新訂閱、接收儲存於主題中的訊息，以及在持久性階段作業期間如預期運作。當此測試案例通過時，表示用戶端裝置能夠以預期的方式與 AWS IoT Core 代理程式維持持久性工作階段。如需 AWS IoT 持久性工作階段的詳細資訊，請參閱使用 MQTT 持久性工作階段。

在此測試案例中，用戶端裝置應在乾淨的階段作業旗標設為 false 下與 AWS IoT Core 連結，並且訂閱主題篩選條件。成功訂閱後，裝置將由 AWS IoT Core Device Advisor 中斷連線。當裝置處於中斷連結的狀態時，QoS 1 訊息承載將儲存在該主題中。Device Advisor 將允許用戶端裝置與測試端點重新連結。此時由於有持久性工作階段，用戶端裝置應恢復其主題訂閱，而不需傳送任何額外的訂閱封包，並從代理程式接收 QoS 1 訊息。重新連結後，如果用戶端裝置傳送額外的 SUBSCRIBE 封包，再次重新訂閱其觸發程序主題，和/或如果用戶端並未收到來自觸發程序主題的儲存訊息，測試案例將會失敗。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為至少 4 分鐘。第一次連線時，用戶端裝置需要明確訂閱之前沒有訂閱的 TRIGGER_TOPIC。若要通過測試案例，用戶端裝置必須使用 QoS 1 成功訂閱 TRIGGER_TOPIC。重新連結後，用戶端裝置應了解有

作用中的持久性工作階段；因此應接受觸發程序主題傳送的已儲存訊息，並針對該特定訊息傳回 PUBACK。

```
"tests":[
   {
      "name":"my_mqtt_persistent_session_happy_case",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         // if Payload not provided, a string will be stored in the trigger topic to
 be sent back to the client device
         "PAYLOAD": "The message which should be received from AWS IoT Broker after
 re-connecting to a persistent session from the specified trigger topic.",

         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Persistent_Session_Happy_Case",
         "version":"0.0.0"
      }
   }
]
```

「持久性工作階段到期 - 工作階段到期」

此測試案例有助於在已中斷連線的裝置重新連線到過期的持久性工作階段時，驗證裝置行為。工作階段過期後，我們希望裝置透過明確傳送新的 SUBSCRIBE 封包來重新訂閱之前訂閱的主題。

在第一次連線期間，我們預期測試裝置會與 AWS IoT 代理程式連線，因為其`CleanSession`旗標設為 false 以啟動持久性工作階段。然後，裝置應訂閱觸發程序主題。然後，在成功訂閱和啟動持久性工作階段後， AWS IoT Core Device Advisor 會中斷裝置連線。中斷連線後， AWS IoT Core Device Advisor 會允許測試裝置重新連線至測試端點。此時，當測試裝置傳送另一個 CONNECT 封包時， AWS IoT Core Device Advisor 會傳回 CONNACK 封包，指出持久性工作階段已過期。測試裝置需要正確解譯此封包，並且在持久性工作階段終止時，應再次重新訂閱相同的觸發程序主題。如果測試裝置未重新訂閱其主題觸發程序，則測試案例失敗。為了通過測試，裝置需要了解持久性工作階段已經結束，並在第二次連線對同一個觸發程序主題回傳一個新的 SUBSCRIBE 封包。

如果此測試案例對於某測試裝置通過，表示該裝置能在持久性工作階段過期時以預期的方式處理重新連線。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為至少 4 分鐘。測試裝置需要明確訂閱之前沒有訂閱的 TRIGGER_TOPIC。要通過測試案例，測試裝置必須在 CleanSession 旗標設為 false 下傳送 CONNECT 封包，並成功訂閱具有 QoS 1 的觸發程序主題。成功連線後， AWS IoT Core Device Advisor 會中斷裝置連線。中斷連線後， AWS IoT Core Device Advisor 允許裝置重新連線，且裝置預期會重新訂閱相同的 ，TRIGGER_TOPIC因為 AWS IoT Core Device Advisor 會終止持久性工作階段。

```
"tests":[
   {
      "name":"my_expired_persistent_session_test",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Expired_Persistent_Session",
         "version":"0.0.0"
      }
   }
]
```

影子

使用這些測試來驗證待測裝置是否正確使用 AWS IoT Device Shadow 服務。如需詳細資訊，請參閱 AWS IoT Device Shadow 服務。如果這些測試案例是在您的測試套件中設定，則在啟動套件執行時需要提供一個物件。

目前不支援 MQTT over WebSocket。

發布

「裝置在連接後發佈狀態 (Happy 案例)」

驗證裝置是否可以在連線至 後發佈其狀態 AWS IoT Core

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_shadow_publish_reported_state",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME",
         "REPORTED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         }
      },
      "test":{
         "id":"Shadow_Publish_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

可以提供 REPORTED_STATE，以在您的裝置連接之後，對其確切的影子狀態進行額外驗證。根據預設，此測試案例會驗證您的裝置發佈狀態。

如果未提供 *SHADOW_NAME*，則測試案例會依預設尋找發佈至未具名 (傳統) 影子類型之主題字首的訊息。如果您的裝置使用具名影子類型，請提供影子名稱。如需詳細資訊，請參閱在裝置中使用影子。

更新

「裝置將回報狀態更新為所需狀態 (Happy 案例)」

驗證您的裝置是否讀取所有收到的更新訊息，並同步處理裝置的狀態以符合所需的狀態屬性。您的裝置應該在同步處理之後發佈其最新回報狀態。如果您的裝置在執行測試之前已有現有影子，請確定針對測試案例設定的所需狀態，以及現有回報狀態並未符合。您可以識別 Device Advisor 所傳送的影子更新訊息，方法為查看影子文件中的 ClientToken 欄位，因為它將是 `DeviceAdvisorShadowTestCaseSetup`。

API 測試案例定義：

> **Note**
>
> `EXECUTION_TIMEOUT` 的預設值為 5 分鐘。我們建議的逾時值為 2 分鐘。

```
"tests":[
   {
      "name":"my_shadow_update_reported_state",
      "configuration": {
         "DESIRED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         },
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME"
      },
      "test":{
         "id":"Shadow_Update_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

`DESIRED_STATE` 應該至少有一個屬性和相關聯的值。

如果未提供 `SHADOW_NAME`，則測試案例會依預設尋找發佈至未具名 (傳統) 影子類型之主題字首的訊息。如果您的裝置使用具名影子類型，請提供影子名稱。如需詳細資訊，請參閱在裝置中使用影子。

任務執行

「裝置可以完成任務執行」

此測試案例可協助您驗證裝置是否可以使用 AWS IoT 任務接收更新，並發佈成功更新的狀態。如需 AWS IoT 任務的詳細資訊，請參閱 任務。

若要成功執行此測試案例，您需要授予裝置角色 兩個預留 AWS 主題。若要訂閱作業活動相關的訊息，請使用 notify 和 notify-next 主題。您的裝置角色必須將 PUBLISH 動作授予下列主題：

- $aws/things/thingName/jobs/jobId/get
- $aws/things/thingName/jobs/jobId/update

建議您將 SUBSCRIBE 和 RECEIVE 動作授予下列主題：

- $aws/things/thingName/jobs/get/accepted
- $aws/things/thingName/jobs/jobId/get/rejected
- $aws/things/thingName/jobs/jobId/update/accepted
- $aws/things/thingName/jobs/jobId/update/rejected

建議您將 SUBSCRIBE 動作授予下列主題：

- $aws/things/thingName/jobs/notify-next

如需有關這些保留主題的詳細資訊，請參閱AWS IoT Job 的保留主題。

目前不支援 MQTT over WebSocket。

API 測試案例定義：

> **Note**
>
> `EXECUTION_TIMEOUT` 的預設值為 5 分鐘。我們建議的逾時值為 3 分鐘。根據提供 AWS IoT 的任務文件或來源，調整逾時值（例如，如果任務需要很長時間才能執行，請定義測試案例的較長逾時值）。若要執行測試，需要有效的 AWS IoT 任務文件或現有的任務 ID。AWS IoT 任務文件可以 JSON 文件或 S3 連結的形式提供。如果提供 Job 文件，則可選擇性提供作業 ID。如果提供任務 ID，Device Advisor 將在代表您建立 AWS IoT 任務時使用該 ID。如果未提供 Job 文件，您可以提供與執行測試案例位於相同區域的現有 ID。在此情況下，Device Advisor 將在執行測試案例時使用該 AWS IoT 任務。

```
"tests": [
   {
      "name":"my_job_execution",
      "configuration": {
         // optional:
         // Test case will create a job task by using either JOB_DOCUMENT or
 JOB_DOCUMENT_SOURCE.
         // If you manage the job task on your own, leave it empty and provide the
 JOB_JOBID (self-managed job task).
         // JOB_DOCUMENT is a JSON formatted string
         "JOB_DOCUMENT": "{
            \"operation\":\"reboot\",
            \"files\" : {
               \"fileName\" : \"install.py\",
               \"url\" : \"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
bucket-name/key}\"
            }
         }",
         // JOB_DOCUMENT_SOURCE is an S3 link to the job document. It will be used
 only if JOB_DOCUMENT is not provided.
         "JOB_DOCUMENT_SOURCE": "https://s3.amazonaws.com/bucket-name/key",
         // JOB_JOBID is mandatory, only if neither document nor document source is
 provided. (Test case needs to know the self-managed job task id).
         "JOB_JOBID": "String",
         // JOB_PRESIGN_ROLE_ARN is used for the presign Url, which will replace the
 placeholder in the JOB_DOCUMENT field
         "JOB_PRESIGN_ROLE_ARN": "String",
         // Presigned Url expiration time. It must be between 60 and 3600 seconds,
 with the default value being 3600.
         "JOB_PRESIGN_EXPIRES_IN_SEC": "Long"
         "EXECUTION_TIMEOUT": "300", // in seconds
      },
      "test": {
         "id": "Job_Execution",
         "version": "0.0.0"
      }
   }
]
```

如需建立和使用任務文件的詳細資訊，請參閱任務文件。

許可和政策

您可以使用這些測試，來判斷與裝置憑證相連的政策是否遵循標準最佳實務。

目前不支援 MQTT over WebSocket。

「裝置憑證連接政策不包含萬用字元」

驗證與裝置相關聯的許可政策是否遵循最佳實務，且未授與裝置超過所需的許可。

API 測試案例定義：

> **Note**
>
> EXECUTION_TIMEOUT 的預設值為 1 分鐘。建議設定至少 30 秒的逾時時間。

```
"tests":[
   {
        "name":"my_security_device_policies",
        "configuration": {
            // optional:
            "EXECUTION_TIMEOUT":"60"    // in seconds
        },
        "test": {
            "id": "Security_Device_Policies",
            "version": "0.0.0"
        }
    }
]
```

長時間測試

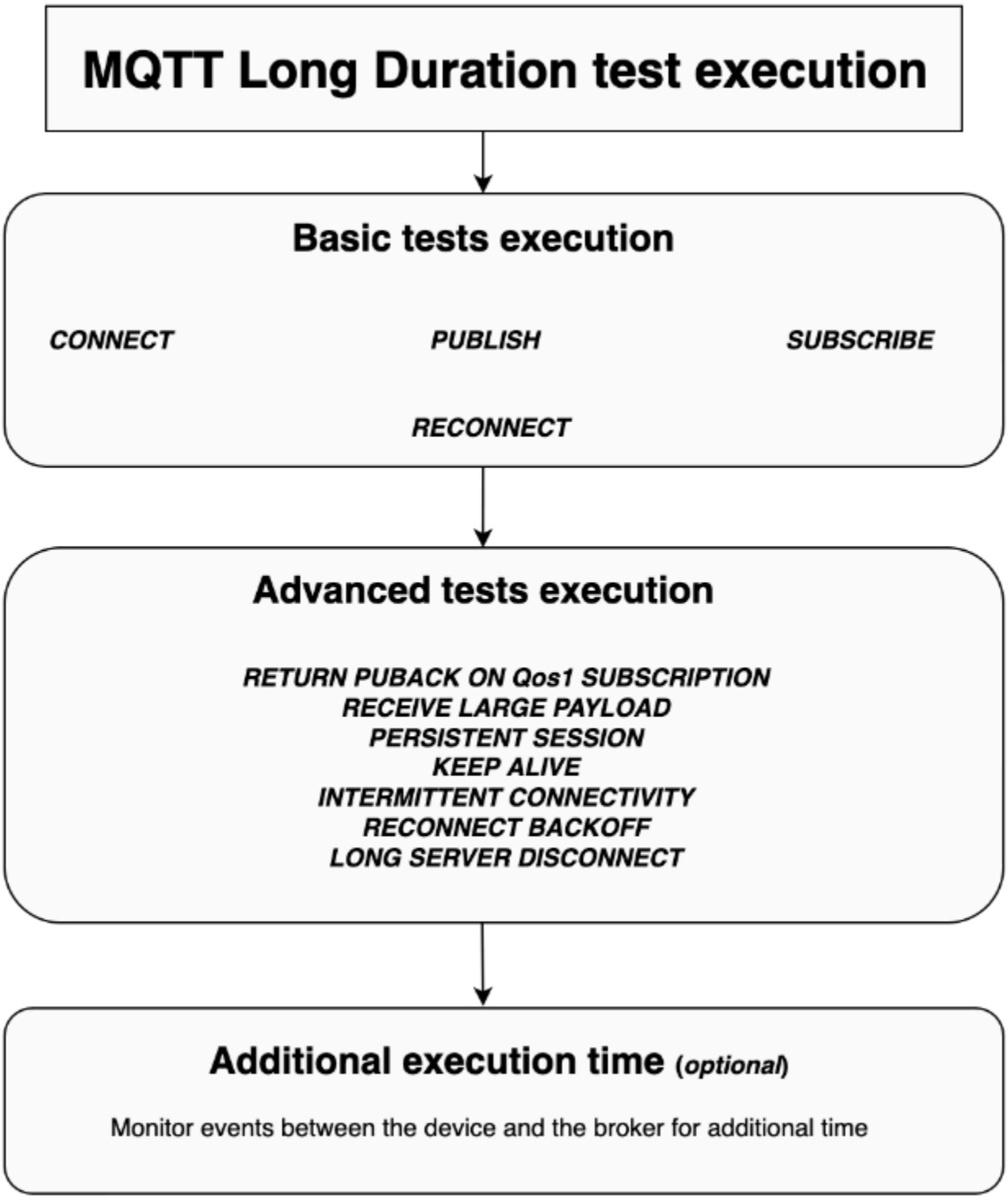
長時間測試是一款新型測試套件，可監測裝置在長時間運作期間的行為。相較於執行著重於裝置特定行為的個別測試，長時間測試會檢查裝置在其使用壽命內各種實際案例中的行為。Device Advisor 會以最有效率的順序來協調測試。該測試會產生結果和日誌，其中包括一份摘要日誌，詳載有關裝置效能的實用指標。

MQTT 長時間測試案例

在 MQTT 長時間測試案例中，起初會以 MQTT Connect、訂閱、發佈和重新連線等簡易案例的情境來觀察裝置行為。然後，在多種複雜的故障情境下觀察裝置，例如 MQTT 重新連線輪詢、長時間伺服器中斷連線以及間歇性連線。

MQTT 長時間測試案例執行流程

MQTT 長時間測試案例的執行分為三個階段：


MQTT Long Duration test execution
Basic tests execution
CONNECT
PUBLISH
SUBSCRIBE
RECONNECT
Advanced tests execution
RETURN PUBACK ON Qos1 SUBSCRIPTION
RECEIVE LARGE PAYLOAD
PERSISTENT SESSION
KEEP ALIVE
INTERMITTENT CONNECTIVITY
RECONNECT BACKOFF
LONG SERVER DISCONNECT
Additional execution time (optional)
Monitor events between the device and the broker for additional time

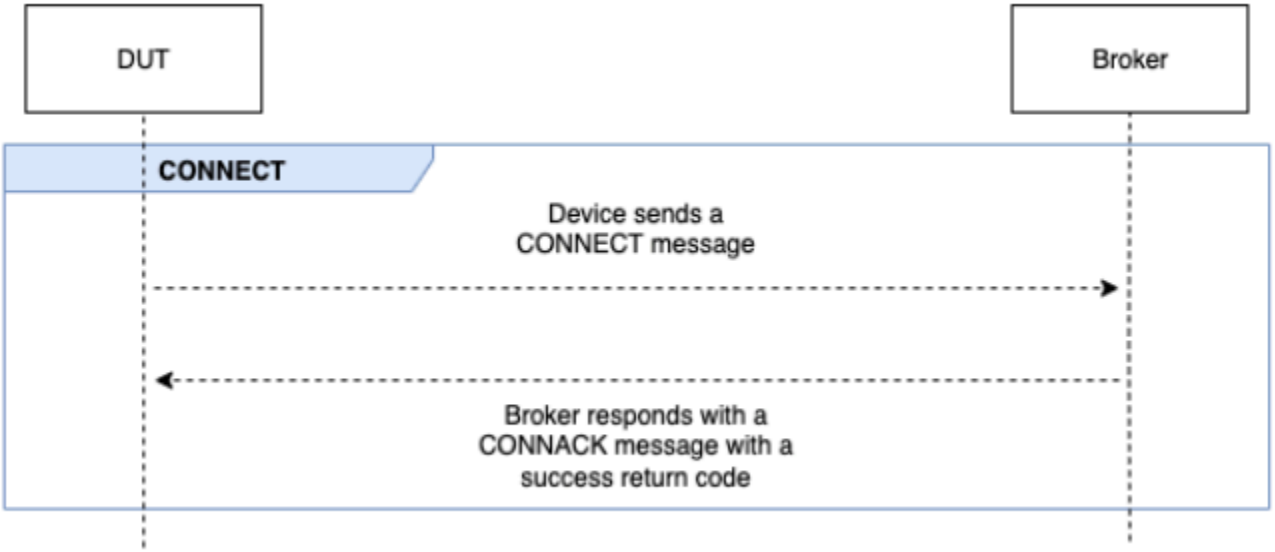
基本測試執行

此階段的測試案例會以並行方式執行簡易測試。測試將驗證裝置是否依據在組態中所做的選擇進行操作。

根據所選操作，一組基本測試可以包括以下內容：

CONNECT

此案例會驗證裝置是否能夠與代理程式成功建立連線。

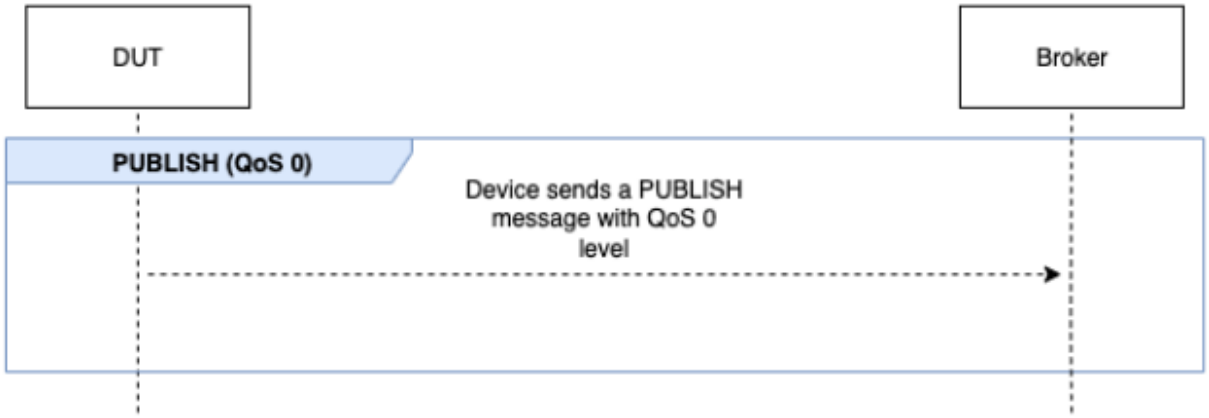


發佈

此案例會驗證裝置是否成功針對代理程式進行發佈。

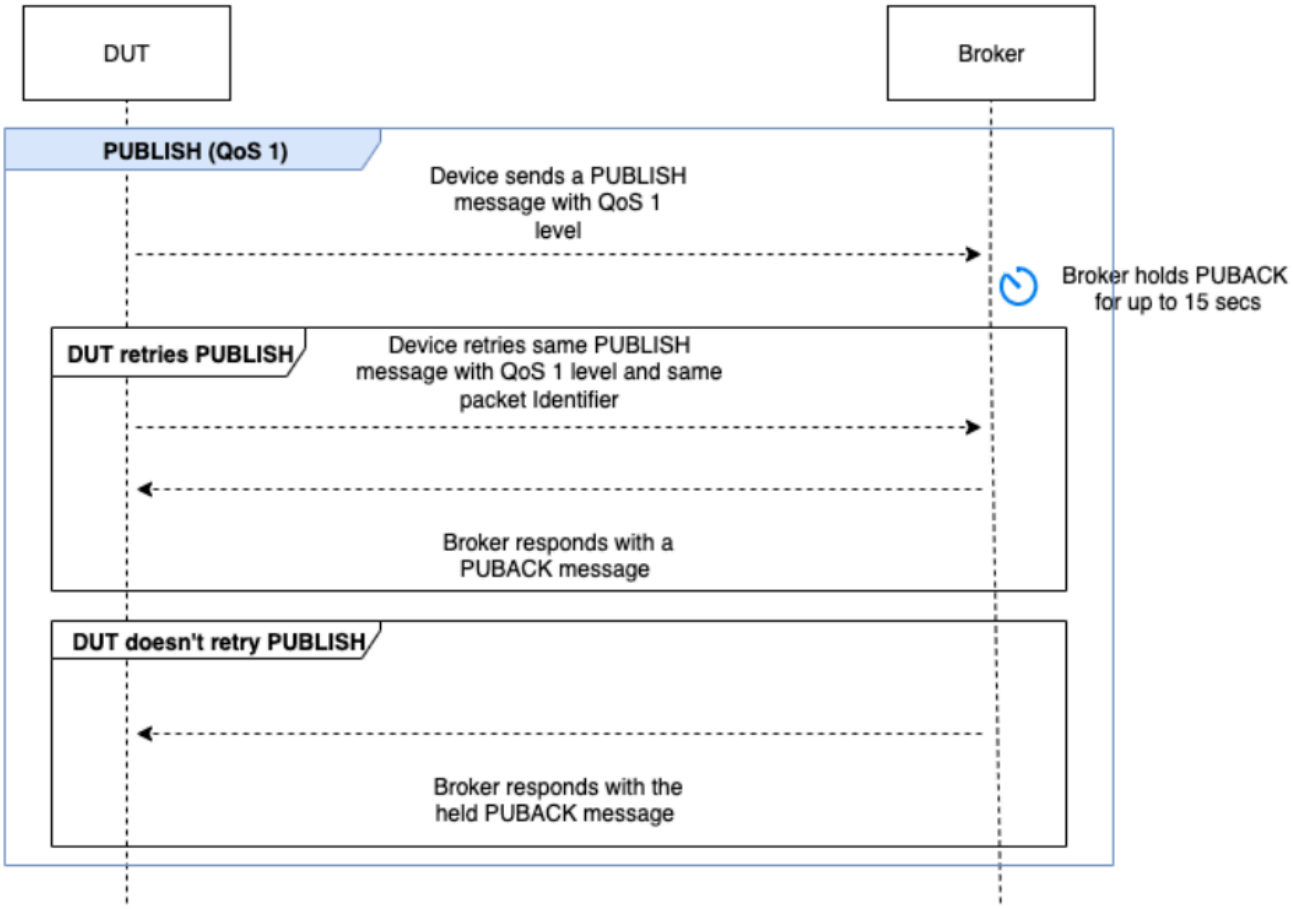
QoS 0

此測試案例會驗證裝置在以 QoS 0 發佈期間是否成功將 PUBLISH 訊息傳送至代理程式。測試不會等待裝置接收關於 PUBACK 的訊息。



QoS 1

在此測試案例中，裝置預計將向具有 QoS 1 的代理程式傳送兩條 PUBLISH 訊息。在第一條 PUBLISH 訊息之後，代理程式會等待最多 15 秒，然後再回應。裝置必須在 15 秒的時段中重新嘗試具有相同封包識別碼的原始 PUBLISH 訊息。在此情況下，代理程式會以 PUBACK 訊息回應，且測試會進行驗證。如果裝置沒有重試 PUBLISH，則原始 PUBACK 會傳送到裝置，並將測試標記為附帶警告的通過，同時提供系統訊息。測試執行期間，如果裝置失去連線並重新連接，則測試案例將重置而不會故障，並且裝置必須再次執行測試案例步驟。

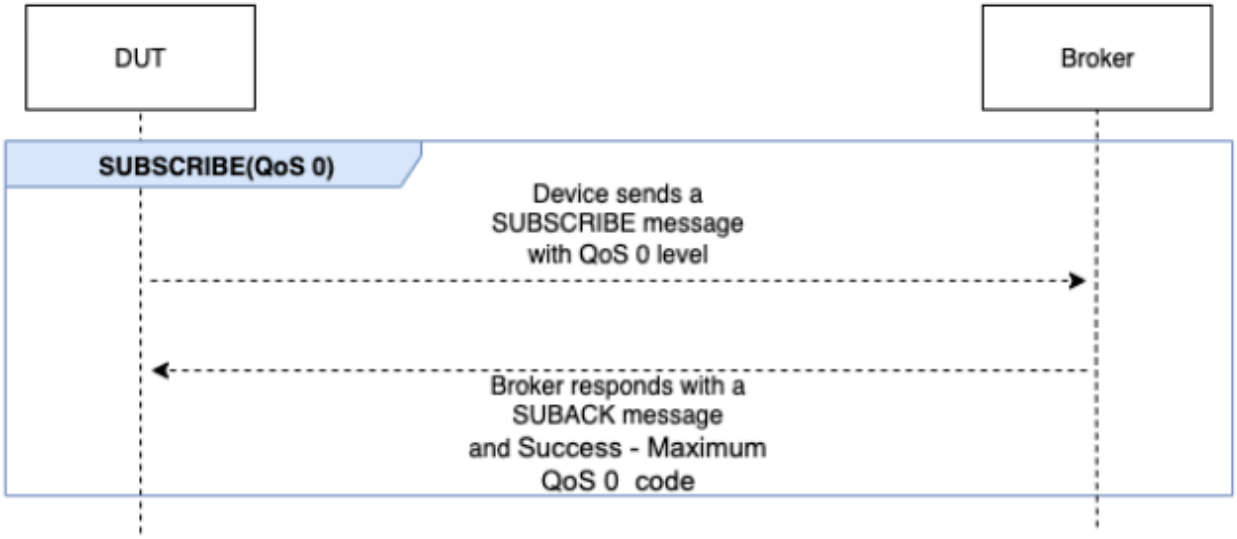


訂閱

此案例會驗證裝置是否成功針對代理程式進行訂閱。

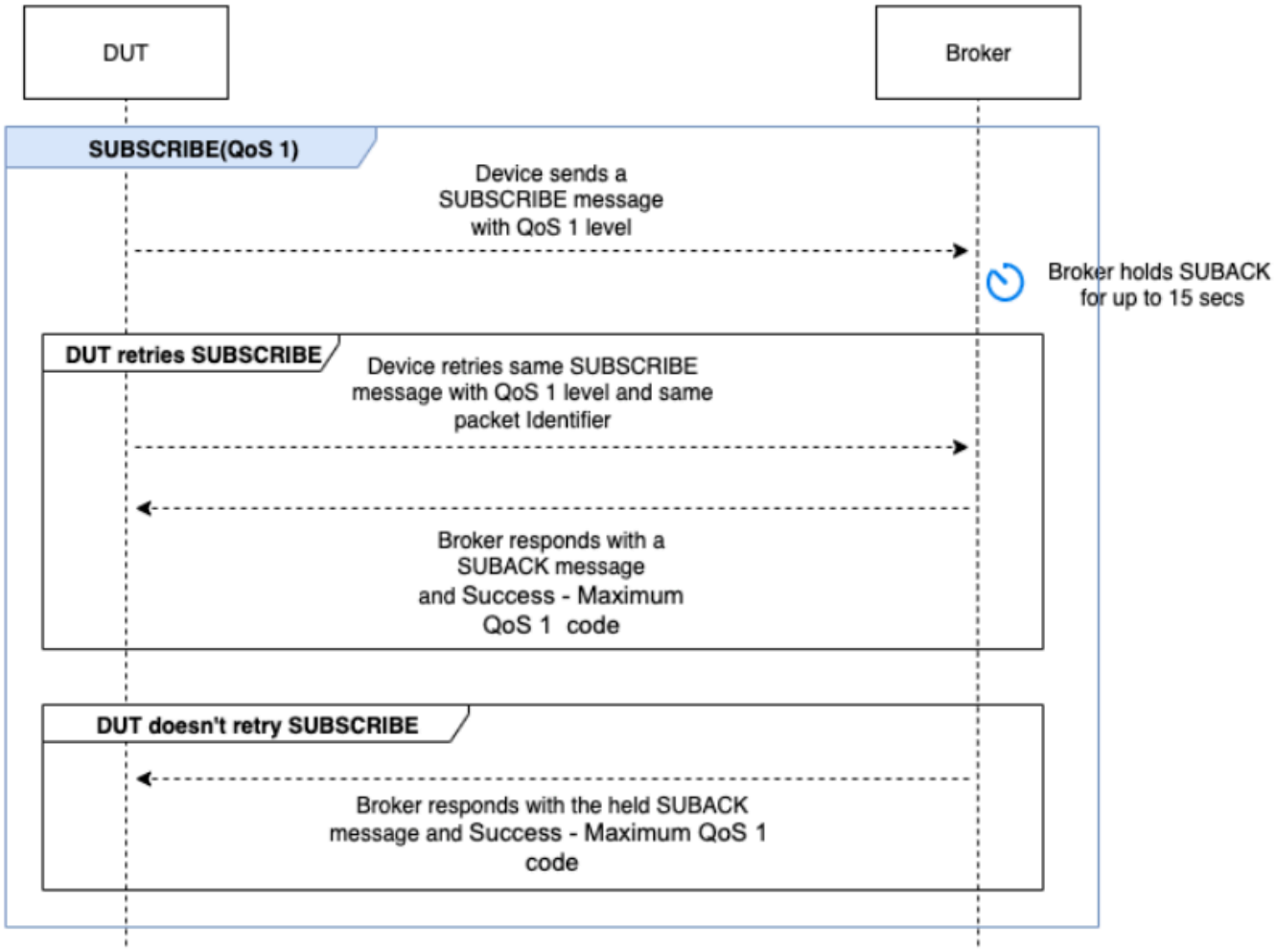
QoS 0

此測試案例會驗證裝置在以 QoS 0 訂閱期間是否成功將 SUBSCRIBE 訊息傳送至代理程式。測試不會等待裝置收到 SUBACK 訊息。



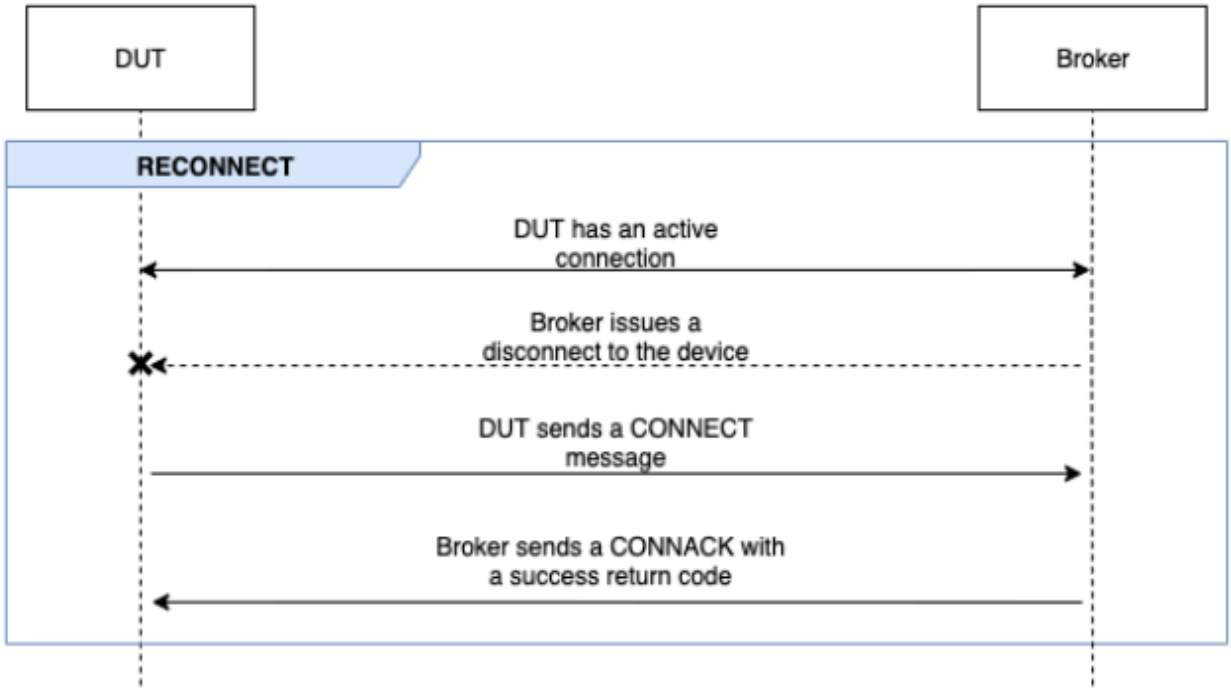
QoS 1

在此測試案例中，裝置預計將向具有 QoS 1 的代理程式傳送兩條 SUBSCRIBE 訊息。在第一條 SUBSCRIBE 訊息之後，代理程式會等待最多 15 秒，然後再回應。裝置必須在 15 秒的時段中重新嘗試具有相同封包識別碼的原始 SUBSCRIBE 訊息。在此情況下，代理程式會以 SUBACK 訊息回應，且測試會進行驗證。如果裝置沒有重試 SUBSCRIBE，則原始 SUBACK 會傳送到裝置，並將測試標記為附帶警告的通過，同時提供系統訊息。測試執行期間，如果裝置失去連線並重新連接，則測試案例將重置而不會故障，並且裝置必須再次執行測試案例步驟。



重新連線

此案例會驗證裝置在從成功的連線中斷之後，是否成功與代理程式重新連線。如果先前已在測試套件期間連線多次，則 Device Advisor 不會中斷裝置連線。相反，它會將測試標記為通過。



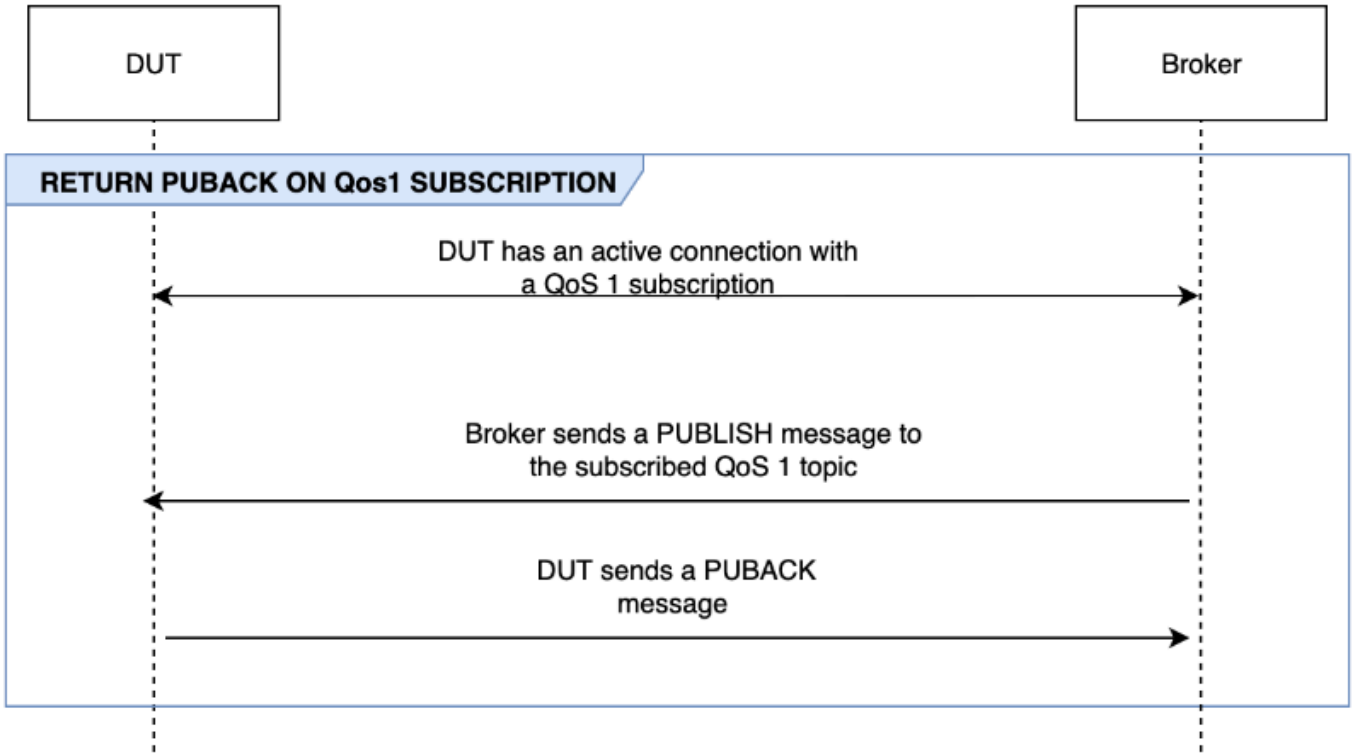
進階測試執行

此階段的測試案例會以序列方式執行較複雜的測試，以驗證裝置是否遵循最佳實務。這些進階測試可供選擇，若無需要可以選擇不執行。根據案例需求，每項進階測試都各有專屬的逾期值。

RETURN PUBACK ON QoS 1 SUBSCRIPTION

> **Note**
>
> 只有當您的裝置能夠執行 QoS 1 訂閱時，才可選取此案例。

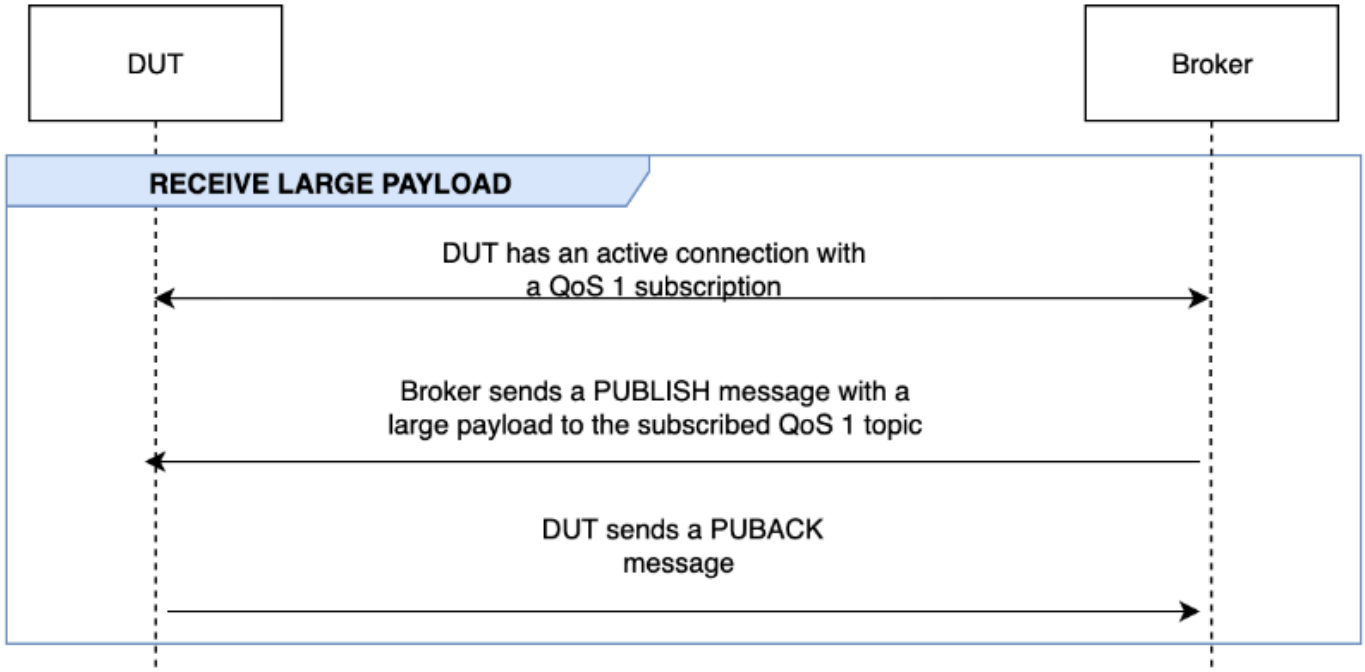
此案例會驗證在裝置訂閱主題並收到來自代理人的 PUBLISH 訊息之後，是否會傳回 PUBACK 訊息。



RECEIVE LARGE PAYLOAD

> **Note**
>
> 只有當您的裝置能夠執行 QoS 1 訂閱時，才可選取此案例。

此案例會驗證裝置在收到具有大型承載的 QoS 1 主題的代理程式發出的 PUBLISH 訊息後，是否會以 PUBACK 訊息進行回應。可以使用 LONG_PAYLOAD_FORMAT 選項設定預期承載的格式。



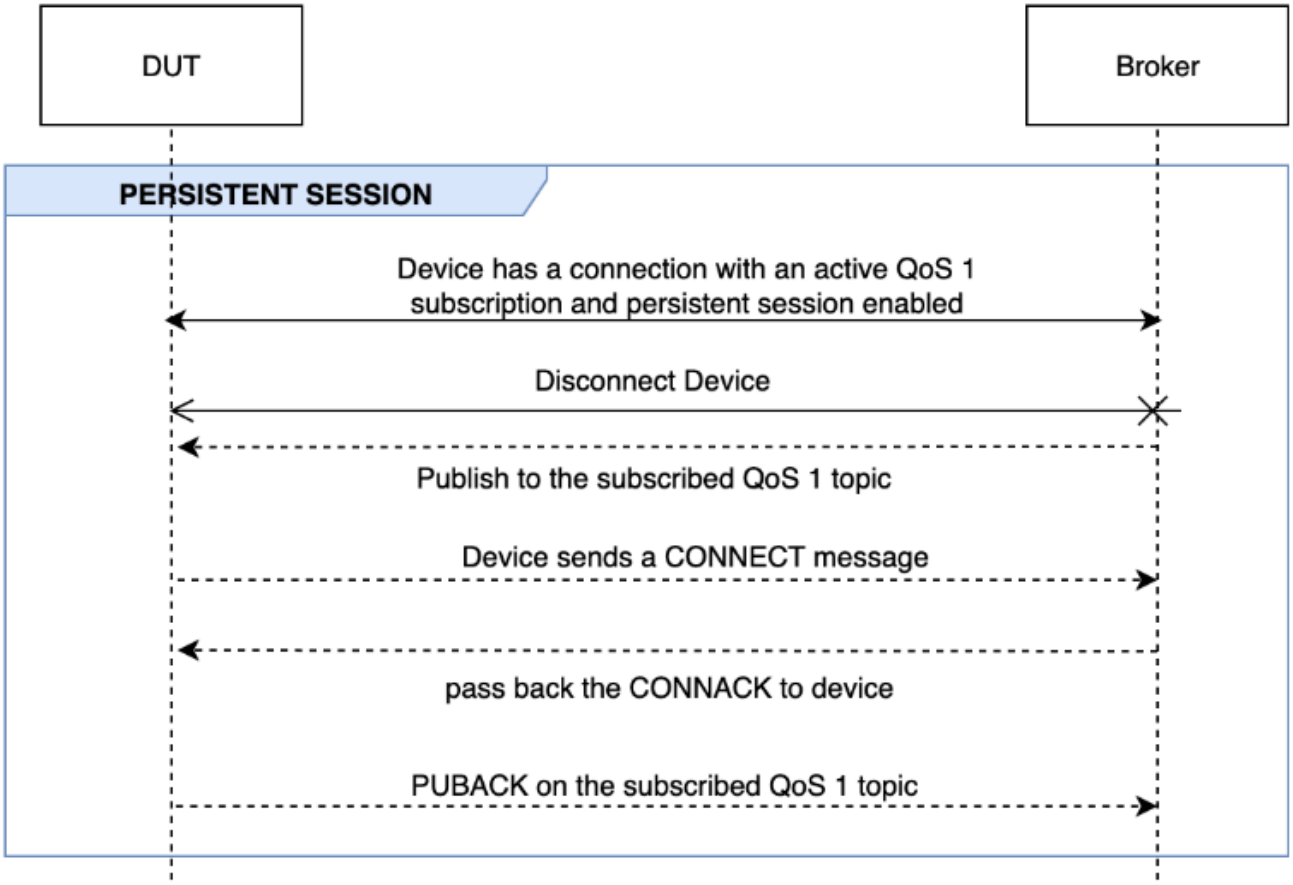
持久性工作階段

> **Note**
>
> 只有當您的裝置能夠執行 QoS 1 訂閱且可以維持持久性工作階段時，才可選取此案例。

此案例會驗證維持持久性工作階段時的裝置行為。測試會驗證是否滿足下列條件：

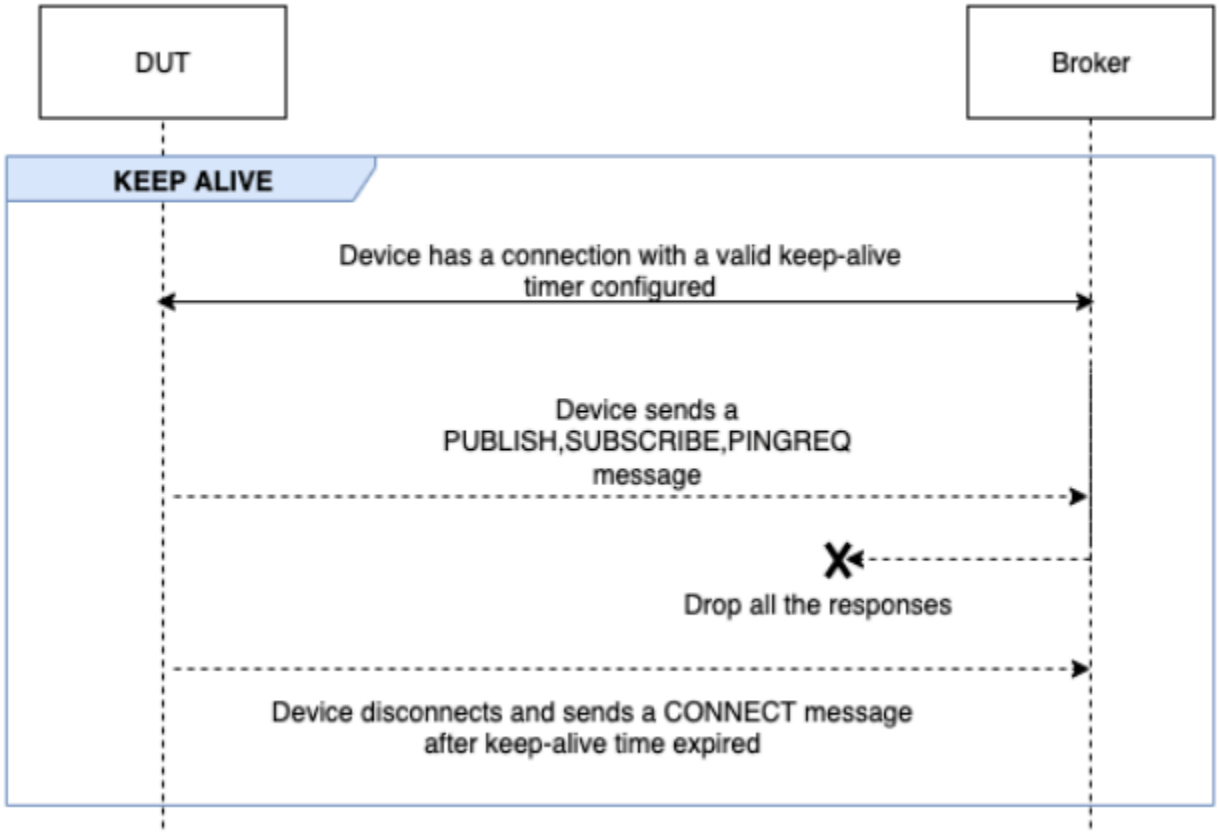
- 裝置連線至具有作用中 QoS 1 訂閱且已啟用持久性工作階段的代理程式。
- 裝置在工作階段期間成功中斷與代理程式的連線。
- 裝置會重新連線至代理程式，並繼續訂閱其觸發主題，而不會明確重新訂閱這些主題。
- 裝置成功接收代理程式為其訂閱主題所儲存的訊息，並如預期執行。

如需 AWS IoT 持久性工作階段的詳細資訊，請參閱使用 MQTT 持久性工作階段。



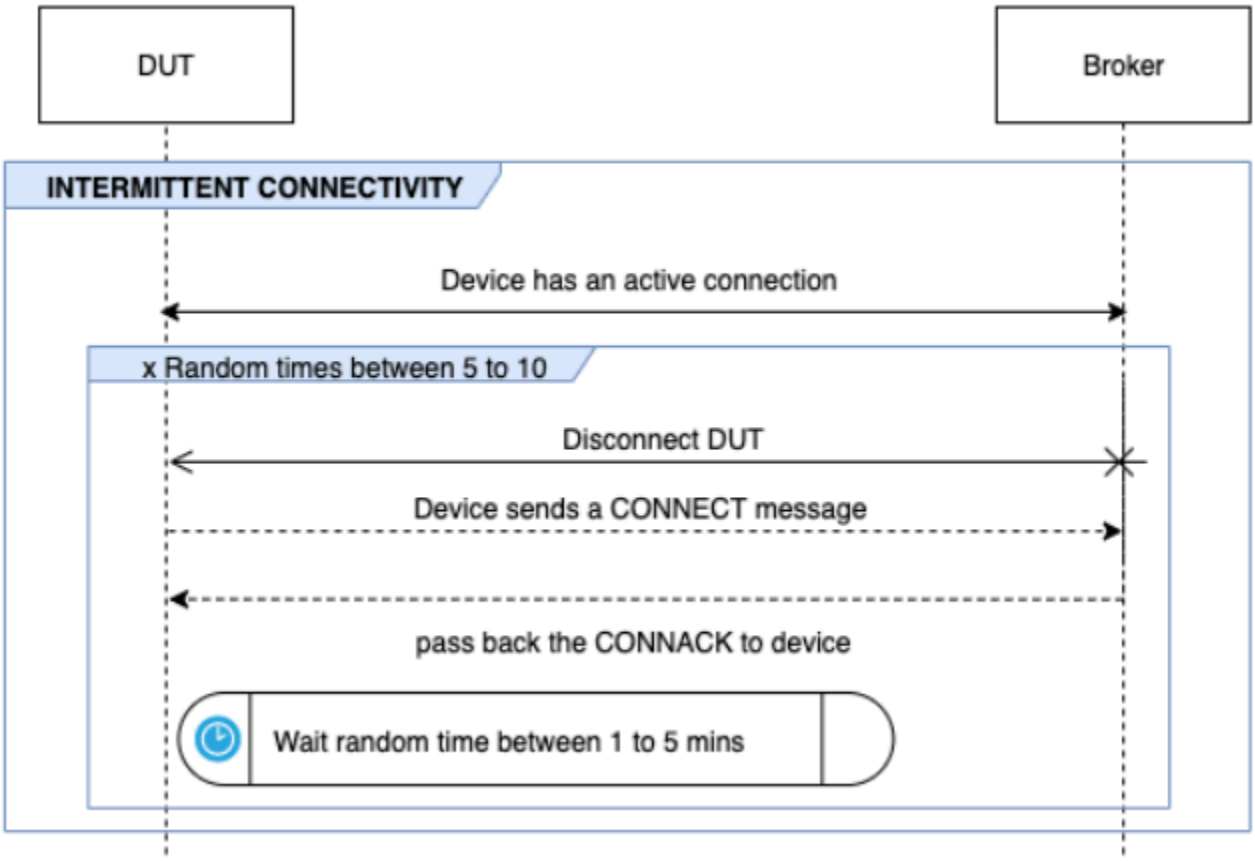
Keep-Alive

此案例會驗證裝置在未收到代理程式的 Ping 回應後是否能成功中斷連線。必須為連線設定有效的保持連線計時器。在此測試中，代理程式會封鎖所有針對 PUBLISH、SUBSCRIBE 和 PINGREQ 訊息而傳送的回應。其也會驗證待測裝置是否中斷與 MQTT 的連線。



INTERMITTENT CONNECTIVITY

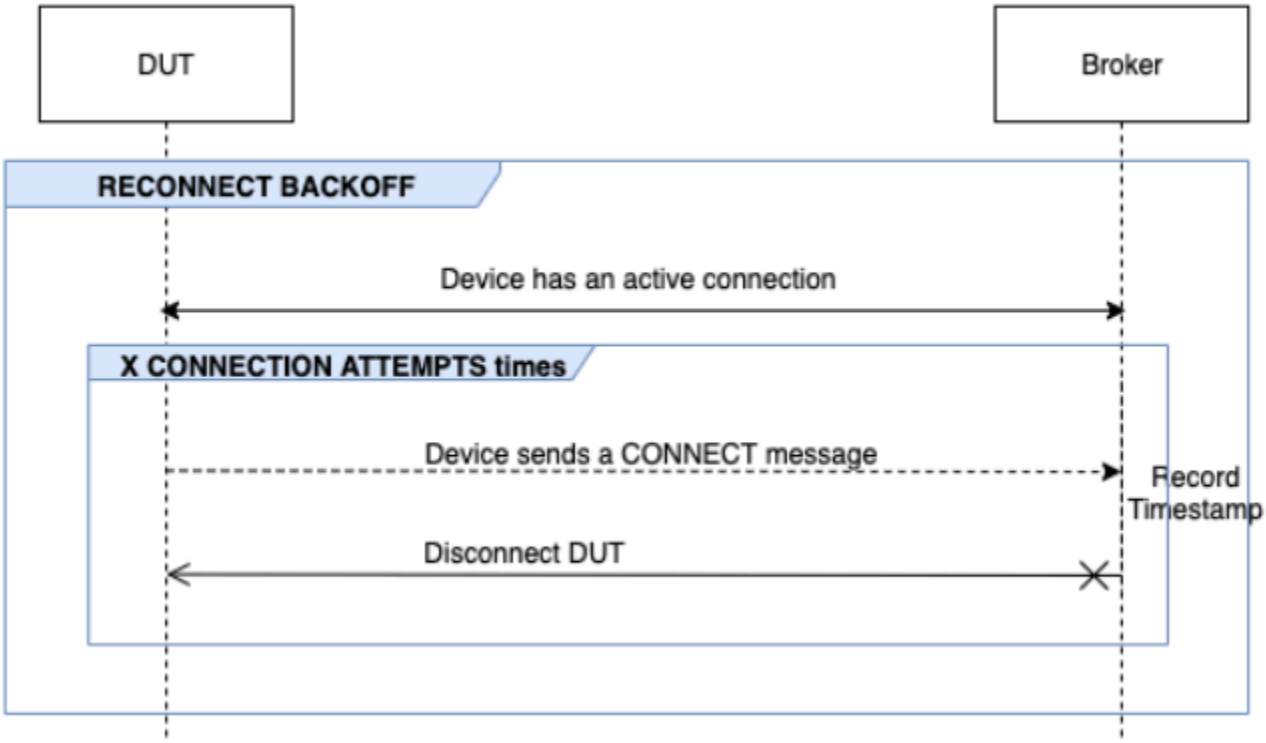
此案例會驗證代理程式在隨機間隔內與裝置中斷連線之後，裝置是否可以恢復與代理程式的連線。



RECONNECT BACKOFF

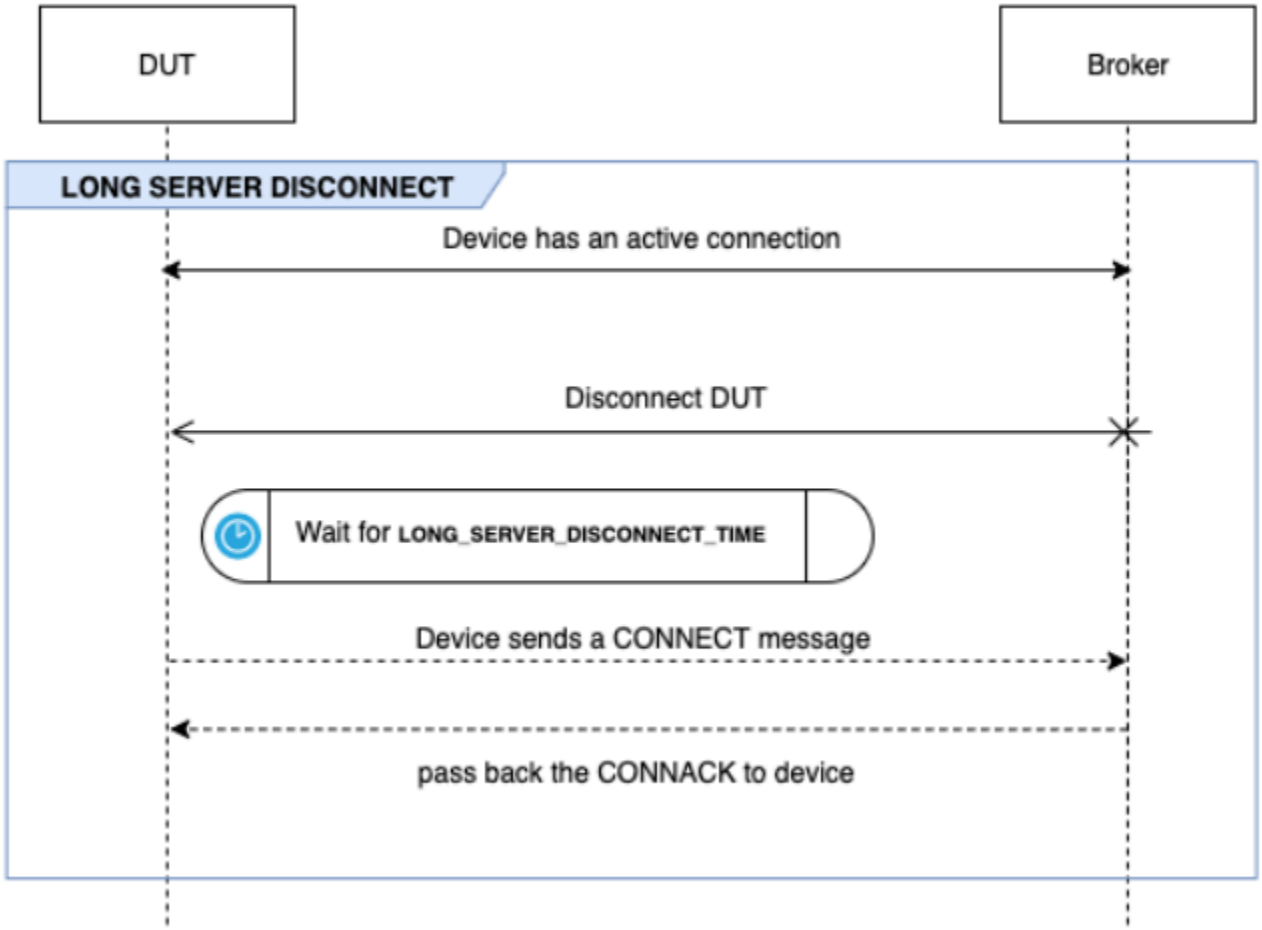
此案例會驗證代理程式多次中斷連線後，裝置是否會實作退避機制。Device Advisor 會將退避類型報告為指數、抖動、線性或常數。您可以使用 BACKOFF_CONNECTION_ATTEMPTS 選項來設定退避嘗試次數。預設值為 5。可以設定介於 5 到 10 之間的值。

若要通過此測試，建議對受測裝置實作指數退避和抖動機制。



長時間伺服器中斷連線

此案例會驗證在代理程式長時間 (最多 120 分鐘) 中斷與裝置的連線之後，裝置是否可以成功重新連線。可以使用 LONG_SERVER_DISCONNECT_TIME 選項來設定伺服器中斷連線的時間。預設值為 120 分鐘。此值可以設定的範圍介於 30 至 120 分鐘。



額外執行時間

額外執行時間是指測試從完成上述所有測試之後到結束測試案例之前所等待的時間。客戶可使用此額外時段來監控裝置並記錄裝置與代理程式的所有通訊。可以使用 ADDITIONAL_EXECUTION_TIME 選項來設定額外執行時間。此選項依預設為 0 分鐘，可設定的範圍介於 0 到 120 分鐘。

MQTT 長時間測試組態選項

為 MQTT 長時間測試提供的所有組態選項均非強制性。下列選項可供使用：

操作

裝置執行的操作清單，例如 CONNECT、PUBLISH 和 SUBSCRIBE。測試案例會依據指定的操作來執行情境案例。系統會將未指定的操作假設為有效。

```
{
"OPERATIONS": ["PUBLISH", "SUBSCRIBE"]
```

```
//by default the test assumes device can CONNECT
}
```

案例

根據所選操作，測試案例會執行情境案例來驗證裝置行為。案例有兩種類型：

- 基本案例屬於簡易測試，用於驗證裝置是否可以執行在組態中選擇的操作。這些條件會根據組態中指定的操作預先選取。組態中不再需要輸入。
- 進階案例是對裝置執行較複雜的情境案例，以驗證裝置在真實世界條件下是否遵循最佳實務。這些選項均非強制性質，可以作為案例陣列傳遞給測試套件的組態輸入項。

```
{
    "SCENARIOS": [      // list of advanced scenarios
                "PUBACK_QOS_1",
                "RECEIVE_LARGE_PAYLOAD",
                "PERSISTENT_SESSION",
                "KEEP_ALIVE",
                "INTERMITTENT_CONNECTIVITY",
                "RECONNECT_BACK_OFF",
                "LONG_SERVER_DISCONNECT"
    ]
}
```

BASIC_TESTS_EXECUTION_TIME_OUT:

測試案例等待所有基本測試完成的時間上限。預設值為 60 分鐘。此值可以設定的範圍介於 30 至 120 分鐘。

LONG_SERVER_DISCONNECT_TIME:

在「長時間伺服器中斷連線」測試期間，測試案例中斷並恢復與裝置連線所花費的時間。預設值為 60 分鐘。此值可以設定的範圍介於 30 至 120 分鐘。

ADDITIONAL_EXECUTION_TIME:

若設定此選項，系統會在完成所有測試後提供一個時段，用以監控裝置與代理程式之間的事件。預設值為 0 分鐘。此值可以設定的範圍介於 0 至 120 分鐘。

BACKOFF_CONNECTION_ATTEMPTS:

此選項可設定測試案例中斷與裝置連線的次數。重新連線退避測試會使用此功能。預設值為 5 次嘗試。此值可以設定的範圍介於 5 至 10 分鐘。

LONG_PAYLOAD_FORMAT:

當測試案例發佈到裝置訂閱的 QoS 1 主題時，裝置所期望的訊息承載格式。

API 測試案例定義：

```
{
"tests":[
   {
      "name":"my_mqtt_long_duration_test",
      "configuration": {
         // optional
         "OPERATIONS": ["PUBLISH", "SUBSCRIBE"],
         "SCENARIOS": [
            "LONG_SERVER_DISCONNECT",
            "RECONNECT_BACK_OFF",
            "KEEP_ALIVE",
            "RECEIVE_LARGE_PAYLOAD",
            "INTERMITTENT_CONNECTIVITY",
            "PERSISTENT_SESSION",
         ],
         "BASIC_TESTS_EXECUTION_TIMEOUT": 60, // in minutes (60 minutes by default)
         "LONG_SERVER_DISCONNECT_TIME": 60,   // in minutes (120 minutes by default)
         "ADDITIONAL_EXECUTION_TIME": 60,     // in minutes (0 minutes by default)
         "BACKOFF_CONNECTION_ATTEMPTS": "5",
         "LONG_PAYLOAD_FORMAT":"{"message":"${payload}"}"
      },
      "test":{
         "id":"MQTT_Long_Duration",
         "version":"0.0.0"
      }
   }
 ]
}
```

MQTT 長時間測試案例摘要日誌

MQTT 長時間測試案例的執行時間較常規測試案例更長。會提供個別摘要日誌，其中列出執行期間的裝置連線、發佈和訂閱等重要事件。詳細資訊包括已測試的項目、未測試的項目以及失敗的項目。測試功能會在日誌結尾列出測試案例執行期間所發生全部事件的摘要。其中包含：

- 在裝置上設定的保持連線計時器。

- 裝置上設定的持久性工作階段旗標。
- 裝置在測試執行期間的連線次數。
- 裝置重新連線退避類型 (若已通過重新連線退避測試的驗證)。
- 在測試案例執行期間作為裝置發佈目標的主題。
- 裝置在測試案例執行期間訂閱的主題。

AWS IoT Core 裝置位置

使用 AWS IoT Core 裝置位置功能之前，請檢閱此功能的條款與條件。請注意， AWS 可能會傳輸您的地理位置搜尋請求參數，例如用於執行搜尋的位置資料，以及其他資訊給您選擇的第三方資料提供者，這些資訊可能位於 AWS 區域 您目前使用的 之外。要使用的第三方供應商和求解器會根據收到的輸入承載進行選擇。如需詳細資訊，請參閱 AWS 服務條款。

使用 AWS IoT Core 裝置位置，使用第三方求解器測試 IoT 裝置的位置。求解器是由第三方提供的演算法，可解析測量資料並估算裝置的位置。藉由識別裝置的位置，您就可以在現場對其進行追蹤和偵錯，進而排解任何疑難問題。

針對收集自各種來源的測量資料進行解析，並將地理位置資訊報告為 GeoJSON 承載。GeoJSON 格式可用於對地理資料結構進行編碼。承載含有裝置位置的緯度和經度坐標 (以世界大地坐標系統 (WGS84) 為準)。

主題

- 測量類型和求解器
- AWS IoT Core 裝置位置的運作方式
- 如何使用 AWS IoT Core 裝置位置
- 解析 IoT 裝置的位置
- 使用裝置位置 MQTT 主題解決 AWS IoT Core 裝置位置
- 位置求解器和裝置承載
- AWS IoT Core 裝置位置的進階組態

測量類型和求解器

AWS IoT Core Device Location 會與第三方供應商合作，以解析測量資料並提供預估的裝置位置。下表顯示測量類型和第三方位置求解器，以及支援裝置的相關資訊。如需有關 LoRaWAN 裝置和為其設定裝置位置的資訊，請參閱設定 LoRaWAN 資源的位置。

> **Note**
>
> 一般 IoT 裝置和 Sidewalk 裝置可以使用裝置位置 MQTT 主題來取得位置資訊。對於 Wi-Fi、行動和 IP 地址測量類型，如果裝置以定義的 GeoJSON 格式將測量資料發佈到預留主題，

AWS IoT Core 則裝置位置可以解析裝置的位置。對於 GNSS 和多影格 GNSS 測量類型，裝置必須具有 LR11xx 晶片來掃描測量資料，以使用 GNSS 或多影格 GNSS 求解器取得解析的位置資訊。如需取得 LoRaWAN 裝置位置資訊的資訊，請參閱 AWS IoT Wireless 文件中的設定 LoRaWAN 資源的位置。

測量類型和求解器

測量類型	第三方求解器	支援的裝置
Wi-Fi 存取點	Wi-Fi 型的求解器	一般 IoT 裝置、LoRaWAN 和 Amazon Sidewalk 裝置
行動網路無線電塔：GSM、LTE、CDMA、SCDMA、WCMDA 和 TD-SCDMA 資料	行動網路型求解器	一般 IoT 裝置、LoRaWAN 和 Amazon Sidewalk 裝置
IP 位址	IP 反向查詢求解器	透過 TCP/IP 連線的任何 IoT 裝置
GNSS 掃描資料 (NAV 訊息)	GNSS 求解器	一般 IoT 裝置、LoRaWAN 和 Amazon Sidewalk 裝置
多影格 GNSS 掃描資料 （來自相同裝置的多個 NAV 訊息）	多影格 GNSS 求解器	一般 IoT 裝置
低功耗藍牙 (BLE)	以 BLE 為基礎的求解器	Amazon Sidewalk 裝置

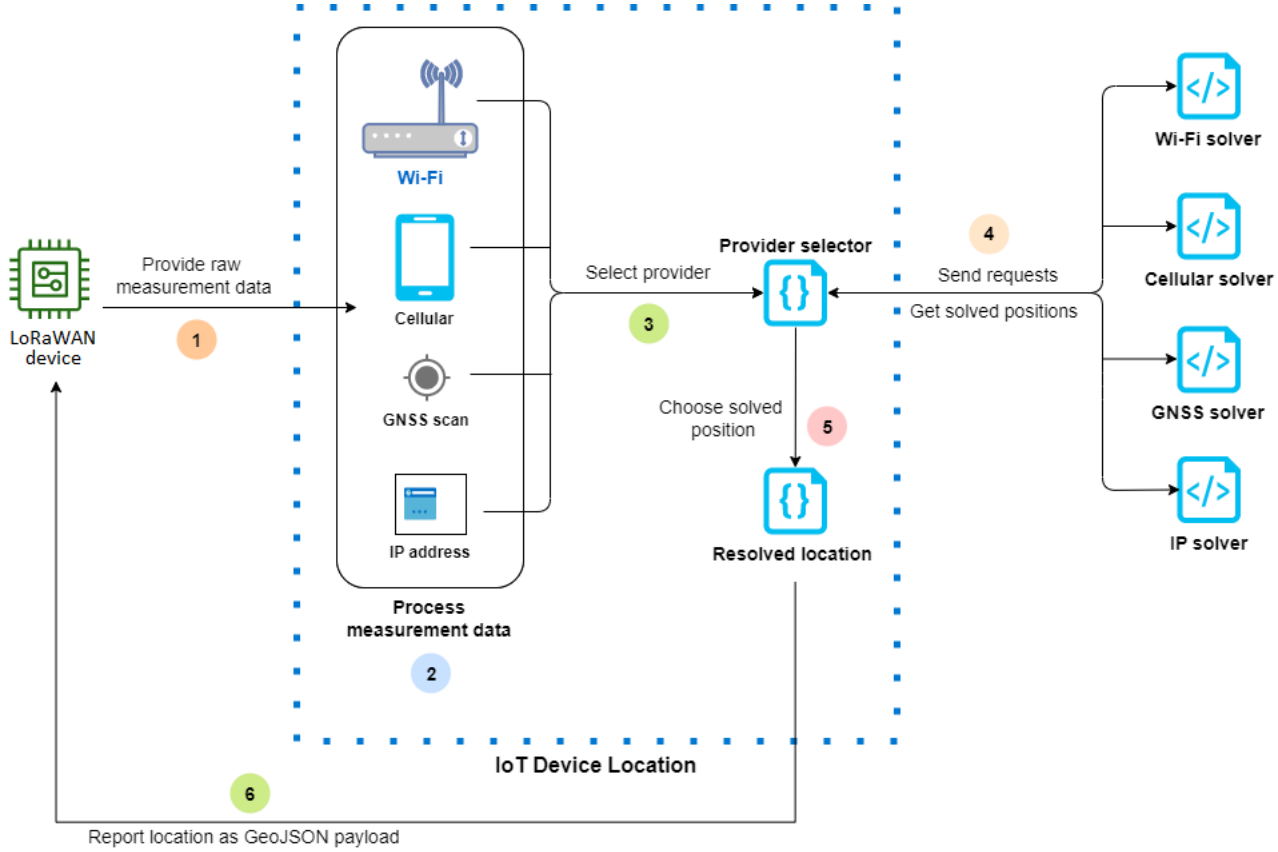
Note

GNSS 和多影格 GNSS 測量類型是互斥的。單一請求可以使用 `Gnss`測量類型從單一 NAV 訊息擷取解析位置，或使用 `GnssMultiFrame`測量類型從同一裝置從多個 NAV 訊息擷取解析位置，但不能同時包含在同一請求中。

如需有關位置求解器的詳細資訊及顯示各種測量類型的裝置承載範例，請參閱 位置求解器和裝置承載。

AWS IoT Core 裝置位置的運作方式

下圖顯示 AWS IoT Core Device Location 如何收集測量資料並解析裝置的位置資訊。



下列步驟顯示 AWS IoT Core 裝置位置的運作方式。

1. 接收測量資料

   先從裝置傳送與裝置位置相關的原始測量資料。將測量資料指定為 JSON 承載。

2. 處理測量資料

   系統會處理測量資料，而 AWS IoT Core Device Location 會選擇要使用的測量資料，可以是 Wi-Fi、行動網路、GNSS 掃描、多影格 GNSS 掃描或 IP 地址資訊。

3. 選擇求解器

   根據測量資料選擇第三方求解器。例如，如果測量資料包含 Wi-Fi 和 IP 地址資訊，則會選擇 Wi-Fi 求解器和 IP 反向查詢求解器。

4. 取得解析的位置

   API 請求會傳送至請求解析位置的求解器提供者。接著 AWS IoT Core ，Device Location 會從求解器取得預估的地理位置資訊。

5. 選擇解析的位置

   已解析的位置資訊及其準確性會進行比較，而 AWS IoT Core Device Location 會選擇準確度最高的地理位置結果。

6. 輸出位置資訊

   地理位置資訊會以 GeoJSON 承載的形式傳送給您。承載包含 WGS84 地理座標、準確度資訊、可信度，以及取得解析位置的時間戳記。

如何使用 AWS IoT Core 裝置位置

下列步驟說明如何使用 AWS IoT Core 裝置位置。

1. 提供測量資料

   將與裝置位置相關的原始測量資料指定為 JSON 承載。若要擷取承載測量資料，請前往裝置日誌，或使用 CloudWatch Logs，然後複製承載資料資訊。JSON 承載必須包含一或多種資料測量類型。如需顯示各種求解器承載格式的範例，請參閱 位置求解器和裝置承載。

2. 解析位置資訊

   使用 AWS IoT 主控台中的裝置位置頁面或 GetPositionEstimate API 操作，傳遞承載測量資料並解析裝置位置。 AWS IoT Core Device Location 接著會選擇準確度最高的求解器，並報告裝置位置。如需詳細資訊，請參閱解析 IoT 裝置的位置。

3. 複製位置資訊

   驗證 AWS IoT Core 裝置位置解析並回報為 GeoJSON 承載的地理位置資訊。您可以複製承載以搭配您的應用程式和其他 AWS 服務使用。例如，您可以使用 Location AWS IoT 規則動作，將地理位置資料傳送至 Amazon Location Service。

下列主題說明如何使用 AWS IoT Core 裝置位置和裝置位置承載的範例。

- 解析 IoT 裝置的位置
- 位置求解器和裝置承載

解析 IoT 裝置的位置

使用 AWS IoT Core 裝置位置從您的裝置解碼測量資料，並使用第三方求解器解析裝置位置。系統會產生解析後的位置作為 GeoJSON 承載，其中包含地理坐標和準確性資訊。您可以從 AWS IoT 主控台、AWS IoT 無線 API 或 解析裝置的位置 AWS CLI。

主題

- 解析裝置位置 (主控台)
- 解析裝置位置 (API)
- 對解析位置時發生的錯誤進行疑難排解

解析裝置位置 (主控台)

若要解析裝置位置 (主控台)

1. 前往 AWS IoT 主控台中的裝置位置頁面。
2. 從您的裝置日誌或 CloudWatch Logs 日誌取得承載測量資料，然後在透過承載解析位置區段輸入這些資料。

   以下程式碼顯示 JSON 承載的範例。承載包含行動網路和 Wi-Fi 測量資料。如果您的承載包含其他類型的測量資料，則會使用具有最佳準確度的求解器。如需詳細資訊和承載範例，請參閱 the section called “位置求解器和裝置承載”。

   > **Note**
   >
   > JSON 承載必須包含一或多種資料測量類型。

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress": "54.240.198.35"
    },
    "WiFiAccessPoints": [{
        "MacAddress": "A0:EC:F9:1E:32:C1",
        "Rss": -77
    }],
    "CellTowers": {
```

```
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
                    "GeranCid": 1
                }
            }]
        }],
        "Wcdma": [{
            "Mcc": 262,
            "Mnc": 7,
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
            "Rsrp": -50,
            "Rsrq": -5,
```

```
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    },
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 68
        }
    }
}
```

3. 若要解析位置資訊，請選擇 Resolve (解析)。

位置資訊的類型為 blob，並以採用 GeoJSON 格式的承載形式傳回，此格式可用於對地理資料結構進行編碼。該承載包含：

- WGS84 地理座標，其中包括緯度和經度資訊。其亦可能包括海拔高度資訊。
- 報告的位置資訊類型，例如 Point (點)。點位置類型將位置表示為 WGS84 緯度和經度，並編碼為 GeoJSON 點的型態。
- 水平和垂直準確度資訊，指出求解器預估的位置資訊與實際裝置位置之間的差異 (單位公尺)。
- 可信度，此資料表示位置回應估計值的不確定性。預設值為 0.68，表示實際裝置位置處於預估位置不確定半徑內的機率為 68%。您可以使用 AdvancedConfiguration 參數自訂此值。如需詳細資訊，請參閱AWS IoT Core 裝置位置的進階組態。
- 裝置所在的城市、州、國家/地區和郵遞區號。只有在使用 IP 反向查詢求解器時，才會報告此資訊。
- 時間戳記資訊對應於解析位置的日期和時間。其使用的是 Unix 時間戳記格式。

以下程式碼顯示藉由解析位置而傳回的 GeoJSON 裝載範例。

> **Note**
>
> 如果 AWS IoT Core Device Location 在嘗試解析位置時回報錯誤，您可以疑難排解錯誤並解析位置。如需詳細資訊，請參閱對解析位置時發生的錯誤進行疑難排解。

```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "timestamp": "2026-04-07T01:16:39.971767716Z"
    }
}
```

4. 前往資源位置區段，並驗證 AWS IoT Core 裝置位置 報告的地理位置資訊。您可以複製承載以與其他應用程式和 AWS 服務搭配使用。例如，您可以使用 Location 將地理位置資料傳送至 Amazon Location Service。

解析裝置位置 (API)

若要使用 AWS IoT Wireless API 解析裝置位置，請使用 GetPositionEstimate API 操作或 get-position-estimate CLI 命令。將承載測量資料指定為輸入項，然後執行 API 操作以解析裝置位置。

> **Note**
>
> GetPositionEstimate API 操作不會儲存任何裝置或狀態資訊，也無法用於擷取歷史位置資料。此功能會執行一次性操作，以解析測量資料並產生位置估計值。若要擷取位置資訊，在每次執行此 API 操作時，都必須指定承載資訊。

下列命令範例顯示如何使用此 API 操作解析位置。

> **Note**
>
> 執行 get-position-estimate CLI 命令時，您必須將輸出 JSON 檔案指定為第一個輸入。這個 JSON 檔案會以 GeoJSON 格式儲存從 CLI 回應取得的估計位置資訊。例如，下列命令會將位置資訊儲存在 *locationout.json* 檔案中。

> **Note**
>
> GnssMultiFrame 測量類型和 AdvancedConfiguration 參數僅適用於 AWS CLI 第 2 版。
> AWS CLI 第 1 版處於維護模式且即將終止支援，且第 2 版中引進的功能不會回溯到第 1 版。若要將它們與 搭配使用 AWS CLI，請遷移至AWS CLI 版本 2。

```
aws iotwireless get-position-estimate locationout.json \
    --ip IpAddress="54.240.198.35" \
    --wi-fi-access-points \
        MacAddress="A0:EC:F9:1E:32:C1",Rss=-75 \
        MacAddress="A0:EC:F9:15:72:5E",Rss=-67 \
    --advanced-configuration \
        WiFiCellular={ConfidencePercent=68}
```

此範例包含 Wi-Fi 存取點和 IP 地址做為測量類型。 AWS IoT Core 裝置位置會在 Wi-Fi 求解器和 IP 反向查詢求解器之間進行選擇，並選取準確度更高的求解器。

選用--advanced-configuration參數包含 WiFiCellular 物件，其中包含 ConfidencePercent 欄位，可自訂 Wi-Fi 和行動測量資料的可信度等級。在此範例中，

ConfidencePercent設定為 *68*，這也是未指定此參數時的預設值。如需詳細資訊，請參閱AWS IoT Core 裝置位置的進階組態。

系統會將解析的位置傳回為採用 GeoJSON 格式的承載，此格式可用於對地理資料結構進行編碼。然後它儲存在 *locationout.json* 檔案中。承載包含 WGS84 緯度和經度座標、準確度和可信度資訊、位置資料類型，以及解析位置的時間戳記。

```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "timestamp": "2026-04-07T01:16:39.971767716Z"
    }
}
```

對解析位置時發生的錯誤進行疑難排解

當您嘗試解析位置時，您可能會看到以下任何錯誤代碼。使用 GetPositionEstimate API 操作時，AWS IoT Core 裝置位置可能會產生錯誤，或者參考與 AWS IoT 主控台中錯誤對應的行號。

- 400 - 錯誤

  此錯誤表示裝置位置無法驗證 AWS IoT Core 裝置承載 JSON 的格式。發生錯誤的原因可能是：

  - JSON 測量資料的格式不正確。
  - 裝載只包含時間戳記資訊。
  - 測量資料參數 (例如 IP 地址) 無效。

  若要解決此錯誤，請檢查您的 JSON 格式是否正確，並包含來自一或多種測量類型的資料作為輸入項。如果 IP 地址無效，請參閱 IP 反向查詢求解器 以深入了解如何提供有效 IP 地址以解決錯誤。

- 403 - 錯誤

此錯誤表示您沒有執行 API 操作或使用 AWS IoT 主控台擷取裝置位置的許可。若要解決此錯誤，請確認您具有執行此動作所需的許可。如果您的 AWS 管理主控台 工作階段或 AWS CLI 工作階段字符已過期，可能會發生此錯誤。若要解決此錯誤，請重新整理工作階段字符以使用 AWS CLI，或登出 AWS 管理主控台 ，然後使用您的登入資料登入。

- 404 - 錯誤

  此錯誤表示 AWS IoT Core 裝置位置找不到或解決任何位置資訊。由於輸入的測量資料中可能有資料不足等情況，因此可能會發生錯誤。例如：

  - MAC 地址或行動網路基地台的資訊不夠充分。
  - 無法使用 IP 地址查詢和擷取位置。
  - GNSS 承載不足。
  - 多影格 GNSS 擷取不夠。

  若要解決這種情況下的錯誤，請檢查您的測量資料是否包含解析裝置位置所需的足夠資訊。

- 500 - 錯誤

  此錯誤表示 AWS IoT Core 裝置位置功能嘗試解析位置時，內部伺服器發生異常狀況。若要嘗試修正此錯誤，請重新整理工作階段，然後重新嘗試傳送要解析的測量資料。

使用裝置位置 MQTT 主題解決 AWS IoT Core 裝置位置

您可以使用預留 MQTT 主題，透過裝置位置功能取得裝置的最新 AWS IoT Core 位置資訊。

裝置位置 MQTT 主題的格式

AWS IoT Core 裝置位置的預留主題使用以下字首：

`$aws/device_location/{customer_device_id}/`

若要建立完整的主題，請先將 *customer_device_id* 取代為您用來識別裝置的唯一 ID。如果您的裝置已註冊為 AWS IoT 物件`WirelessDeviceId`，建議您指定 *thingName*，例如 for LoRaWAN 和 Sidewalk 裝置，以及 。然後，您可以將主題附加到主題 Stub，例如 `get_position_estimate` 或 `get_position_estimate/accepted`，如下節所示。

> **Note**
>
> *{customer_device_id}* 只能包含英文字母、數字和破折號。訂閱裝置位置主題時，您只能使用加號 (+) 作為萬用字元。例如，您可以針對 *{customer_device_id}* 使用 + 萬用字元，以取得您裝置的位置資訊。當您訂閱主題 `$aws/device_location/+/get_position_estimate/accepted` 時，如果已成功解析，則會發佈一則訊息，其中包含符合任何裝置 ID 之裝置的位置資訊。

以下是用於與 AWS IoT Core 裝置位置互動的預留主題。

裝置位置 MQTT 主題

主題	允許操作	說明
$aws/device_location/*customer_device_id*/get_position_estimate	發布	裝置會發佈至此主題，以取得要由 AWS IoT Core 裝置位置解析的掃描原始測量資料。
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	訂閱	AWS IoT Core 裝置位置成功解析裝置位置時，會將位置資訊發佈至此主題。
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	訂閱	AWS IoT Core 裝置位置無法解析裝置位置時，會將錯誤資訊發佈至此主題。

裝置位置 MQTT 主題的政策

若要從裝置位置主題接收訊息，您的裝置必須使用允許它連線到 AWS IoT 裝置閘道並訂閱 MQTT 主題的政策。

以下為接收各種主題訊息所需的政策範例。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/rejected"
            ]
        }
    ]
}
```

裝置位置主題和承載

以下顯示 AWS IoT Core 裝置位置主題、其訊息承載的格式，以及每個主題的範例政策。

主題

- /get_position_estimate
- /get_position_estimate/accepted
- /get_position_estimate/rejected

/get_position_estimate

發佈訊息至此主題，從裝置取得原始測量資料，以供 AWS IoT Core 裝置位置解析。

```
$aws/device_location/customer_device_id/get_position_estimate
```

AWS IoT Core 裝置位置會透過發佈至 /get_position_estimate/accepted或 來回應 /get_position_estimate/rejected。

> **Note**
>
> 發佈至此主題的訊息必須是有效的 JSON 承載。如果輸入訊息不是有效的 JSON 格式，您將不會收到任何回應。如需詳細資訊，請參閱訊息承載。

訊息承載

訊息承載格式遵循與 AWS IoT Wireless API 操作請求內文 類似的結構GetPositionEstimate。它包含以下內容：

- 選用的 Timestamp 字串，其對應至位置的解析日期和時間。Timestamp 字串的長度最短可以是 1，最長可以是 10。
- 選用的 MessageId 字串，其可以用來將請求對應至回應。如果您指定此字串，發佈至 get_position_estimate/accepted 或 get_position_estimate/rejected 主題的訊息將會包含此 MessageId。MessageID 字串的長度最短可以是 1，最長可以是 256。
- 來自裝置的測量資料，其中包含下列一或多種測量類型：
  - WiFiAccessPoint
  - CellTowers
  - IpAddress
  - Gnss

- GnssMultiFrame

以下顯示範例訊息承載。

```
{
    "Timestamp": "1664313161",
    "MessageId": "ABCD1",
    "WiFiAccessPoints":  [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -66
        }
    ],
    "Ip":{
        "IpAddress": "54.192.168.0"
    },
    "Gnss":{
      "Payload":"8295A614A2029517F4F77C0A7823B161A6FC57E25183D96535E3689783F6CA48",
      "CaptureTime":1354393948
    }
}
```

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
            ]
        }
    ]
}
```

/get_position_estimate/accepted

AWS IoT Core 裝置位置會在傳回裝置的已解析位置資訊時，發佈對此主題的回應。位置資訊會以 GeoJSON 格式傳回。

```
$aws/device_location/customer_device_id/get_position_estimate/accepted
```

以下顯示訊息承載和範例政策。

訊息承載

以下是 GeoJSON 格式的訊息承載範例。如果您在原始測量資料`MessageId`中指定了 ，且 AWS IoT Core Device Location 成功解析了位置資訊，則訊息承載會傳回相同的`MessageId`資訊。

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z",
        "messageId": "ABCD1"
    }
}
```

範例 政策

以下為所需政策的範例：

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted"
            ]
        }
    ]
}
```

/get_position_estimate/rejected

AWS IoT Core 當裝置位置無法解析裝置位置時，會發佈錯誤回應至此主題。

```
$aws/device_location/customer_device_id/get_position_estimate/rejected
```

以下顯示訊息承載和範例政策。如需錯誤的相關資訊，請參閱 對解析位置時發生的錯誤進行疑難排解。

訊息承載

以下是提供錯誤碼和訊息的訊息承載範例，指出 AWS IoT Core 裝置位置無法解析位置資訊的原因。如果您在提供原始測量資料`MessageId`時指定 ，且 AWS IoT Core 裝置位置無法解析位置資訊，則訊息承載中會傳回相同的`MessageId`資訊。

```
{
    "errorCode": 500,
```

```
    "errorMessage":"Internal server error",
    "messageId": "ABCD1"
}
```

範例 政策

以下為所需政策的範例：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/device_location/
device-123/get_position_estimate/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/device-123/
get_position_estimate/rejected"
      ]
    }
  ]
}
```

位置求解器和裝置承載

位置求解器是可用於解析 IoT 裝置位置的演算法。 AWS IoT Core Device Location 支援下列位置求解器。您會看到這些測量類型的 JSON 承載資料格式範例、求解器支援的裝置，以及解析位置的方式。

若要解析裝置位置，請指定一或多個測量資料類型。系統會綜合所有測量資料傳回單一的解析位置。

主題

- 基於 Wi-Fi 的求解器
- 行動網路求解器
- IP 反向查詢求解器
- 求解器
- 多影格 GNSS 求解器
- Amazon Sidewalk 啟用裝置的 BLE 求解器

基於 Wi-Fi 的求解器

使用 Wi-Fi 解析器依據來自 Wi-Fi 存取點的掃描資訊來解析位置。求解器支援 WLAN 技術，可用於計算透過 TCP/IP、LoRaWAN 無線裝置和啟用 Amazon Sidewalk 的裝置連線之一般 IoT 裝置的裝置位置。

LoRaWAN 裝置必須具有可對傳入 Wi-Fi 掃描資訊進行解碼的 LoRaWAN 晶片組。LoRa Edge 是一個超低功耗地理位置平台，以地理位置應用為目標，整合了長距離 LoRa 收發器、多星系 GNSS 掃描器和被動式 Wi-Fi MAC 掃描器。從裝置收到上行訊息時，Wi-Fi 掃描資料會傳送到 AWS IoT Core 裝置位置，並根據 Wi-Fi 掃描結果估計位置。解碼後的資訊會傳遞到 Wi-Fi 求解器以擷取位置資訊。

若要進一步了解如何解決已啟用 Amazon Sidewalk 裝置的位置資料，請造訪AWS IoT Core for Amazon Sidewalk 開發人員指南。

Wi-Fi 求解器承載範例

下列程式碼範例顯示了含有測量資料的裝置所提供的 JSON 承載。當 AWS IoT Core Device Location 收到此資料做為輸入時，它會傳送 HTTP 請求給求解器提供者，以解析位置資訊。若要擷取資訊，請指定 MAC 地址和 RSS (接收訊號強度) 的值。若要執行此步驟，請使用此格式提供 JSON 承載，或使用 GetPositionEstimate API 操作的 WiFiAccessPoints object 參數。

```
{
    "Timestamp": 1664313161,    // optional
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",  // required
            "Rss": -75                          // required
        }
    ]
```

```
}
```

行動網路求解器

您可以使用行動網路求解器使用從行動網路無線電塔取得的測量資料來解析位置。求解器支援下列技術。即使您納入任何或所有上述技術的測量資料，也只會得出單筆解析位置資訊。

- GSM
- CDMA
- WCDMA
- TD-SCDMA
- LTE

行動網路求解器承載範例

下列程式碼範例顯示了含有行動網路測量資料的裝置所提供的 JSON 承載。當 AWS IoT Core Device Location 收到此資料做為輸入時，它會傳送 HTTP 請求給求解器提供者，以解析位置資訊。若要擷取資訊，您可以在主控台中使用此格式提供 JSON 承載，或指定 GetPositionEstimate API 操作的 CellTowers 參數值。您可以使用上述任何或所有行動網路技術來指定參數值，藉以提供測量資料。

LTE (長期演進技術)

使用此測量資料時，您必須指定特定資訊，例如網路或行動網路的國家/地區代碼及其他選擇性參數，包括有關本機 ID 的資訊。下列程式碼顯示承載格式的範例。如需這些參數的詳細資訊，請參閱 LTE 物件。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Lte": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "EutranCid": int,             // required. Make sure that you use int for
 EutranCid.
          "Tac": int,                   // optional
          "LteLocalId": {               // optional
              "Pci": int,               // required
              "Earfcn": int,            // required
```

```
          },
          "LteTimingAdvance": int,      // optional
          "Rsrp": int,                  // optional
          "Rsrq": float,                // optional
          "NrCapable": boolean,         // optional
          "LteNmr": [                   // optional
                {
                    "Pci": int,         // required
                    "Earfcn": int,      // required
                    "EutranCid": int,   // required
                    "Rsrp": int,        // optional
                    "Rsrq": float       // optional
                }
            ]
         }
      ]
   }
}
```

GSM（全球行動通訊系統）

使用此測量資料時，您必須指定特定資訊，例如網路或行動網路的國家/地區代碼、基地台資訊及其他選擇性參數。下列程式碼顯示承載格式的範例。如需這些參數的詳細資訊，請參閱 GSM 物件。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Gsm": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "Lac": int,                   // required
          "GeranCid": int,              // required
          "GsmLocalId": {               // optional
              "Bsic": int,              // required
              "Bcch": int,              // required
          },
          "GsmTimingAdvance": int,      // optional
          "RxLevel": int,               // optional
          "GsmNmr": [                   // optional
            {
                "Bsic": int,            // required
                "Bcch": int,            // required
```

```
                "RxLevel": int,          // optional
                "GlobalIdentity": {
                    "Lac": int,          // required
                    "GeranCid": int      // required
                }
            }
        ]
    }
]
}
```

CDMA (分碼多重進接)

使用此測量資料時，您必須指定特定資訊，例如訊號功率和識別資訊、基地台資訊及其他選擇性參數。下列程式碼顯示承載格式的範例。如需這些參數的詳細資訊，請參閱 CDMA 物件。

```
{
    "Timestamp": 1664313161,                // optional
    "CellTowers": {
        "Cdma": [
        {
            "SystemId": int,                // required
            "NetworkId": int,               // required
            "BaseStationId": int,           // required
            "RegistrationZone": int,        // optional
            "CdmaLocalId": {                // optional
              "PnOffset": int,              // required
              "CdmaChannel": int,           // required
            },
            "PilotPower": int,              // optional
            "BaseLat": float,               // optional
            "BaseLng": float,               // optional
            "CdmaNmr": [                    // optional
                {
                    "PnOffset": int,        // required
                    "CdmaChannel": int,     // required
                    "PilotPower": int,      // optional
                    "BaseStationId": int    // optional
                }
             ]
        }
    ]
}
```

```
}
```

WCDMA (寬頻分碼多重進接)

使用此測量資料時，您必須指定特定資訊，例如網路與國家/地區代碼、訊號功率和識別資訊、基地台資訊及其他選擇性參數。下列程式碼顯示承載格式的範例。如需這些參數的詳細資訊，請參閱 CDMA 物件。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Wcdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "WcdmaLocalId": {             // optional
              "Uarfcndl": int,          // required
              "Psc": int,               // required
          },
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "WcdmaNmr": [                 // optional
                {
                  "Uarfcndl": int,      // required
                  "Psc": int,           // required
                  "UtranCid": int,      // required
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
             ]
          }
       ]
    }
}
```

TD-SCDMA (分時-同步分碼多重進接)

使用此測量資料時，您必須指定特定資訊，例如網路與國家/地區代碼、訊號功率和識別資訊、基地台資訊及其他選擇性參數。下列程式碼顯示承載格式的範例。如需這些參數的詳細資訊，請參閱 CDMA 物件。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Tdscdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "TdscdmaLocalId": {           // optional
              "Uarfcn": int,            // required
              "CellParams": int,        // required
          },
          "TdscdmaTimingAdvance": int,  // optional
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "TdscdmaNmr": [               // optional
                {
                  "Uarfcn": int,        // required
                  "CellParams": int,    // required
                  "UtranCid": int,      // optional
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
             ]
         }
      ]
   }
}
```

IP 反向查詢求解器

您可以使用 IP 反向查詢求解器，以 IP 地址作為輸入項來解析位置。求解器可以從已佈建的裝置取得位置資訊 AWS IoT。使用 IPv4 或 IPv6 標準模式格式，或 IPv6 十六進位壓縮模式來指定 IP 地址資訊。然後，您將取得解析位置估計值，包括裝置所在城市和國家/地區等額外資訊。

> **Note**
>
> 使用 IP 反向查詢，即表示您同意不將其用於識別或定位特定家庭或街道地址。

IP 反向查詢求解器承載範例

下列程式碼範例顯示了含有測量資料的裝置所提供的 JSON 承載。當 AWS IoT Core Device Location 在測量資料中收到 IP 地址資訊時，它會在求解器提供者的資料庫中查詢此資訊，然後用於解析位置資訊。若要擷取資訊，請使用此格式提供 JSON 承載，或指定 GetPositionEstimate API 操作的 Ip 參數值。

> **Note**
>
> 使用此求解器時，除了座標之外，還會報告裝置所在的城市、州、國家/地區和郵遞區號。如需範例，請參閱 解析裝置位置 (主控台)。

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress":"54.240.198.35"
      }
}
```

求解器

使用 GNSS (全球導航衛星系統) 求解器，依據 GNSS 掃描結果訊息或 NAV 訊息包含的資訊來擷取裝置位置。您可以選擇性地提供額外的 GNSS 輔助資訊，以減少求解器搜尋訊號時必須使用的變數數目。藉由提供此輔助資訊 (包括位置、高度以及擷取時間和準確度資訊)，求解器可以輕鬆識別視圖中的衛星並計算裝置位置。

此求解器可與 LoRaWAN 裝置和啟用 Amazon Sidewalk 的裝置，以及其他已佈建 IoT AWS IoT 裝置的裝置搭配使用 AWS IoT。對於一般 IoT 裝置，如果裝置支援使用 GNSS 進行位置估算，則收發器會在從裝置接收到 GNSS 掃描資訊後解析位置資訊。對於 LoRaWAN 裝置，這類裝置必須具備 LoRa 邊緣晶片組。從裝置接收上行訊息時，GNSS 掃描資料會傳送至 AWS IoT for LoRaWAN 或 AWS IoT Amazon Sidewalk，並根據收發器的掃描結果估計位置。

> **Note**
>
> `Gnss` 和 `GnssMultiFrame`測量類型是互斥的。單一請求不能同時包含兩者。

GNSS 求解器承載範例

下列程式碼範例顯示了含有測量資料的裝置所提供的 JSON 承載。當 AWS IoT Core 裝置位置收到包含測量資料中承載的 GNSS 掃描資訊時，它會使用收發器和包含的任何其他協助資訊來搜尋訊號並解析位置資訊。若要擷取資訊，請使用此格式提供 JSON 承載，或指定 GetPositionEstimate API 操作的 Gnss 參數值。

> **Note**
>
> 在 AWS IoT Core 裝置位置可以解析裝置位置之前，您必須從承載中移除目的地位元組。

```
{
    "Timestamp": 1664313161,                    // optional
    "Gnss": {
        "AssistAltitude": number,               // optional
        "AssistPosition": [ number ],           // optional
        "CaptureTime": number,                  // optional
        "CaptureTimeAccuracy": number,          // optional
        "Payload": "string",                    // required
        "Use2DSolver": boolean                  // optional
    }
}
```

多影格 GNSS 求解器

使用多影格 GNSS（全域導航衛星系統）求解器，使用相同裝置擷取的多個 GNSS 掃描結果訊息或 NAV 訊息中包含的資訊來擷取裝置位置。您可以在`Captures`清單中提供擷取。您可以選擇性地提供額外的 GNSS 輔助資訊，以減少求解器搜尋訊號時必須使用的變數數目。藉由提供此輔助資訊 (包括位置、高度以及擷取時間和準確度資訊)，求解器可以輕鬆識別視圖中的衛星並計算裝置位置。

> **Note**
>
> `Gnss` 和 `GnssMultiFrame`測量類型是互斥的。單一請求不能同時包含兩者。

多影格 GNSS 求解器承載範例

下列程式碼範例顯示了含有測量資料的裝置所提供的 JSON 承載。當 AWS IoT Core Device Location 收到包含測量資料中擷取的多影格 GNSS 掃描資訊時，它會使用收發器和包含的任何其

他協助資訊來搜尋訊號並解析位置資訊。若要擷取資訊，請使用此格式提供 JSON 承載，或指定 GetPositionEstimate API 操作的 GnssMultiFrame 參數值。

> **Note**
>
> 1. 在 AWS IoT Core 裝置位置可以解析裝置位置之前，您必須從每個擷取承載中移除目的地位元組。
> 2. `Captures` 清單中提供的擷取數目必須是下列其中一個值：2、4、8、16 或 32。
> 3. 當請求`Use2DSolver`設為 時`true`，您還必須提供 `AssistAltitude`。否則，請求會失敗並出現驗證錯誤。

```
{
    "Timestamp": 1664313161,                    // optional
    "GnssMultiFrame": {
        "AssistAltitude": number,               // optional
        "AssistPosition": [ number ],           // optional
        "CaptureTimeAccuracy": number,          // optional
        "Use2DSolver": boolean,                 // optional
        "Captures": [                           // required, must be 2, 4, 8, 16, or 32
 captures
            {
                "Payload": "string",            // required
                "CaptureTime": number           // optional
            },
            {
                "Payload": "string",            // required
                "CaptureTime": number           // optional
            },
                        ...
            {
                "Payload": "string",            // required
                "CaptureTime": number           // optional
            }
        ]
   }
}
```

Amazon Sidewalk 啟用裝置的 BLE 求解器

> **Note**
>
> 對於以低功耗藍牙為基礎的位置， 會根據連線至 Amazon Sidewalk 且啟用社群調查結果功能的附近 Sidewalk 閘道的大致位置 AWS IoT 傳回位置座標。閘道位置資料是 AWS 內容，僅為了協助您定位連線至 Amazon Sidewalk 的裝置而提供給您，而且您只能將資料用於該用途。您只能透過我們一般提供給您的界面和功能來使用和存取位置資料，而且您不得嘗試重新識別、反向工程或重新映射我們提供給您的任何閘道位置資料。

透過 連線的 Amazon Sidewalk 終端裝置 AWS IoT Core for Amazon Sidewalk 可以使用 Amazon Sidewalk 終端裝置發佈的 BLE、Wi-Fi 或 GNSS 上行訊息來解析其位置資料。如需詳細資訊，請參閱 AWS IoT Core for Amazon Sidewalk 開發人員指南。

AWS IoT Core 裝置位置的進階組態

您可以使用 `AdvancedConfiguration` 參數來自訂 AWS IoT Core Device Location 如何解析裝置的位置。使用 GetPositionEstimate API 操作、 AWS IoT 主控台或裝置位置 MQTT 主題時，支援此參數。

> **Note**
>
> `AdvancedConfiguration` 參數是選用的，目前支援 HTTP API 流程和 MQTT 裝置位置流程。LoRaWAN 裝置位置流程不支援此功能。
> 未使用此參數的現有整合不會受到影響，並繼續像以前一樣運作。未指定 參數時，會使用預設可信度層級 68%，這是引入此參數之前的預設行為。

主題

- WiFiCellular 組態
- 使用 AdvancedConfiguration 的 Wi-Fi 承載範例
- 使用 AdvancedConfiguration 的行動承載範例

WiFiCellular 組態

當您的測量資料包含 Wi-Fi、行動網路或兩者承載資訊時，就會套用此`WiFiCellular`組態。使用此組態來自訂 Wi-Fi 和行動網路求解器的可信度等級。

`ConfidencePercent`

使用 Wi-Fi 或行動測量資料解析裝置位置時使用的可信度百分比。此值決定實際裝置位置落在報告的準確性半徑內的機率。指定介於 50 到 99 之間的整數值。

較高的可信度百分比會產生較大的準確度半徑，而較低的可信度百分比則會產生較小的準確度半徑。未指定此參數時，預設可信度層級為 68%。

> **Note**
>
> 只有當測量資料包含 Wi-Fi 或行動網路資訊時，才會使用此`WiFiCellular`組態。如果測量資料僅包含其他類型，例如 IP 地址或 GNSS，則會忽略`WiFiCellular`組態。

使用 AdvancedConfiguration 的 Wi-Fi 承載範例

下列範例顯示 Wi-Fi 測量承載，其中包含將可信度層級設定為 95% 的 `AdvancedConfiguration` 參數。

```
{
    "Timestamp": 1664313161,
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -75
        },
        {
            "MacAddress": "A0:EC:F9:15:72:5E",
            "Rss": -67
        }
    ],
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 95
        }
    }
```

```
}
```

下列 GeoJSON 承載顯示已解析的位置，其可信度等級設定為 0.95，符合請求的 95% 可信度。

```
{
    "coordinates": [
        23.864079,
        61.44683
    ],
    "type": "Point",
    "properties": {
        "horizontalAccuracy": 22,
        "horizontalConfidenceLevel": 0.95,
        "timestamp": "2026-04-03T00:13:15.134343033Z"
    }
}
```

使用 AdvancedConfiguration 的行動承載範例

下列範例顯示行動測量承載，其中包含將可信度層級設定為 55% 的 AdvancedConfiguration 參數。

```
{
    "Timestamp": 1664313161,
    "CellTowers": {
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
```

```
                    "GeranCid": 1
                }
            }]
        }],
        "Wcdma": [{
            "Mcc": 262,
            "Mnc": 7,
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
            "Rsrp": -50,
            "Rsrq": -5,
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    },
```

```
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 55
        }
    }
}
```

下列 GeoJSON 承載顯示已解析的位置，其可信度等級設定為 0.55，符合請求的 55% 可信度。

```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 34,
        "verticalConfidenceLevel": 0.55,
        "horizontalAccuracy": 254,
        "horizontalConfidenceLevel": 0.55,
        "timestamp": "2026-04-06T23:43:45.198088456Z"
    }
}
```

事件訊息

本節包含更新或變更物件或任務 AWS IoT 時， 發佈的訊息相關資訊。

如何產生事件訊息

AWS IoT 會在發生特定事件時發佈事件訊息。例如，在新增、更新或刪除事物時，由登錄檔產生的事件。每個事件都會觸發傳送一則事件訊息。事件訊息會透過具 JSON 承載的 MQTT 發佈。承載內容取決於事件的類型。

> **Note**
>
> 事件訊息保證會發佈一次。事件訊息也有可能發佈超過一次。事件訊息的順序無法保證。

接收事件訊息的政策

若要接收事件訊息，您的裝置必須使用適當的政策，允許它連線到 AWS IoT 裝置閘道並訂閱 MQTT 事件主題。您也必須訂閱合適的主題篩選條件。

以下為接收生命週期事件所需的政策範例：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:/$aws/events/*"
            ]
        }
    ]
}
```

啟用 的事件 AWS IoT

在預留主題的訂閱者可以接收訊息之前，您必須從 或使用 API AWS 管理主控台 或 CLI 啟用事件訊息。如需不同選項管理的事件訊息資訊，請參閱AWS IoT 事件組態設定表。

- 若要啟用事件訊息，請前往 AWS IoT 主控台的設定索引標籤，然後在事件型訊息區段中選擇管理事件。您可以指定要管理的事件。
- 若要透過使用 API 或 CLI 來控制要發佈的事件類型，請呼叫 UpdateEventConfigurations API 或使用 update-event-configurations CLI 命令。例如：

```
aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\":
 true}}"
```

> **Note**
>
> 所有引號 (") 都會與反斜線 (\) 一起逸出。

您可以透過呼叫 DescribeEventConfigurations API 或使用 describe-event-configurations CLI 命令來取得目前的事件設定。例如：

```
aws iot describe-event-configurations
```

AWS IoT 事件組態設定資料表

事件類別 (AWS IoT 主控台：設定：事件型訊息)	**eventConfigurations** 金鑰值 (AWS CLI/API)	事件訊息主題
(只能使用 AWS CLI/API 進行設定)	`CA_CERTIFICATE`	`$aws/events/certificates/registered/`*`caCertificateId`*

事件類別 (AWS IoT 主控台：設定：事件型訊息)	**eventConfigurations** 金鑰值 (AWS CLI/API)	事件訊息主題
(只能使用 AWS CLI/API 進行設定)	CERTIFICATE	$aws/events/presence/connected/ *clientId*
(只能使用 AWS CLI/API 進行設定)	CERTIFICATE	$aws/events/presence/disconnected/ *clientId*
(只能使用 AWS CLI/API 進行設定)	CERTIFICATE	$aws/events/subscriptions/subscribed/ *clientId*
(只能使用 AWS CLI/API 進行設定)	CERTIFICATE	$aws/events/subscriptions/unsubscribed/ *clientId*
任務完成，取消	JOB	$aws/events/job/*jobID*/canceled
任務完成，取消	JOB	$aws/events/job/*jobID*/cancellation_in_progress
任務完成，取消	JOB	$aws/events/job/*jobID*/completed
任務完成，取消	JOB	$aws/events/job/*jobID*/deleted
任務完成，取消	JOB	$aws/events/job/*jobID*/deletion_in_progress

事件類別 (AWS IoT 主控台：設定：事件型訊息)	**eventConfigurations** 金鑰值 (AWS CLI/API)	事件訊息主題
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/canceled
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/deleted
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/failed
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/rejected
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/removed
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/succeeded
任務執行：成功、失敗、拒絕、取消、移除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/timed_out
物件：建立，更新，刪除	THING	$aws/events/thing/ *thingName* /created
物件：建立，更新，刪除	THING	$aws/events/thing/ *thingName* /updated
物件：建立，更新，刪除	THING	$aws/events/thing/ *thingName* /deleted

事件類別 (AWS IoT 主控台：設定：事件型訊息)	**eventConfigurations** 金鑰值 (AWS CLI/API)	事件訊息主題
物件群組：新增、移除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /created
物件群組：新增、移除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /updated
物件群組：新增、移除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /deleted
物件群組階層：新增、移除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /added
物件群組階層：新增、移除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /removed
物件群組成員資格：新增、移除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /added

事件類別 (AWS IoT 主控台：設定：事件型訊息)	**eventConfigurations** 金鑰值 (AWS CLI/API)	事件訊息主題
物件群組成員資格：新增、移除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /removed
物件類型：建立、更新、刪除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /created
物件類型：建立、更新、刪除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /updated
物件類型：建立、更新、刪除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /deleted
物件類型關聯：新增、移除	THING_TYPE_ASSOCIATION	$aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /added $aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /removed

登錄檔事件

該登錄檔可能會在建立、更新或刪除物件、物件類型和物件群組時發佈事件訊息。不過，這些事件預設為無法使用。如需如何開啟這些事件的詳細資訊，請參閱 啟用 的事件 AWS IoT。

此登錄檔可以提供下列事件類型：

- 物件事件
- 物件類型事件
- 物件群組事件

物件事件

建立/更新/刪除的物件

該登錄檔會在建立、更新或刪除事物時發佈以下事件訊息：

- $aws/events/thing/*thingName*/created
- $aws/events/thing/*thingName*/updated
- $aws/events/thing/*thingName*/deleted

訊息包含的承載範例如下：

```
{
    "eventType" : "THING_EVENT",
    "eventId" : "f5ae9b94-8b8e-4d8e-8c8f-b3266dd89853",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName" : "MyThing",
    "versionNumber" : 1,
    "thingTypeName" : null,
    "attributes": {
                "attribute3": "value3",
                "attribute1": "value1",
                "attribute2": "value2"
    }
}
```

承載包含以下屬性：

eventType

設定為「THING_EVENT」。

eventId

獨特的事件 ID (字串)。

timestamp

事件發生時的 UNIX 時間戳記。

operation

觸發事件的操作。有效的值如下：

- CREATED (已建立)
- UPDATED (已更新)
- DELETED

accountId

您的 AWS 帳戶 ID。

thingId

建立、更新或刪除的物件之 ID。

thingName

建立、更新或刪除的物件之名稱。

versionNumber

建立、更新或刪除的物件之版本。物件建立時，此值設定為 1。物件更新時，此值會增加 1。

thingTypeName

與物件關聯的物件類型 (若存在)。否則為 `null`。

屬性

與物件關聯的名稱/值對之集合。

物件類型事件

物件類型相關事件：

- 物件類型Created/Updated/Deprecated/Undeprecated/Deleted
- 關聯事物類型或取消關聯事物

物件類型Created/Updated/Deprecated/Undeprecated/Deleted

當物件類型建立、更新、棄用、取消棄用或刪除時，登錄檔會發佈下列事件訊息：

- $aws/events/thingType/*thingTypeName*/created
- $aws/events/thingType/*thingTypeName*/updated
- $aws/events/thingType/*thingTypeName*/deleted

訊息包含的承載範例如下：

```
{
    "eventType" : "THING_TYPE_EVENT",
    "eventId" : "8827376c-4b05-49a3-9b3b-733729df7ed5",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingTypeId" : "c530ae83-32aa-4592-94d3-da29879d1aac",
    "thingTypeName" : "MyThingType",
    "isDeprecated" : false|true,
    "deprecationDate" : null,
    "searchableAttributes" : [ "attribute1", "attribute2", "attribute3" ],
    "propagatingAttributes": [
        {
     "userPropertyKey": "key",
     "thingAttribute": "model"
        },
        {
     "userPropertyKey": "key",
     "connectionAttribute": "iot:ClientId"
        }
    ],
    "description" : "My thing type"
}
```

承載包含以下屬性：

eventType

設定為「THING_TYPE_EVENT」。

eventId

獨特的事件 ID (字串)。

timestamp

事件發生時的 UNIX 時間戳記。

operation

觸發事件的操作。有效的值如下：

- CREATED (已建立)
- UPDATED (已更新)
- DELETED

accountId

您的 AWS 帳戶 ID。

thingTypeId

正在建立、更新、棄用或刪除之物件類型的 ID。

thingTypeName

正在建立、更新、棄用或刪除的物件類型名稱。

isDeprecated

如果指定物件類型已棄用，則為 `true`。否則為 `false`。

deprecationDate

棄用物件類型的 UNIX 時間戳記。

searchableAttributes

與可用於搜尋的物件類型關聯之名稱/值對集合。

propagatingAttributes

傳播屬性的清單。傳播屬性可以包含物件屬性、連線屬性和使用者屬性金鑰。如需詳細資訊，請參閱新增訊息擴充的傳播屬性。

description

物件類型描述。

關聯事物類型或取消關聯事物

當事物類型與事物關聯或取消關聯時，登錄檔會發佈以下事件訊息。

- `$aws/events/thingTypeAssociation/thing/thingName/thingType/typeName/added`
- `$aws/events/thingTypeAssociation/thing/thingName/thingType/typeName/removed`

以下是 `added` 承載的範例。`removed` 訊息的承載類似。

```
{
    "eventId" : "87f8e095-531c-47b3-aab5-5171364d138d",
    "eventType" : "THING_TYPE_ASSOCIATION_EVENT",
    "operation" : "ADDED",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName": "myThing",
    "thingTypeName" : "MyThingType",
    "timestamp" : 1234567890123,
}
```

承載包含以下屬性：

eventId

獨特的事件 ID (字串)。

eventType

設定為 "THING_TYPE_ASSOCIATION_EVENT"。

operation

觸發事件的操作。有效的值如下：

- ADDED
- REMOVED

thingId

類型關聯已變更的物件 ID。

thingName

類型關聯已變更的物件名稱。

thingTypeName

與物件相關聯或不再相關聯的物件類型。

timestamp

事件發生時的 UNIX 時間戳記。

物件群組事件

物件群組相關事件：

- 建立/更新/刪除物件群組
- 從事物群組新增或移除的事物
- 從事物群組新增或移除的事物群組

建立/更新/刪除物件群組

登錄檔會在建立、更新或刪除事物群組時發佈以下事件訊息。

- `$aws/events/thingGroup/groupName/created`
- `$aws/events/thingGroup/groupName/updated`
- `$aws/events/thingGroup/groupName/deleted`

以下是 `updated` 承載的範例。`created` 和 `deleted` 訊息的承載類似。

```
{
  "eventType": "THING_GROUP_EVENT",
  "eventId": "8b9ea8626aeaa1e42100f3f32b975899",
  "timestamp": 1603995417409,
  "operation": "UPDATED",
  "accountId": "571EXAMPLE833",
  "thingGroupId": "8757eec8-bb37-4cca-a6fa-403b003d139f",
```

```
  "thingGroupName": "Tg_level5",
  "versionNumber": 3,
  "parentGroupName": "Tg_level4",
  "parentGroupId": "5fce366a-7875-4c0e-870b-79d8d1dce119",
  "description": "New description for Tg_level5",
  "rootToParentThingGroups": [
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/TgTopLevel",
      "groupId": "36aa0482-f80d-4e13-9bff-1c0a75c055f6"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level1",
      "groupId": "bc1643e1-5a85-4eac-b45a-92509cbe2a77"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level2",
      "groupId": "0476f3d2-9beb-48bb-ae2c-ea8bd6458158"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level3",
      "groupId": "1d9d4ffe-a6b0-48d6-9de6-2e54d1eae78f"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level4",
      "groupId": "5fce366a-7875-4c0e-870b-79d8d1dce119"
    }
  ],
  "attributes": {
    "attribute1": "value1",
    "attribute3": "value3",
    "attribute2": "value2"
  },
  "dynamicGroupMappingId": null
}
```

承載包含以下屬性：

eventType

設定為「THING_GROUP_EVENT」。

eventId

獨特的事件 ID (字串)。

timestamp

事件發生時的 UNIX 時間戳記。

operation

觸發事件的操作。有效的值如下：

- CREATED (已建立)
- UPDATED (已更新)
- DELETED

accountId

您的 AWS 帳戶 ID。

thingGroupId

建立、更新或刪除的物件群組之 ID。

thingGroupName

建立、更新或刪除的物件群組之名稱。

versionNumber

物件群組版本。物件群組建立時，此值設定為 1。物件群組更新時，此值會增加 1。

parentGroupName

父事物群組名稱 (若有的話)。

parentGroupId

父事物群組 ID (若有的話)。

description

物件群組描述。

rootToParentThingGroups

關於父物件群組的一系列資訊。每個父物件群組都有一個元素，從根物件群組開始，一直到物件群組的父項。每個項目都包含物件群組的 `groupArn` 和 `groupId`。

屬性

與物件群組關聯的名稱/值對之集合。

從事物群組新增或移除的事物

該登錄檔會在事物新增至事物群組，或從事物群組移除時發佈以下事件訊息。

- $aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/added
- $aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/removed

訊息包含的承載範例如下：

```
{
    "eventType" : "THING_GROUP_MEMBERSHIP_EVENT",
    "eventId" : "d684bd5f-6f6e-48e1-950c-766ac7f02fd1",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "groupArn" : "arn:aws:iot:ap-northeast-2:123456789012:thinggroup/
MyChildThingGroup",
    "groupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "thingArn" : "arn:aws:iot:ap-northeast-2:123456789012:thing/MyThing",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "membershipId" : "8505ebf8-4d32-4286-80e9-c23a4a16bbd8"
}
```

承載包含以下屬性：

eventType

設定為「THING_GROUP_MEMBERSHIP_EVENT」。

eventId

事件 ID。

timestamp

事件發生時的 UNIX 時間戳記。

operation

當物件新增到物件群組時，ADDED。當物件從物件群組移除時，REMOVED。

accountId

您的 AWS 帳戶 ID。

groupArn

物件群組的 ARN。

groupId

群組的 ID。

thingArn

物件群組新增或移除之物件的 ARN。

thingId

物件群組新增或移除之物件的 ID。

membershipId

代表物件和物件群組之間關係的 ID。當您在物件群組新增物件，就會產生此值。

從事物群組新增或移除的事物群組

當一個事物群組新增至另一個事物群組，或從另一個事物群組中移除事物群組時，登錄檔都會發佈以下事件訊息。

- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/added`
- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/removed`

訊息包含的承載範例如下：

```
{
    "eventType" : "THING_GROUP_HIERARCHY_EVENT",
    "eventId" : "264192c7-b573-46ef-ab7b-489fcd47da41",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "thingGroupId" : "8f82a106-6b1d-4331-8984-a84db5f6f8cb",
    "thingGroupName" : "MyRootThingGroup",
    "childGroupId" : "06838589-373f-4312-b1f2-53f2192291c4",
```

```
    "childGroupName" : "MyChildThingGroup"
}
```

承載包含以下屬性：

eventType

設定為「THING_GROUP_HIERARCHY_EVENT」。

eventId

事件 ID。

timestamp

事件發生時的 UNIX 時間戳記。

operation

當物件新增到物件群組時，ADDED。當物件從物件群組移除時，REMOVED。

accountId

您的 AWS 帳戶 ID。

thingGroupId

父物件群組的 ID。

thingGroupName

父物件群組的名稱。

childGroupId

子物件群組的 ID。

childGroupName

子物件群組的名稱。

任務事件

當 AWS IoT 任務待定、完成或取消，以及當裝置在執行任務時回報成功或失敗時，任務服務會發佈至 MQTT 通訊協定上的預留主題。藉由訂閱這些主題，裝置或管理及監控應用程式就能夠追蹤任務的狀態。

如何啟用任務事件

來自 AWS IoT Jobs 服務的回應訊息不會透過訊息代理程式傳遞，也無法由其他用戶端或規則訂閱。若要訂閱任務活動相關的訊息，請使用 notify 和 notify-next 主題。如需任務主題的詳細資訊，請參閱 任務主題。

若要收到任務更新的通知，請使用 AWS 管理主控台或使用 API 或 CLI 來啟用這些任務事件。如需詳細資訊，請參閱啟用 的事件 AWS IoT。

任務事件的運作方式

由於取消和刪除任務可能需要一些時間，系統會傳送兩個訊息來表示請求開始和結束。例如，當取消請求開始時，系統會將訊息傳送到 $aws/events/job/jobID/cancellation_in_progress 主題。當取消請求完成時，系統會將訊息傳送到 $aws/events/job/jobID/canceled 主題。

任務刪除請求的程序亦同。管理及監控應用程式可以藉由訂閱這些主題，以追蹤任務的狀態。如需發佈與訂閱 MQTT 主題的詳細資訊，請參閱 the section called “裝置通訊協定”。

任務事件類型

下列範例顯示不同類型的任務事件：

完成/取消/刪除的任務

當任務完成、取消、刪除或正在取消或刪除時， AWS IoT 任務服務會在 MQTT 主題上發佈訊息：

- $aws/events/job/*jobID*/completed
- $aws/events/job/*jobID*/canceled
- $aws/events/job/*jobID*/deleted
- $aws/events/job/*jobID*/cancellation_in_progress
- $aws/events/job/*jobID*/deletion_in_progress

completed 訊息包含的承載範例如下：

```
{
  "eventType": "JOB",
  "eventId": "7364ffd1-8b65-4824-85d5-6c14686c97c6",
  "timestamp": 1234567890,
  "operation": "completed",
  "jobId": "27450507-bf6f-4012-92af-bb8a1c8c4484",
  "status": "COMPLETED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/a39f6f91-70cf-4bd2-a381-9c66df1a80d0",
```

```
    "arn:aws:iot:us-east-1:123456789012:thinggroup/2fc4c0a4-6e45-4525-
a238-0fe8d3dd21bb"
  ],
  "description": "My Job Description",
  "completedAt": 1234567890123,
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123,
  "jobProcessDetails": {
    "numberOfCanceledThings": 0,
    "numberOfRejectedThings": 0,
    "numberOfFailedThings": 0,
    "numberOfRemovedThings": 0,
    "numberOfSucceededThings": 3
  }
}
```

canceled 訊息包含的承載範例如下。

```
{
  "eventType": "JOB",
  "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
  "timestamp": 1234567890,
  "operation": "canceled",
  "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
  "status": "CANCELED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
  ],
  "description": "My job description",
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123
}
```

deleted 訊息包含的承載範例如下。

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
```

```
      "operation": "deleted",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETED",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

cancellation_in_progress 訊息包含的承載範例如下：

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "cancellation_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "CANCELLATION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

deletion_in_progress 訊息包含的承載範例如下：

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
```

```
      "timestamp": 1234567890,
      "operation": "deletion_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

任務執行結束狀態

當裝置將任務執行更新為終端機狀態時，AWS IoT 任務服務會發佈訊息：

- $aws/events/jobExecution/*jobID*/succeeded
- $aws/events/jobExecution/*jobID*/failed
- $aws/events/jobExecution/*jobID*/rejected
- $aws/events/jobExecution/*jobID*/canceled
- $aws/events/jobExecution/*jobID*/timed_out
- $aws/events/jobExecution/*jobID*/removed
- $aws/events/jobExecution/*jobID*/deleted

訊息包含的承載範例如下：

```
{
  "eventType": "JOB_EXECUTION",
  "eventId": "cca89fa5-8a7f-4ced-8c20-5e653afb3572",
  "timestamp": 1234567890,
  "operation": "succeeded|failed|rejected|canceled|removed|timed_out",
  "jobId": "154b39e5-60b0-48a4-9b73-f6f8dd032d27",
  "thingArn": "arn:aws:iot:us-east-1:123456789012:myThing/6d639fbc-8f85-4a90-924d-
a2867f8366a7",
  "status": "SUCCEEDED|FAILED|REJECTED|CANCELED|REMOVED|TIMED_OUT",
  "statusDetails": {
    "key": "value"
```

```
  }
}
```

命令執行事件

AWS IoT 當命令執行變更狀態時， 會將事件訊息發佈至 MQTT 主題。您可以使用這些事件來監控命令執行進度，並建置可回應狀態變更的應用程式。

命令執行事件主題

命令執行事件會發佈至下列 MQTT 主題：

```
$aws/events/commandExecution/commandId/status
```

其中：

- *commandId* 是 命令的識別符。
- *status* 是命令執行的狀態。有效值為：CREATED、IN_PROGRESS、SUCCEEDED、FAILED、REJECTED、TIMED_OUT。

若要接收所有命令和所有狀態的通知，請使用萬用字元訂閱下列主題：

```
$aws/events/commandExecution/+/#
```

命令執行事件訊息

當命令執行狀態變更時， 會將事件訊息 AWS IoT 發佈至對應的 MQTT 主題。訊息包含的承載範例如下：

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status": "FAILED",
    "statusReason": {
        "reasonCode": "DEVICE_TOO_BUSY",
        "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
```

```
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp": 1717708862107
}
```

承載包含以下屬性：

executionId

命令執行的唯一識別符（字串）。

status

命令執行的狀態。有效值為：CREATED、IN_PROGRESS、SUCCEEDED、FAILED、REJECTED、TIMED_OUT。

statusReason

如果可用，則包含狀態額外資訊的物件。包含 reasonCode和 reasonDescription 欄位。

eventType

設定為 "COMMAND_EXECUTION"。

commandArn

命令的 Amazon Resource Name (ARN)。

targetArn

命令執行的目標裝置（物件或用戶端）ARN。

timestamp

事件發生時的 UNIX 時間戳記。

生命週期事件

AWS IoT 可以在 MQTT 主題上發佈生命週期事件。預設情況下，這些事件可用且無法停用。

> **Note**
>
> 生命週期的訊息可能不會按照順序傳送。您可能會收到重複的訊息。
> thingName 只有在用戶端使用獨家物件功能連線時，才會包含 。

在本主題中：

- 連線/中斷連線事件
- 連線嘗試失敗事件
- 訂閱/取消訂閱事件

連線/中斷連線事件

> **Note**
>
> 使用 AWS IoT Device Management 機群索引，您可以搜尋物件、執行彙總查詢，並根據物件連線/中斷連線事件建立動態群組。如需詳細資訊，請參閱機群索引。

AWS IoT 當用戶端連線或中斷連線時， 會將訊息發佈至下列 MQTT 主題：

- `$aws/events/presence/connected/clientId`：用戶端已連接到訊息代理程式。
- `$aws/events/presence/disconnected/clientId`：用戶端已與訊息代理程式中斷連線。

以下為發佈至 `$aws/events/presence/connected/clientId` 主題之連線/中斷連線訊息所包含的 JSON 元素。

clientId

連線或中斷連線之用戶端的用戶端 ID。

> **Note**
>
> 包含 # 或 + 的用戶端 ID 不會收到生命週期事件。

thingName

您的 IoT 物件名稱。只有在用戶端使用獨家物件功能進行連線時，`thingName`才會包含 。

clientInitiatedDisconnect

如果用戶端已啟動中斷連線，則為 True。否則為 false。僅在中斷連線訊息中發現。

disconnectReason

用戶端中斷連線的原因。僅在中斷連線訊息中發現。下表包含有效值，以及代理程式是否會在連線中斷時傳送 Last Will and Testament (LWT) 訊息。

中斷連線原因	說明	代理程式將傳送 LWT 訊息
AUTH_ERROR	用戶端驗證失敗或授權失敗。	是 [1]
CLIENT_INITIATED_DISCONNECT	用戶端表示將中斷連線。如果用戶端使用 WebSocket 連線，就可以透過傳送 MQTT DISCONNECT 控制項封包或 Close frame 以執行這項操作。	否
CLIENT_ERROR	用戶端執行錯誤操作導致其中斷連線。例如，若在相同連線上傳送超過 1 個 MQTT CONNECT 封包，或者用戶端嘗試以超過承載限制的承載發佈，用戶端將會中斷連線。	是
CONNECTION_LOST	用戶端伺服器連線已中斷。這可能會在高網路延遲期間或網際網路連線中斷時發生。	是
DUPLICATE_CLIENTID	用戶端使用已在使用中的用戶端 ID。在這種情況下，已經連線的用戶端會因為此中斷連線的原因而中斷連線。	是
FORBIDDEN_ACCESS	用戶端不允許連線。例如，具有連線失敗的 IP 地址用戶端將無法連線。	是 [1]
MQTT_KEEP_ALIVE_TIMEOUT	如果用戶端持續作用時間的 1.5 倍沒有用戶端伺服器通訊，則用戶端會中斷連線。	是
SERVER_ERROR	因未預期的伺服器問題而中斷連線。	是
SERVER_INITIATED_DISCONNECT	因為操作原因導致伺服器故意中斷用戶端的連線。	是

中斷連線原因	說明	代理程式將傳送 LWT 訊息
API_INITIATED_DISCONNECT	用戶端已使用 DeleteConnection API 中斷連線。	是[2]
THROTTLED	因為超過調節限制導致用戶端中斷連線。	是
WEBSOCKET_TTL_EXPIRATION	用戶端被中斷連線，因為 WebSocket 已經超過其存留時間值連接的時間。	是
CUSTOMAUTH_TTL_EXPIRATION	用戶端已中斷連線，因為其連線時間超過其自訂授權方的time-to-live值。	是

[1]如果裝置在收到此錯誤之前有作用中的連線。

[2]若要防止 Last Will and Testament (LWT) 訊息，請將 `preventWillMessage=true` 設定為覆寫 `DeleteConnection` API 的預設 LWT 傳送行為。

eventType

事件的類型。有效值為 `connected` 或 `disconnected`。

ipAddress

連線用戶端的 IP 地址。這可以是 IPv4 或 IPv6 格式。僅在連線訊息中發現。

principalIdentifier

用於身分驗證的憑證。對於 TLS 交互身分驗證憑證，此為憑證 ID。對於其他連線，此為 IAM 憑證。

sessionIdentifier

在工作階段生命週期內 AWS IoT 存在於 中的全域唯一識別符。

timestamp

事件發生的約略時間。

versionNumber

生命週期事件的版本號碼。這是每個用戶端 ID 連線的依序遞增長整數值。訂閱者戶可以使用版本號碼來推斷生命週期事件的順序。

> **Note**
>
> 用戶端連線的連線和中斷連線訊息具有相同的版本號碼。
> 版本號碼可能會略過值，並且每個事件並不保證會以 1 為值持續增加。
> 如果用戶端約一小時未連線，則版本號碼將重設為 0。若為持久工作階段，當用戶端中斷連線時間超過為持久工作階段所設的存留時間 (TTL) 後，版本號碼將重設為 0。

連線訊息具有以下結構。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

中斷連線訊息具有以下結構。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

處理用戶端中斷連線

最佳實務是一律為生命週期事件實作等待狀態，包括 Last Will and Testament (LWT) 訊息。收到中斷連線的訊息時，您的程式碼會等待一段時間並驗證裝置是否仍然離線，才採取動作。使用 SQS 延遲佇列是執行此操作的其中一種方法。當用戶端收到 LWT 或生命週期事件時，您可以將訊息加入佇列 (例如，5 秒鐘)。當該訊息變為可用並由 Lambda 或其他服務進行處理時，您可以在進行下一個動作前，先檢查該裝置是否仍處於離線狀態。

連線嘗試失敗事件

AWS IoT 當用戶端未獲授權進行連線，或設定最後一個遺囑和見證，且用戶端未獲授權發佈至最後一個遺囑主題時， 會將訊息發佈至下列 MQTT 主題。

```
$aws/events/presence/connect_failed/clientId
```

以下是發佈至`$aws/events/presence/connect_failed/clientId`主題的連線授權訊息中包含的 JSON 元素清單。

clientId

嘗試且無法連線之用戶端的用戶端 ID。

> **Note**
>
> 包含 # 或 + 的用戶端 ID 不會收到生命週期事件。

thingName

您的 IoT 物件名稱。只有在用戶端使用獨家物件功能進行連線時，`thingName`才會包含 。

timestamp

事件發生的約略時間。

eventType

事件的類型。有效值為 `connect_failed`。

connectFailureReason

連線失敗的原因。有效值為 `AUTHORIZATION_FAILED`。

principalIdentifier

用於身分驗證的憑證。對於 TLS 交互身分驗證憑證，此為憑證 ID。對於其他連線，此為 IAM 憑證。

sessionIdentifier

在工作階段生命週期內 AWS IoT 存在於 中的全域唯一識別符。

ipAddress

連線用戶端的 IP 地址。這可以是 IPv4 或 IPv6 格式。僅在連線訊息中發現。

連線失敗訊息具有下列結構。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "connect_failed",
    "connectFailureReason": "AUTHORIZATION_FAILED",
    "principalIdentifier": "12345678901234567890123456789012",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "ipAddress" : "192.0.2.0"
}
```

訂閱/取消訂閱事件

AWS IoT 當用戶端訂閱或取消訂閱 MQTT 主題時， 會將訊息發佈至下列 MQTT 主題：

```
$aws/events/subscriptions/subscribed/clientId
```

或

```
$aws/events/subscriptions/unsubscribed/clientId
```

其中 `clientId` 是連線至 AWS IoT 訊息代理程式的 MQTT 用戶端 ID。

發佈至此主題的訊息具有以下結構：

```
{
```

```
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

以下為發佈至 $aws/events/subscriptions/subscribed/*clientId* 和 $aws/events/subscriptions/unsubscribed/*clientId* 主題之訂閱與取消訂閱訊息所包含的 JSON 元素。

clientId

訂閱或取消訂閱之用戶端的用戶端 ID。

> **Note**
>
> 包含 # 或 + 的用戶端 ID 不會收到生命週期事件。

thingName

您的 IoT 物件名稱。只有在用戶端使用獨家物件功能進行連線時，thingName才會包含 。

eventType

事件的類型。有效值為 subscribed 或 unsubscribed。

principalIdentifier

用於身分驗證的憑證。對於 TLS 交互身分驗證憑證，此為憑證 ID。對於其他連線，此為 IAM 憑證。

sessionIdentifier

在工作階段生命週期內 AWS IoT 存在於 中的全域唯一識別符。

timestamp

事件發生的約略時間。

topics

用戶端已訂閱的一系列 MQTT 主題。

> **Note**
>
> 生命週期的訊息可能不會按照順序傳送。您可能會收到重複的訊息。

故障診斷 AWS IoT

協助我們改善此主題

讓我們知道如何能使其變得更好

以下資訊可能有助於診斷 AWS IoT內的常見問題。

任務

- AWS IoT Core 故障診斷指南
- AWS IoT Device Management 疑難排解指南
- AWS IoT Device Advisor 疑難排解指南
- AWS IoT 錯誤

AWS IoT Core 故障診斷指南

協助我們改善此主題

讓我們知道如何能使其變得更好

這是 的疑難排解區段 AWS IoT Core。

主題

- 診斷連線問題
- 診斷規則問題
- 診斷影子的問題
- 診斷串流限制
- 對裝置機群中斷連線進行疑難排解

診斷連線問題

協助我們改善此主題

讓我們知道如何能使其變得更好

成功連線至 AWS IoT 需要：

- 有效的連線
- 有效且作用中的憑證
- 允許所需連線和作業的政策

連線

如何找到正確的端點？

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS` 返回的 `endpointAddress`

  或

- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` 返回的 `domainName`

如何找到正確的伺服器名稱指示 (SNI) 值？

正確的 SNI 值是 describe-endpoint 傳回的 `endpointAddress`，或是 describe-domain-configuration 命令傳回的 `domainName`。其地址與上一個步驟中的端點相同。將裝置連線至 時 AWS IoT Core，用戶端可以傳送伺服器名稱指示 (SNI) 延伸，這並非必要，但強烈建議這麼做。您必須使用 SNI 擴充功能，才能使用如多帳戶註冊、自訂網域及 VPC 端點等功能。如需詳細資訊，請參閱 中的 Transport Security AWS IoT。

如何解決持續存在的連線問題？

您可以使用 AWS Device Advisor 協助疑難排解。Device Advisor 的預先建置測試可協助您針對最佳實務，驗證您的裝置軟體是否使用了 TLS、MQTT、AWS IoT Device Shadow，以及 AWS IoT 工作。

這是現有 Device Advisor 內容的連結。

身分驗證

裝置必須經過身分驗證才能連線至 AWS IoT 端點。對於使用 X.509 用戶端憑證進行身分驗證的裝置，憑證必須向 註冊 AWS IoT 並處於作用中狀態。

我的裝置如何驗證 AWS IoT 端點？

將 AWS IoT CA 憑證新增至用戶端的信任存放區。請參閱 AWS IoT Core中的伺服器驗證文件，然後遵循連結下載適當的憑證授權機構憑證。

當裝置連線到 時會檢查什麼 AWS IoT？

當裝置嘗試連接至 AWS IoT時：

1. AWS IoT 會檢查有效憑證和伺服器名稱指示 (SNI) 值。
2. AWS IoT 會檢查所使用的憑證是否已向 AWS IoT 帳戶註冊，以及是否已啟用。
3. 當裝置嘗試在 中執行任何動作 AWS IoT，例如訂閱或發佈訊息時，會檢查連接至其用來連線之憑證的政策，以確認裝置已獲授權執行該動作。

如何驗證憑證是否正確設定？

請使用 OpenSSL `s_client` 命令來測試與 AWS IoT 端點的連線：

```
openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

如需使用 `openssl s_client` 的詳細資訊，請參閱 OpenSSL s_client 文件。

如何檢查憑證的狀態？

- 列出憑證

  若您不知道憑證 ID，您可使用 `aws iot list-certificates` 命令，查看您所有憑證的狀態。

- 顯示憑證的詳細資訊

  若您知道憑證的 ID，此命令會顯示更多有關憑證的詳細資訊。

  ```
  aws iot describe-certificate --certificate-id "certificateId"
  ```

- 在 AWS IoT 主控台中檢閱憑證

  在 AWS IoT 主控台 的左側選單中，依序選擇 Secure (安全) 和 Certificates (憑證)。

從清單中選擇您用來連線的憑證，開啟其詳細資訊頁面。

於憑證的詳細資訊頁面中，您可查看其目前的狀態。

可使用詳細資訊頁面右上角的 Actions (動作)選單來變更憑證的狀態。

Authorization

AWS IoT 資源使用 AWS IoT Core 政策 來授權這些資源執行動作。若要授權動作，指定的 AWS IoT 資源必須連接政策文件，以授予執行該動作的許可。

我從代理程式收到 PUBNACK 或 SUBNACK 的回應？ 我要怎麼做？

請確定您用來呼叫的憑證已附加政策 AWS IoT。根據預設，所有發佈/訂閱的操作都會遭拒。

請確定連接的政策會授權您正嘗試執行的動作。

請確定連接的政策會授權您正嘗試執行授權動作的資源。

我的日誌中有一則 AUTHORIZATION_FAILURE 項目。

請確定您用來呼叫的憑證已附加政策 AWS IoT。根據預設，所有發佈/訂閱的操作都會遭拒。

請確定連接的政策會授權您正嘗試執行的動作。

請確定連接的政策會授權您正嘗試執行授權動作的資源。

如何查看政策授權的內容？

在AWS IoT 主控台的左側選單中，選擇安全性，然後選擇憑證。

從清單中選擇您用來連線的憑證，開啟其詳細資訊頁面。

於憑證的詳細資訊頁面中，您可查看其目前的狀態。

於憑證詳細資訊頁面的左側選單中，選擇 Policies (政策)，查看連接至憑證的政策。

選擇想要的政策以查看其詳細資訊頁面。

於政策的詳細資訊頁面中，檢閱政策的政策文件以查看其授權內容。

選擇 Edit policy document (編輯政策文件)，對政策文件進行變更。

安全性與身分

當您提供 AWS IoT 自訂網域組態的伺服器憑證時，憑證最多有四個網域名稱。

如需詳細資訊，請參閱 AWS IoT Core 端點和配額。

診斷規則問題

協助我們改善此主題
讓我們知道如何能使其變得更好

本節說明當您遇到規則問題時，要檢查的一些事項。

配置 CloudWatch Logs 進行疑難排解

對規則所發生的問題進行疑難排解的最佳方法是使用 CloudWatch Logs。當您為 啟用 CloudWatch Logs 時 AWS IoT，您可以查看觸發哪些規則及其成功或失敗。也會獲得 WHERE 子句條件是否符合的資訊。如需詳細資訊，請參閱AWS IoT 使用 CloudWatch Logs 進行監控。

最常見的問題在於授權。若您的角色未獲得在該資源上執行 AssumeRole 的授權，即會顯示在日誌中。以下為精細記錄所產生的日誌範例：

```
{
    "timestamp": "2017-12-09 22:49:17.954",
    "logLevel": "ERROR",
    "traceId": "ff563525-6469-506a-e141-78d40375fc4e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecution",
    "clientId": "iotconsole-123456789012-3",
    "topicName": "test-topic",
    "ruleName": "rule1",
    "ruleAction": "DynamoAction",
    "resources": {
        "ItemHashKeyField": "id",
        "Table": "trashbin",
        "Operation": "Insert",
        "ItemHashKeyValue": "id",
        "IsPayloadJSON": "true"
    },
```

```
    "principalId": "ABCDEFG1234567ABCD890:outis",
    "details": "User: arn:aws:sts::123456789012:assumed-role/dynamo-
testbin/5aUMInJH is not authorized to perform: dynamodb:PutItem on
 resource: arn:aws:dynamodb:us-east-1:123456789012:table/testbin (Service:
 AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException; Request ID:
 AKQJ987654321AKQJ123456789AKQJ987654321AKQJ987654321)"
}
```

以下為全域記錄所產生的類似日誌範例：

```
2017-12-09 22:49:17.954 TRACEID:ff562535-6964-506a-e141-78d40375fc4e
PRINCIPALID:ABCDEFG1234567ABCD890:outis [ERROR] EVENT:DynamoActionFailure
TOPICNAME:test-topic CLIENTID:iotconsole-123456789012-3
MESSAGE:Dynamo Insert record failed. The error received was User:
 arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJI is not authorized to
 perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
testbin
(Service: AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException;
 Request ID: AKQJ987654321AKQJ987654321AKQJ987654321AKQJ987654321).
Message arrived on: test-topic, Action: dynamo, Table: trashbin, HashKeyField: id,
 HashKeyValue: id, RangeKeyField: None, RangeKeyValue: 123456789012
No newer events found at the moment. Retry.
```

如需詳細資訊，請參閱the section called “在 CloudWatch 主控台中檢視 AWS IoT 日誌”。

診斷外部服務

外部服務受最終使用者的控制。在執行規則之前，請確認您連結至規則的外部服務已經設定，並為您的應用程式提供足夠的輸送量和容量單位。

診斷 SQL 問題

若您的 SQL 查詢並未傳回您想要的資料：

- 請檢閱錯誤訊息的日誌。
- 確認您的 SQL 語法符合訊息中的 JSON 文件。

  檢閱用於查詢中的物件和屬性名稱，及主題訊息承載的 JSON 文件中所使用名稱。如需 SQL 查詢中 JSON 格式的詳細資訊，請參閱 JSON Extensions。
- 檢查 JSON 物件或屬性名稱是否包含保留或數字字元。

如需 SQL 查詢中 JSON 物件參考中保留字元的詳細資訊，請參閱 JSON Extensions。

診斷影子的問題

協助我們改善此主題

讓我們知道如何能使其變得更好

診斷影子

問題	準則疑難排解
裝置的影子文件遭受 `Invalid JSON document` 拒絕。	如果您不熟悉 JSON，請修改此指南中提供的範本供自行使用。如需詳細資訊，請參閱影子文件範例。
我已提交正確的 JSON，但該檔案完全未存放於裝置的影子文件中，或僅存放了部分。	請確認您是否有按照 JSON 格式編排準則進行。僅有 `desired` 和 `reported` 內的 JSON 欄位會儲存。在這些部分之外的 JSON 內容 (即使格式正確) 會被忽略。
我收到裝置影子超出允許大小的錯誤。	裝置影子僅支援 8 KB 的資料。請嘗試縮短您 JSON 文件內的欄位名稱，或直接建立更多物件，以建立更多的影子。可與裝置相關聯的物件/影子數目，並不受限制。唯一的要求是，每個物件名稱在您的帳戶中皆不得重複。
當我收到裝置的影子時，其大於 8 KB。怎麼會出現這種情況？	收到時， AWS IoT 服務會將中繼資料新增至裝置的影子。該服務會在回應中包含此資料，而不會計入 8 KB 的限制之中。只有傳送至裝置影子的狀態文件內之 `desired` 與 `reported` 狀態的資料，才會計入限制。
我的請求因為版本錯誤而遭拒。我該怎麼辦？	請執行 GET 操作，以同步到文件的最新版本。當使用 MQTT 時，訂閱 ./update/accepted 主

問題	準則疑難排解
	題，以接收狀態變更的通知和 JSON 文件的最新版本。
時間戳記出現幾秒鐘的差距。	當 AWS IoT 服務收到文件或狀態文件發佈到 ./update/accepted 和 ./update/delta 訊息時，會更新個別欄位和整個 JSON 文件的時間戳記。網路傳遞的訊息可能會延遲幾秒，而導致時間戳記有幾秒鐘的差距。
我的裝置可發佈於對應的影子主題且已訂閱該主題，但當我嘗試透過 HTTP REST API 更新影子文件時，卻出現 HTTP 403。	請確認已在 IAM 中建立政策，以存取這類主題及您所使用憑證相應的動作 (UPDATE/GET/DELETE)。IAM 政策與憑證政策彼此獨立。
其他問題。	Device Shadow 服務會將錯誤記錄至 CloudWatch Logs。如要識別裝置和組態的問題，啟用 CloudWatch Logs 並檢視日誌即可取得除錯資訊。

診斷串流限制

故障診斷「超過您 AWS 帳戶的串流限制」

如果您看到 `"Error: You have exceeded the limit for the number of streams in your AWS account."`，您可以清除帳戶中未使用的串流，而不要求增加限制。

若要清除您使用 AWS CLI 或 SDK 建立的未使用串流：

```
aws iot delete-stream –stream-id value
```

如需詳細資訊，請參閱 delete-stream。

> **Note**
>
> 您可使用 `list-streams` 命令來找出串流 ID。

對裝置機群中斷連線進行疑難排解

協助我們改善此主題

讓我們知道如何能使其變得更好

AWS IoT 裝置機群中斷連線可能有多種原因。本文說明如何診斷中斷連線原因，以及如何處理定期維護 AWS IoT 服務或限流限制所造成的中斷連線。

如要診斷中斷連線的原因

您可查看 CloudWatch 中的 AWSIotLogsV2 日誌群組，在日誌項目 `disconnectReason` 欄位中識別中斷連線的原因。

您也可以使用 AWS IoT生命週期事件功能來識別中斷連線的原因。如果您已訂閱生命週期的中斷連線事件 (`$aws/events/presence/disconnected/clientId`)，當中斷連線發生 AWS IoT 時，您會收到來自 的通知。您可在通知的 `disconnectReason` 欄位中識別中斷連線的原因。

如需詳細資訊，請參閱 CloudWatch AWS IoT 日誌項目和生命週期事件。

疑難排解因 AWS IoT 服務維護而中斷連線的問題

由 AWS IoT服務維護引起的中斷連線會記錄為 `SERVER_INITIATED_DISCONNECT` AWS IoT的生命週期事件和 CloudWatch。若要處理這些中斷連線，請調整您的用戶端設定，以確保您的裝置可以自動重新連線至 AWS IoT 平台。

如要對因調節限制而造成的中斷連線進行疑難排解

調節限制所造成的中斷會記錄為 `THROTTLED` AWS IoT的生命週期事件和 CloudWatch。如要處理這些中斷連線，您可依裝置計數增加，而請求訊息代理程式限制增加。

如需詳細資訊，請參閱 AWS IoT 核心訊息代理程式。

AWS IoT Device Management 疑難排解指南

協助我們改善此主題

讓我們知道如何能使其變得更好

這是 的疑難排解區段 AWS IoT Device Management。

主題

- AWS IoT 任務故障診斷
- AWS IoT 命令故障診斷
- 機群索引故障診斷
- AWS IoT 裝置管理軟體套件目錄故障診斷

AWS IoT 任務故障診斷

這是 AWS IoT 任務的疑難排解區段。

如何尋找 AWS IoT 任務端點？

如何尋找 AWS IoT 任務控制平面端點？

AWS IoT 任務支援使用 HTTPS 通訊協定控制平面 API 操作。確認您已使用 HTTPS 通訊協定連線至正確的控制平面端點。

如需 AWS 區域特定的端點清單，請參閱AWS IoT 核心 - 控制平面端點。

如需符合 FIPS AWS IoT 標準任務控制平面端點的清單，請參閱依服務分類的 FIPS 端點

> **Note**
>
> AWS IoT 任務和 AWS IoT Core 共用相同的 AWS 區域特定端點。

如何尋找 AWS IoT 任務資料平面端點？

AWS IoT 任務支援使用 HTTPS 和 MQTT 通訊協定的資料平面 API 操作。確認您已使用 HTTPS 或 MQTT 通訊協定連線至正確的資料平面端點。

- HTTPS 通訊協定
  - 使用下列 CLI describe-endpoint 命令，如下所示或 `DescribeEndpoint` REST API。對於端點類型，請使用 `iot:Jobs`。

    ```
    aws iot describe-endpoint --endpoint-type iot:Jobs
    ```
- MQTT 通訊協定

- 使用下列 CLI describe-endpoint 命令，如下所示或 DescribeEndpoint REST API。對於端點類型，請使用 iot:Data-ATS。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如需 FIPS 相容AWS IoT 任務資料平面端點的清單，請參閱依服務的 FIPS 端點

如何監控 AWS IoT 任務活動並提供指標？

使用 Amazon CloudWatch 監控 AWS IoT 任務活動可提供持續 AWS IoT 任務操作的即時可見性，並透過 AWS IoT 規則使用 CloudWatch 警示協助控制成本。您必須先設定記錄，才能監控 AWS IoT 任務活動和設定 CloudWatch 警示。如需設定記錄的詳細資訊，請參閱 設定 AWS IoT 記錄。

如需 Amazon CloudWatch 以及如何透過 IAM 使用者角色設定許可以使用 CloudWatch 資源的詳細資訊，請參閱 Amazon CloudWatch 的身分和存取管理。

如何使用 Amazon CloudWatch 設定 AWS IoT 任務指標和監控？

若要設定 AWS IoT 記錄，請遵循設定 AWS IoT logging. AWS IoT logging 設定中所述的步驟，可以在 中完成 AWS 管理主控台 AWS CLI，或 API。針對特定物件群組設定的 AWS IoT 記錄，只能在 AWS CLI 或 API 中完成。

AWS IoT 任務指標區段包含用於監控任務活動 AWS IoT 的任務 AWS IoT 指標。它說明如何檢視 AWS 管理主控台 和 中的指標 AWS CLI。

此外，您可以設定 CloudWatch 警示，提醒您要密切監控的特定指標。如需警示設定的指引，請參閱使用 Amazon CloudWatch 警示。

裝置機群和單一裝置故障診斷

任務執行會**QUEUED**無限期維持 的狀態

當狀態為 的任務執行QUEUED未繼續進行下一個邏輯狀態，例如 IN_PROGRESS、 FAILED或 TIMED_OUT時，以下其中一個案例可能是原因：

- 在位於 CloudWatch 主控台的 CloudWatch 日誌中檢閱您的裝置活動。如需詳細資訊，請參閱AWS IoT 使用 CloudWatch Logs 監控。
- 與任務和後續任務執行相關聯的 IAM 角色可能沒有連接到該 IAM 角色的 IAM 政策的其中一個政策陳述式中列出的正確許可。使用 describe-job API 識別連結至該任務和後續任務執行的

IAM 角色，並檢閱 IAM 政策以取得正確的許可。政策許可陳述式更新後，您應該能夠對資源執行 AssumeRole API 命令。

未為我的物件或物件群組建立任務執行

當任務將其狀態更新為 時IN_PROGRESS，它會開始將任務文件推展到目標群組中的所有裝置。此狀態更新會為每個目標裝置建立任務執行。如果未為其中一個目標裝置建立任務執行，請參閱下列指引：

- 是否為任務thing的直接目標、任務的狀態為 IN_PROGRESS，以及任務是否並行？ 如果符合所有三個條件，則任務仍會將任務執行傳送至目標群組中的所有裝置，且該特定 thing 尚未收到其任務執行。
  - 在 AWS 管理主控台中檢閱目標群組中的裝置以取得任務和任務狀態，或使用 describe-job API 命令。
  - 使用 describe-job API 命令來檢閱任務是否將 IsConcurrent 屬性設定為 true 或 false。如需詳細資訊，請參閱任務限制。
- thing 並非由任務直接鎖定目標。
  - 如果 Thing 已新增至 ，ThingGroup且任務以 為目標ThingGroup，則請確認 Thing是 的一部分ThingGroup。
  - 如果任務是狀態為 IN_PROGRESS且並行的快照任務，則任務仍會將任務執行傳送至目標群組中的所有裝置，且該特定 Thing 尚未收到其任務執行。
  - 如果任務是狀態為 IN_PROGRESS且並行的連續任務，則任務仍會將任務執行傳送至目標群組中的所有裝置，且該特定 Thing 尚未收到其任務執行。僅適用於持續任務，您也可以Thing從 移除 ，ThingGroup然後將 Thing 新增至 ThingGroup。
  - 如果任務是狀態為 IN_PROGRESS且不是並行的快照任務，則 AWS IoT 任務可能不會確認 Thing或 ThingGroup成員關係。建議您在AddThingToThingGroup呼叫後新增幾秒鐘的等待時間，然後再建立 Job。或者，您可以將目標選擇切換為 Continuous，讓服務回填延遲Thing和ThingGroup成員資格連接事件。

新任務因**LimitedExceededException**錯誤而失敗

如果您的任務建立失敗且錯誤回應為 LimitedExceededException，請呼叫 list-jobs API 並使用 檢閱所有任務isConcurrent=true，以判斷您是否處於任務並行限制。如需並行任務的其他資訊，請參閱任務限制。若要檢視任務並行限制和請求提高限制，請參閱AWS IoT Device Management 任務限制和配額。

任務文件大小限制

任務文件大小受限於 MQTT 承載大小。如果您需要大於 32 kB (KB)、32，000 B （位元組）的任務文件，請在 Amazon S3 中建立和存放任務文件，並在 `CreateJob` API 或使用 的 `documentSource` 欄位中新增 Amazon S3 物件 URL AWS CLI。對於 AWS 管理主控台，在建立任務時，在 Amazon S3 URL 文字方塊中新增 Amazon S3 物件 URL。

- AWS 管理主控台 建立任務文件：使用 建立和管理任務 AWS 管理主控台
- AWS CLI 建立任務文件：使用 建立和管理任務 AWS CLI
- `CreateJob` API 文件：CreateJob

裝置端 MQTT 訊息請求調節限制

如果您收到錯誤代碼 400 `ThrottlingException`，裝置端 MQTT 訊息會因為達到同時裝置端請求的限制而失敗。如需節流AWS IoT Device Management 限制以及是否可以調整的詳細資訊，請參閱任務限制和配額。

連線逾時錯誤

錯誤代碼 400 `RequestExpired`表示由於高延遲或低用戶端逾時值而導致連線失敗。

- 如需測試用戶端與伺服器端之間連線的資訊，請參閱測試與裝置資料端點的連線。

無效的 API 命令

確認已輸入正確的 API 命令，以避免顯示 API 命令無效的錯誤訊息。如需所有 API 命令的完整清單，請參閱 AWS IoT API AWS IoT 參考。

服務端連線錯誤

錯誤代碼 503 `ServiceUnavailable`表示錯誤源自伺服器端。

- 如需AWS Health 儀板表 所有 AWS 服務的目前狀態，請參閱 （所有服務）。 AWS
- 如需了解您個人的目前狀態，請參閱 AWS Health 儀板表 （個人 AWS 帳戶) AWS 帳戶。

AWS IoT 命令故障診斷

這是 AWS IoT Device Management Commands 的疑難排解區段。

命令執行問題

命令執行會保持在 CREATED 狀態

當命令執行保持 CREATED 狀態且未繼續 IN_PROGRESS或其他狀態時，請考慮下列事項：

- 確認裝置已連線至 AWS IoT Core，並已訂閱命令請求主題。
- 檢查裝置政策是否允許命令請求主題`iot:Receive`上的 `iot:Subscribe`和 ，以及命令回應主題`iot:Publish`上的 。
- 如果裝置離線並使用 MQTT 持久性工作階段，命令會在 AWS IoT Core 中等待。當裝置在持久性工作階段逾時和執行逾時之前重新連線時，可以處理 命令。如果執行逾時過期，則執行會轉換為 TIMED_OUT。

UpdateCommandExecution 上的 DataConflict 錯誤

當平行或在短時間內對服務提出多個`UpdateCommandExecution`請求 （ 例如，緊`IN_PROGRESS`接著）時，就會`DataConflict`發生錯誤`SUCCEEDED`。

若要解決此問題：

- 訂閱 `/accepted`和 `/rejected`回應主題，以確認在傳送下一個狀態更新之前已處理每個狀態更新。
- 接收`DataConflict`錯誤時，使用指數退避實作重試邏輯。

命令執行意外移至 TIMED_OUT 終端機狀態

當命令執行在裝置可以處理它`TIMED_OUT`之前轉換為 時：

- 檢閱為命令執行設定的逾時值。預設逾時可能太短，不適合您的使用案例。
- 如果裝置在傳送命令時離線，請確認裝置已重新連線，並在逾時過期之前收到執行請求。
- 雲端啟動`TIMED_OUT`是非終端的。裝置仍然可以將執行更新為終端機狀態 (`SUCCEEDED`、`REJECTED`、 `FAILED`或 `TIMED_OUT`)。

> **Note**
>
> 裝置起始`TIMED_OUT`是終端機狀態，無法對此命令執行進行進一步更新。

如何檢視 CloudWatch 日誌中的錯誤？

`UpdateCommandExecution` MQTT 請求的錯誤會記錄在 Amazon CloudWatch 的`AWSIoTLogsV2`日誌群組中。若要啟用記錄和檢視日誌，請參閱 設定 AWS IoT 記錄。

UpdateCommandExecution 錯誤代碼

當 UpdateCommandExecution MQTT 請求失敗時，服務會將錯誤回應發佈至/rejected主題。錯誤回應包含錯誤代碼和訊息。下表列出可傳回的錯誤代碼。

錯誤碼	可重試	說明
InvalidStateTransition	否	不允許請求的狀態轉換。例如，從 轉換為 SUCCEEDED IN_PROGRESS ，更新裝置啟動的TIMED_OUT 執行。
TerminalStateReached	否	命令執行已經達到結束狀態，無法更新。
ResourceNotFound	否	指定的命令執行不存在。
DataConflict	是	命令執行同時修改。當快速連續傳送狀態更新時，可能會發生這種情況。訂閱 /accepted 和 /rejected 回應主題，以確認在傳送下一個狀態更新之前已處理每個狀態更新。如需詳細資訊，請參閱 UpdateCommandExecution 上的 DataConflict 錯誤。
InternalError	是	發生內部伺服器錯誤。以指數退避重試請求。

以下是發佈至 /rejected主題的錯誤回應範例：

```
{
    "clientToken": "client-token-1",
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "error": "DataConflict",
    "errorMessage": "The command execution was modified concurrently"
}
```

機群索引故障診斷

機群索引服務的疑難排解彙總查詢

若您遇到類型不符錯誤，您可使用 CloudWatch Logs 來疑難排解問題。在機群索引服務寫入日誌之前，必須先啟用 CloudWatch Logs。如需詳細資訊，請參閱AWS IoT 使用 CloudWatch Logs 進行監控。

若要對非受管欄位進行彙總查詢，您必須指定您在 `customFields` 引數中定義的欄位傳遞給 `UpdateIndexingConfiguration` 或 update-indexing-configuration。如果欄位值與設定的欄位資料類型不一致，則在您執行彙總查詢時會忽略此值。

若因類型不符而無法索引欄位，機群索引服務會將錯誤日誌發送至 CloudWatch 日誌。錯誤日誌包含欄位名稱、無法轉換的值，以及裝置的物件名稱。以下是範例錯誤日誌：

```
{
  "timestamp": "2017-02-20 20:31:22.932",
  "logLevel": "ERROR",
  "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
  "accountId": "000000000000",
  "status": "SucceededWithIssues",
  "eventType": "IndexingCustomFieldFailed",
  "thingName": "thing0",
  "failedCustomFields": [
    {
      "Name": "attributeName1",
      "Value": "apple",
      "ExpectedType": "String"
    },
    {
      "Name": "attributeName2",
      "Value": "2",
      "ExpectedType": "Boolean"
    }
  ]
}
```

如果裝置已中斷連線大約一小時，則可能會缺少連線狀態的 `timestamp` 值。若為持久工作階段，當用戶端中斷連線時間超過為持久工作階段所設的存留時間 (TTL) 後，則可能會缺少此值。只有針對用戶端 ID 有相符物件名稱的連線，連線狀態資料才會建立索引。（用戶端 ID 是用來連接裝置的值 AWS IoT Core。）

疑難排解機群索引組態

無法降級機群索引組態

移除與機群指標或動態群組相關聯的資料來源時，不支援機群索引組態降級。

例如，若您的索引組態具有登錄資料、影子資料和連線資料，且查詢 `thingName:TempSensor* AND shadow.desired.temperature>80` 有機群指標，則更新索引組態以僅包含登錄資料將導致錯誤。

不支援修改現有機群指標使用的自訂欄位。

由於不相容的機群指標或動態群組，無法更新您的索引組態

若由於不相容的機群指標或動態群組而無法更新索引組態，請先刪除不相容的機群指標或動態群組，然後再更新索引組態。

對位置索引和地理查詢進行故障診斷

若要疑難排解位置索引和地理查詢中不相符的類型錯誤，您可以啟用 CloudWatch 日誌。如需如何使用 AWS IoT CloudWatch 監控 的詳細資訊，請遵循step-by-step指南。

當您使用地理位置查詢為位置資料編製索引時，您在 中指定的位置欄位`geoLocations`必須符合您傳遞給 的位置欄位`UpdateIndexingConfiguration`。如果不相符，機群索引會將不相符的類型錯誤傳送至 CloudWatch。錯誤日誌包含欄位名稱、無法轉換的值，以及裝置的物件名稱。

以下是範例錯誤日誌：

```
{
"timestamp": "2023-11-09 01:39:43.466",
    "logLevel": "ERROR",
    "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "IndexingGeoLocationFieldFailed",
    "thingName": "thing0",
    "failedGeolocationFields": [
        {
"Name": "attributeName1",
            "Value": "apple",
            "ExpectedType": "Geopoint"
        }
```

```
    ],
    "reason": "failed to index the field because it could not be converted to one of
 the expected geoLocation formats."
}
```

如需詳細資訊，請參閱索引位置資料。

機群指標疑難排解

無法在 CloudWatch 中查看資料點

如果能夠建立機群指標，卻無法在 CloudWatch 中看到資料點，原因可能是沒有符合查詢字串條件的物件。

請參閱下列範例指令，了解如何建立機群指標：

```
aws iot create-fleet-metric --metric-name "example_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature>80" --period 60 --aggregation-field
 "attributes.temperature" --aggregation-type name=Statistics,values=count
```

如果沒有符合查詢字串條件 `--query-string "thingName:TempSensor* AND attributes.temperature>80"` 的物件：

- 您能夠使用 `values=count` 建立機群指標；CloudWatch 中會顯示資料點。`count` 數值的資料點一律為 0。
- 如果使用 `values` 而非 `count`，您能夠建立機群指標，卻無法在 CloudWatch 中看到機群指標，而且 CloudWatch 中也不會顯示任何資料點。

AWS IoT 裝置管理軟體套件目錄故障診斷

這是 AWS IoT Device Management Software Package Catalog 的疑難排解區段。

對錯誤訊息進行一般故障診斷

本節列出在整個軟體套件版本生命週期中常見的錯誤。

`HeadBucket` 錯誤

呼叫 HeadBucket API 操作或 head-bucket CLI 命令，以在任務部署期間驗證用於檔案上傳的 Amazon S3 儲存貯體時，會出現下列錯誤訊息。

如需在任務部署期間使用 Amazon S3 儲存貯體上傳檔案的詳細資訊，請參閱 檔案上傳的預先簽章 URL。

```
InvalidRoleException
    "Permission denied when attempting to use role %s to access bucket %s."
```

```
InvalidRequestException
    "Cross region S3 bucket is not supported for presigned url upload placeholder"
```

```
InvalidRequestException
    "S3 bucket in job document presigned url upload placeholder not found"
```

```
InvalidRequestException
    "Given S3 bucket name is invalid."
```

```
InvalidRequestException
    "Provided S3 bucket is not valid: %s. Error: %s"
```

Amazon S3 GetObject

提供無效的引數時，會發生下列錯誤訊息，導致 Amazon S3 `GetObject` API 操作失敗。

```
InvalidRequestException
    "Provided argument for presigned url is invalid"
```

Amazon S3 版本 ID 支援

使用版本控制控制請求存取 Amazon S3 儲存貯體時，請務必包含您的 `versionId`或以下可能填入的錯誤。

如需使用版本控制控制之 Amazon S3 儲存貯體的詳細資訊，請參閱在 Amazon S3 儲存貯體中使用版本控制

```
InvalidRequestException
    "VersionId not found when attempting to access s3 url"
```

檔案上傳之預先簽章 URL 內的預留位置

在用於在任務部署期間將檔案上傳至目的地 Amazon S3 儲存貯體的預先簽章 URL 內遇到預留位置問題時，會出現下列錯誤訊息。如需在任務部署期間使用 Amazon S3 儲存貯體上傳檔案以及本機預留位置的詳細資訊，請參閱 檔案上傳的預先簽章 URL。

無法辨識本機預留位置時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "Undefined placeholder, ${...}, inside of presign url upload parameter"
```

嘗試在非用於檔案上傳的預先簽章 URL 中使用本機預留位置時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "Local placeholder, ${...}, is only valid inside of presign url upload"
```

Amazon S3 URL 巢狀不正確

當 Amazon S3 URL 在另一個預留位置內巢狀不正確時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "${aws:%s[...]} should not be the second layer pattern."
```

套件版本成品巢狀

當套件版本成品預先簽章的 URL 在另一個預留位置內巢狀不正確時，會顯示下列錯誤訊息。

```
InvalidJobDocumentException
    "${aws:iot:package:[...]:artifact:s3-presigned-url} cannot be nested inside another
 placeholder."
```

缺少套件版本成品

找不到參考的套件版本成品時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "Package %s version %s does not have an associated artifact to generate an S3
 presigned url."
```

軟體套件和套件 Verion 預留位置

由於 destinationPackageVersions 參數中參考的多個軟體套件和套件版本，或套件版本詳細資訊頁面上的版本 ARN 索引標籤，導致軟體套件和套件版本的任務文件預留位置無法解析為任務部署所需的有效值時，會顯示下列錯誤訊息。

```
InvalidJobDocumentException
    "Cannot resolve empty package name and version name given multiple elements in
 destination package versions."
```

使用空白軟體套件和套件版本

當您嘗試使用空白套件或套件版本，但任務文件中沒有其他套件時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "Empty package name and version name have to be used in pair."
```

任務文件中的空值使用

當您嘗試在任務文件中將 指定$null為套件版本時，會出現下列錯誤訊息。使用 CreateJob API 操作時， $null 只能在 destinationPackageVersions 參數內使用。

```
InvalidJobDocumentException
    "$null is not allowed to be referenced as a package version in job documents."
```

套件版本中的所有屬性

當您嘗試使用套件版本中的所有屬性，並以額外的文字或預留位置包圍它時，會出現下列錯誤訊息。

如需在軟體套件版本中使用所有屬性的詳細資訊，請參閱 AWS IoT 任務的替代參數

```
InvalidJobDocumentException
    "The package version attribute placeholder for all attributes has to be a
 json value by itself and not appended with other strings or nested with other
 placeholders."
```

檔案上傳的預先簽章 URL 中的本機預留位置限制

當您超過在任務部署期間用於檔案上傳之預先簽章 URL 的本機預留位置數量限制時，會顯示下列錯誤訊息。

如需在任務部署期間使用預先簽章 URL 進行檔案上傳的詳細資訊，請參閱 檔案上傳的預先簽章 URL

```
InvalidJobDocumentException
```

```
    "The occurrence of local placeholder %s within S3 presigned url upload placeholder
 exceeds limit of %d."
```

Amazon S3 儲存貯體中的本機預留位置

當您嘗試將本機預留位置 URL 放置在 Amazon S3 儲存貯體名稱中，以在任務部署期間用於檔案上傳的預先簽章 URL 預留位置時，會出現下列錯誤訊息。

如需在任務部署期間使用預先簽章 URL 進行檔案上傳的詳細資訊，請參閱 檔案上傳的預先簽章 URL

```
InvalidJobDocumentException
    "S3 bucket name in presigned url upload is not allowed to contain any placeholders"
```

開啟和關閉括號

當您將參數或預留位置新增至沒有關閉架構 "}" 的任務文件時，會出現下列錯誤訊息。

```
InvalidJobDocumentException
    "One or more parameters or placeholders are not terminated."
```

具有 Amazon S3 預先簽章 URL 的 IAM 角色

當您嘗試在沒有 IAM 角色的任務文件中使用 Amazon S3 預先簽章的 URL 時，會出現下列錯誤訊息。

如需 Amazon S3 預先簽章 URLs的詳細資訊，請參閱使用預先簽章URLs。

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document."
```

套件版本成品具有 Amazon S3 預先簽章 URL 的 IAM 角色

當您嘗試在不含 IAM 角色的任務文件中使用代表套件版本成品的 Amazon S3 預先簽章 URL 時，會出現下列錯誤訊息。

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document for package %s version %s artifact."
```

軟體物料清單錯誤訊息

本節列出與連結至套件版本之軟體物料清單 (SBOM) 相關的常見錯誤。

SBOM 關聯請求的輸入驗證

使用 AssociateSbomWithPackageVersion API 操作且 s3Location 參數為 null 時，會出現下列錯誤訊息。

```
InvalidRequestException "Associate request needs to include SBOM reference"
```

如需 AssociateSbomWithPackageVersion API 操作的詳細資訊，請參閱 AssociateSbomWithPackageVersion。

SBOM 驗證錯誤

本節列出與軟體套件版本相關聯的軟體物料清單 (SBOM) 初始驗證期間常見的錯誤。

使用 AssociateSbomWithPackageVersion API 操作並在 s3Location 參數bucket中顯示下列錯誤訊息為 null。

```
InvalidRequestException "S3 bucket name for SBOM cannot be null"
```

當 AssociateSbomWithPackageVersion API 操作的 s3Location 參數bucket中的字串太長時，會出現下列錯誤訊息。

```
InvalidRequestException "S3 bucket name for SBOM is illegal. String length exceeds
 limit"
```

參數key為 null 時，會出現下列錯誤訊息。

```
InvalidRequestException "S3 key name for SBOM cannot be null"
```

當 AssociateSbomWithPackageVersion API 操作的 s3Location 參數key中的字串太長時，會出現下列錯誤訊息。

```
InvalidRequestException "S3 key name for SBOM is illegal. String length exceeds limit"
```

當 AssociateSbomWithPackageVersion API 操作的 s3Location 參數version中的字串為 null 時，會出現下列錯誤訊息。

```
InvalidRequestException "S3 object version for SBOM cannot be null"
```

當 AssociateSbomWithPackageVersion API 操作的 s3Location 參數version中的字串太長時，會出現下列錯誤訊息。

```
InvalidRequestException "S3 object version for SBOM is illegal. String length exceeds
 limit"
```

當存放在 Amazon S3 儲存貯體中的 SBOM zip 封存檔案大小過大時，會顯示下列錯誤訊息。

```
InvalidRequestException "S3 object file size exceeds limit"
```

當您使用 AssociateSbomWithPackageVersion API 操作且目前進行中的 SBOM 驗證數量已達上限時，會顯示下列錯誤訊息。

```
LimitExceededException "Too many ongoing SBOM validation workflows. Please wait and
 retry"
```

Amazon S3 儲存貯體中 SBOM 檔案的存取問題

當另一個實體因為 Amazon S3 儲存貯體不存在或尚未授予存取 Amazon S3 儲存貯體的適當許可而無法存取 Amazon S3 儲存貯體時，會顯示下列錯誤訊息。

如需存取 Amazon S3 儲存貯體所需許可政策的詳細資訊，請參閱 軟體物料儲存帳單。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket exists and S3 permission is granted."
```

當另一個實體因為 Amazon S3 儲存貯體不存在或尚未授予適當的許可來存取存放在 Amazon S3 儲存貯體中的內容，而無法存取 key 參數中的 SBOM zip 封存檔案時，會顯示下列錯誤訊息。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the key
 exists and S3 permission is granted."
```

當另一個實體因儲存貯體、金鑰和版本 ID 不存在而無法存取 Amazon S3 儲存貯體，或未授予存取 Amazon S3 儲存貯體的適當許可時，會顯示下列錯誤訊息。此外，如果授予的許可不足以存取 Amazon S3 儲存貯體中的 SBOM zip 封存檔案，則可能會出現此錯誤訊息。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket/key/version exists and S3 permission is granted."
```

當另一個實體因為儲存貯體位於另一個區域而無法存取 Amazon S3 儲存貯體時，會顯示下列錯誤訊息。

```
InvalidRequestException "Cross-region S3 bucket for %s is not supported."
```

當另一個實體因為使用 AssociateSbomWithPackageVersion API 操作時錯誤拼寫 bucket、 或 version 參數而無法存取 Amazon S3 儲存貯體時key，會顯示下列錯誤訊息。

```
InvalidRequestException "Please make sure SBOM S3 bucket name/key length/version is
 valid"
```

AWS IoT Device Advisor 疑難排解指南

協助我們改善此主題

讓我們知道如何能使其變得更好

一般

問：我可以平行執行多個測試套件嗎？

答案：是。Device Advisor 現在支援使用裝置層級端點，在不同裝置上執行多個測試套件。如果您使用帳戶級別的端點，則可以一次執行一個套件，因為每個帳戶只會有一個 Device Advisor 端點。如需詳細資訊，請參閱如何設定裝置。

問：我從我的裝置看到 TLS 連線遭到 Device Advisor 的拒絕。這是預期的行為嗎？

答：是。Device Advisor 會在每次測試執行的前後拒絕 TLS 連線。我們建議使用者實作裝置重試機制，以獲取 Device Advisor 的全自動化測試體驗。如果您執行具有多個測試案例的測試套件 (例如 TLS 連線、MQTT 連線和 MQTT 發佈)，則建議您為裝置建置一個機制。該機制可以嘗試每 5 秒連線到我們的測試端點，持續一到兩分鐘。然後，您可以採取自動方式依序執行多個測試用例。

問：我能否取得從我的帳戶呼叫 Device Advisor API 的歷史記錄，以進行安全分析和作業的疑難排解？

答：可以。若要接收您帳戶上 Device Advisor API 呼叫的歷史記錄，您只需在 AWS IoT 管理主控台中開啟 CloudTrail，並將事件來源篩選為 iotdeviceadvisor.amazonaws.com。

問：我要如何在 CloudWatch 中檢視 Device Advisor 日誌？

答：若您將所需的政策 (例如 CloudWatchFullAccess) 新增至您的服務角色 (請參閱 設定)，則於測試套件執行期間所產生的日誌會上傳至 CloudWatch。如果測試套件中至少有一個測試案例，則會使用兩個日誌串流建立「aws/iot/deviceadvisor/$testSuiteId」日誌群組。有一個串流是 "$testRunId"，包括在測試套件中執行測試案例前後所採取動作的日誌，例如設定和清理步驟。另一個日誌串流是 "$suiteRunId_$testRunId"，專門用於測試套件的執行。從裝置 和 傳送的事件 AWS IoT Core 將記錄到此日誌串流。

問：裝置許可角色的用途為何？

答：Device Advisor 位於您的測試裝置與 之間 AWS IoT Core ，以模擬測試案例。其會接受來自您測試裝置的連接和訊息，透過承擔您裝置許可角色並代表您啟動連線，將其轉送至 AWS IoT Core 。請務必確保裝置角色許可與您用於執行測試之憑證上的許可相同。當 Device Advisor 使用裝置許可角色 AWS IoT Core 代表您啟動與 的連線時，不會強制執行 AWS IoT 憑證政策。不過，會強制執行您設定之裝置許可角色的許可。

問：Device Advisor 支援哪些區域？

答：在 us-east-1、us-west-2、ap-northeast-1 和 eu-west-1 區域中支援 Device Advisor。

問：為什麼我會看到不一致的結果？

答：結果不一致的主要原因之一是將測試的 `EXECUTION_TIMEOUT` 值設定過低。如需建議和預設 `EXECUTION_TIMEOUT` 值的詳細資訊，請參閱 Device Advisor 測試案例。

問：Device Advisor 支援的 MQTT 通訊協定為何？

答：Device Advisor 支援具有 X509 用戶端憑證的 MQTT 版本 3.1.1。

問：如果嘗試將裝置連結到測試端點後，我的測試案例依舊失敗，並且顯示執行逾時的訊息，那麼該怎麼辦？

答：驗證建立要用作裝置角色的 IAM 角色之下的所有步驟。如果測試仍然失敗，則可能是因為裝置未傳送正確的伺服器名稱指示 (SNI) 擴充功能，Device Advisor 必須有這項功能才能運作。正確的 SNI 值是遵循設定您的裝置區段時傳回的端點地址。 AWS IoT 也需要裝置將伺服器名稱指示 (SNI) 擴充功能傳送至 Transport Layer Security (TLS) 通訊協定。如需詳細資訊，請參閱 中的傳輸安全性 AWS IoT。

問：我的 MQTT 連線失敗，並顯示 "libaws-c-mqtt: AWS_ERROR_MQTT_UNEXPECTED_HANGUP" 錯誤，或目前正自動從 Device Advisor 端點中斷裝置的 MQTT 連線。如何解決此錯誤？

答：導致這種特定錯誤代碼和非預期中斷連線狀況的原因有很多，但最有可能與連接至裝置的裝置角色有關。以下檢查點 (依優先順序排列) 將解決此問題。

- 連接至裝置的裝置角色必須具有執行測試所需的最低 IAM 許可。Device Advisor 會使用連接的裝置角色，代表測試裝置來執行 AWS IoT MQTT 動作。若缺少所需的許可，當裝置嘗試連線至 Device Advisor 端點時，則會顯示 `AWS_ERROR_MQTT_UNEXPECTED_HANGUP` 錯誤或發生非預期的中斷連線狀況。例如，若您選擇執行 MQTT Publish (MQTT 發佈) 測試案例，則必須在角色中包含 Connect (連線) 和 Publish (發佈) 動作，以及對應的 ClientId 和 Topic (您在提供多個值時可使用逗號加以區隔，且可使用萬用字元 (*) 來提供字首值)。例如：若要提供開頭為 `TestTopic` 的任一主題發佈許可，您可提供 "`TestTopic*`" 作為資源值。以下是一些政策範例。
- 在裝置角色中為資源類型定義的值，與程式碼中使用的實際值不相符。例如：角色中定義的 ClientId 與裝置代碼中使用的實際 ClientId 不相符。諸如 ClientId、Topic 和 TopicFilter 等值，在裝置角色與程式碼當中必須相同。
- 裝置所連接的裝置憑證必須處於作用中，且使用資源所需的動作許可，將政策連接至這些裝置。請注意，裝置憑證政策會授予或拒絕存取 AWS IoT 資源和 AWS IoT Core 資料平面操作。Device Advisor 會要求您將作用中的裝置憑證連接至您的裝置，以便授予測試案例期間所使用的動作許可。

AWS IoT 錯誤

協助我們改善此主題

讓我們知道如何能使其變得更好

本節列出 傳送的錯誤代碼 AWS IoT。

訊息代理程式錯誤代碼

錯誤碼	錯誤說明
400	錯誤的請求。
401	未經授權。
403	禁止。
426	需要升級。
503	服務無法使用。

身分與安全性錯誤代碼

錯誤碼	錯誤說明
401	未經授權。

Device Shadow 錯誤代碼

錯誤碼	錯誤說明
400	錯誤的請求。
401	未經授權。
403	禁止。
404	未找到。
409	衝突。
413	請求過大。
422	無法處理請求。
429	請求過多。
500	內部錯誤。
503	服務無法使用。

AWS IoT 裝置SDKs、行動SDKs和 AWS IoT 裝置用戶端

此頁面摘要說明 AWS IoT 裝置 SDKs、開放原始碼程式庫、開發人員指南、範例應用程式和移植指南，以協助您使用 AWS IoT 和您選擇的硬體平台建置創新的 IoT 解決方案。

這些 SDK 適用於您的 IoT 裝置。如果您是開發 IoT 應用程式以在行動裝置上使用，請參閱 AWS 行動 SDKs。如果您是開發 IoT 應用程式或伺服器端程式，請參閱 AWS SDKs。

AWS IoT 裝置 SDKs

AWS IoT 裝置SDKs包含開放原始碼程式庫、包含範例的開發人員指南，以及移植指南，讓您可以在所選的硬體平台上建置創新的 IoT 產品或解決方案。

> **Note**
>
> AWS IoT 裝置 SDKs 已發行 MQTT 5 用戶端。 AWS IoT 裝置SDKs不支援在 macOS 上使用 TLS 1.3。

這些 SDK 可協助您使用 MQTT 和 WSS 通訊協定將 IoT 裝置連接至 AWS IoT 。

C++

AWS IoT C++ 裝置 SDK

AWS IoT C++ 裝置 SDK 可讓開發人員使用 AWS 和 AWS IoT APIs建置連線的應用程式。此 SDK 特別是為並未受限於資源的裝置所設計，需要訊佇列、多重執行緒支援、最新語言功能等進階功能。如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上的裝置 SDK C++ v2
- AWS IoT 裝置 SDK C++ v2 讀我檔案
- AWS IoT 裝置 SDK C++ v2 範例
- AWS IoT 裝置 SDK C++ v2 API 文件

Python

AWS IoT 適用於 Python 的裝置 SDK

適用於 Python 的 AWS IoT 裝置 SDK 可讓開發人員撰寫 Python 指令碼，以使用其裝置透過 MQTT 或 MQTT 透過 WebSocket 通訊協定存取 AWS IoT 平台。透過將其裝置連接到 AWS IoT，使用者可以安全地使用由 提供的訊息中介裝置、規則和影子 AWS Lambda， AWS IoT 以及 Kinesis 和 Amazon S3 等 AWS 其他服務。

- GitHub 上適用於 Python 的AWS IoT 裝置 SDK v2
- AWS IoT 適用於 Python v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 Python v2 的 裝置 SDK 範例
- AWS IoT 適用於 Python v2 的 裝置 SDK API 文件

JavaScript

AWS IoT 適用於 JavaScript 的裝置 SDK

aws-iot-device-sdk.js 套件可讓開發人員撰寫透過 WebSocket 通訊協定 AWS IoT 使用 MQTT 或 MQTT 存取的 JavaScript 應用程式。可用於 Node.js 環境和瀏覽器應用程式中。如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上適用於 JavaScript v2 的裝置 SDK
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK 範例
- AWS IoT 適用於 JavaScript v2 的 裝置 SDK API 文件

Java

AWS IoT 適用於 Java 的裝置 SDK

適用於 Java 的 AWS IoT 裝置開發套件可讓 Java 開發人員透過 MQTT 或 MQTT 透過 WebSocket 通訊協定存取 AWS IoT 平台。此 SDK 內建有影子支援。您可以使用 HTTP 方法來存取影子，包括 GET、UPDATE 與 DELETE。此 SDK 亦支援簡化的影子存取模式，開發人員只需使用 getter 和 setter 方法，即可與影子交換資料，而無需將任何 JSON 文件序列化或還原序列化。

> **Note**
>
> Device AWS IoT SDK for Java v2 現在支援 Android 開發。如需詳細資訊，請參閱適用於 AWS IoT Android 的 裝置開發套件。

如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上的適用於 Java 的 裝置 SDK v2
- AWS IoT 適用於 Java v2 的 裝置 SDK 讀我檔案
- AWS IoT 適用於 Java 的 裝置 SDK v2 範例
- AWS IoT 適用於 Java v2 的 裝置 SDK API 文件

Swift

AWS IoT 適用於 Swift 的裝置 SDK

適用於 Swift 的 AWS IoT 裝置開發套件可讓 Swift 開發人員使用 MQTT 5 通訊協定為 Linux 和 Apple macOS、iOS 和 tvOS 平台建立 AWS IoT 應用程式。

如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上適用於 Swift 的裝置 SDK
- AWS IoT 適用於 Swift Readme 的裝置 SDK
- AWS IoT 適用於 Swift 範例的裝置 SDK

AWS IoT 適用於 Embedded C 的 裝置 SDK

> **Note**
>
> 此 SDK 適合經驗豐富的嵌入式軟體開發人員使用。

適用於 Embedded C 的 AWS IoT Device SDK (C-SDK) 是 MIT 開放原始碼授權下的 C 來源檔案集合，可用於內嵌應用程式，以安全地將 IoT 裝置連接到其中 AWS IoT Core。它包含 MQTT 用戶端、JSON 剖析器，以及 AWS IoT Device Shadow、 AWS IoT Jobs、 AWS IoT Fleet Provisioning 和 AWS IoT Device Defender 程式庫。此 SDK 以來源形式分配，並且可與應用程式碼、其他程式庫及您選擇的作業系統 (OS) 作業系統一起內建於客戶韌體中。

通常以需要最佳化 C 語言執行時間的資源限制裝置 適用於 Embedded C 的 AWS IoT Device SDK 為目標。您可以在任何作業系統上使用軟體開發套件，並將其裝載在任何處理器類型 (例如 MCU 和 MPU) 上。

如需詳細資訊，請參閱下列內容：

- GitHub 上適用於 Embedded C 的AWS IoT 裝置 SDK
- AWS IoT 適用於 Embedded C 的 裝置 SDK 讀我檔案
- AWS IoT 適用於內嵌 C 的 裝置 SDK 範例

AWS 行動SDKs

AWS Mobile SDKs 為行動應用程式開發人員平台特定的 AWS IoT Core 服務 APIs、使用 MQTT 的 IoT 裝置通訊，以及其他 AWS 服務的 APIs 提供支援。

Android

適用於 Android 的 AWS Mobile SDK

適用於 Android 的 AWS Mobile SDK 包含程式庫、範例和文件，供開發人員使用 建置連線的行動應用程式 AWS。此 SDK 也包含對 MQTT 裝置通訊和呼叫 AWS IoT Core 服務的 APIs的支援。如需詳細資訊，請參閱下列內容：

- 適用於 Android 的 AWS Mobile SDK GitHub 上的
- 適用於 Android 的 AWS Mobile SDK 讀我檔案
- 適用於 Android 的 AWS Mobile SDK 範例
- 適用於 Android 的 AWS Mobile SDK API 參考
- AWSIoTClient 類別參考文件

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS 是開放原始碼軟體開發套件，根據 Apache Open Source 授權分佈。提供程式庫、程式碼範例和文件，以協助開發人員使用 AWS Mobile SDK for iOS 建置連線的行動應用程式 AWS。此 SDK 也包括支援 MQTT 裝置通訊和呼叫 AWS IoT Core 服務的 API。如需詳細資訊，請參閱下列內容：

- AWS Mobile SDK for iOS GitHub 上的
- AWS Mobile SDK for iOS 讀我檔案
- AWS Mobile SDK for iOS 範例
- 中的 AWSIoT 類別參考文件 AWS Mobile SDK for iOS

AWS IoT 裝置用戶端

AWS IoT Device Client 提供的程式碼可協助您的裝置連線 AWS IoT、執行機群佈建任務、支援裝置安全政策、使用安全通道連線，以及在裝置上處理任務。您可以在裝置上安裝此軟體，來處理這些例行裝置任務，以便可以專注於您的特定解決方案。

> **Note**
>
> AWS IoT Device Client 適用於採用 x86_64 或 ARM 處理器的微處理器型 IoT 裝置，以及常見的 Linux 作業系統。

C++

AWS IoT 裝置用戶端

如需 C++ AWS IoT 裝置用戶端的詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上 C++ 原始程式碼中的裝置用戶端
- AWS IoT C++ Readme 中的裝置用戶端

舊型 AWS IoT 裝置 SDKs版本

這些是 AWS IoT 裝置 SDKs 的較早版本，已由上述較新版本取代。這些 SDK 只接收維護和安全性更新。它們不會更新以包含新功能，而且不應該用於新專案。

- AWS IoT GitHub 上的 C++ 裝置 SDK
- AWS IoT C++ 裝置 SDK 讀我檔案
- AWS IoT GitHub 上適用於 Python v1 的裝置 SDK
- AWS IoT 適用於 Python v1 的 裝置 SDK 讀我檔案
- AWS IoT GitHub 上的適用於 Java 的裝置 SDK
- AWS IoT 適用於 Java 的 裝置開發套件讀我檔案
- AWS IoT GitHub 上適用於 JavaScript 的裝置 SDK
- AWS IoT 適用於 JavaScript 的 裝置 SDK 讀我檔案

AWS IoT 使用 AWS SDKs程式碼範例

下列程式碼範例示範如何使用 AWS IoT 搭配 AWS 軟體開發套件 (SDK)。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

程式碼範例

- AWS IoT 使用 AWS SDKs的基本範例
  - 您好 AWS IoT
  - AWS IoT 使用 AWS SDK 了解 的基本概念
  - AWS IoT 使用 AWS SDKs的動作
    - AttachThingPrincipal 搭配 AWS SDK 或 CLI 使用
    - CreateKeysAndCertificate 搭配 AWS SDK 或 CLI 使用
    - CreateThing 搭配 AWS SDK 或 CLI 使用
    - CreateTopicRule 搭配 AWS SDK 或 CLI 使用
    - DeleteCertificate 搭配 AWS SDK 或 CLI 使用
    - DeleteThing 搭配 AWS SDK 或 CLI 使用
    - DeleteTopicRule 搭配 AWS SDK 或 CLI 使用
    - DescribeEndpoint 搭配 AWS SDK 或 CLI 使用
    - DescribeThing 搭配 AWS SDK 或 CLI 使用
    - DetachThingPrincipal 搭配 AWS SDK 或 CLI 使用
    - ListCertificates 搭配 AWS SDK 或 CLI 使用
    - ListThings 搭配 AWS SDK 或 CLI 使用
    - SearchIndex 搭配 AWS SDK 或 CLI 使用
    - UpdateIndexingConfiguration 搭配 AWS SDK 或 CLI 使用
    - UpdateThing 搭配 AWS SDK 或 CLI 使用

- AWS IoT 使用 AWS SDKs案例
  - 開始使用物聯網裝置保護
  - 開始使用物聯網簡訊

AWS IoT 使用 AWS SDKs的基本範例

下列程式碼範例示範如何 AWS IoT 搭配 AWS SDKs 使用 的基本概念。

範例

- 您好 AWS IoT
- AWS IoT 使用 AWS SDK 了解 的基本概念
- AWS IoT 使用 AWS SDKs的動作
  - AttachThingPrincipal 搭配 AWS SDK 或 CLI 使用
  - CreateKeysAndCertificate 搭配 AWS SDK 或 CLI 使用
  - CreateThing 搭配 AWS SDK 或 CLI 使用
  - CreateTopicRule 搭配 AWS SDK 或 CLI 使用
  - DeleteCertificate 搭配 AWS SDK 或 CLI 使用
  - DeleteThing 搭配 AWS SDK 或 CLI 使用
  - DeleteTopicRule 搭配 AWS SDK 或 CLI 使用
  - DescribeEndpoint 搭配 AWS SDK 或 CLI 使用
  - DescribeThing 搭配 AWS SDK 或 CLI 使用
  - DetachThingPrincipal 搭配 AWS SDK 或 CLI 使用
  - ListCertificates 搭配 AWS SDK 或 CLI 使用
  - ListThings 搭配 AWS SDK 或 CLI 使用
  - SearchIndex 搭配 AWS SDK 或 CLI 使用
  - UpdateIndexingConfiguration 搭配 AWS SDK 或 CLI 使用
  - UpdateThing 搭配 AWS SDK 或 CLI 使用

您好 AWS IoT

下列程式碼範例示範如何開始使用 AWS IoT。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
```

```
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
```

```
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 listThings。

C++

適用於 C++ 的 SDK

CMakeLists.txt CMake 檔案的程式碼。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iot)

# Set this project's name.
project("hello_iot")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
```

```
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_iot.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_iot.cpp 來源檔案的程式碼。

```
#include <aws/core/Aws.h>
#include <aws/iot/IoTClient.h>
#include <aws/iot/model/ListThingsRequest.h>
#include <iostream>

/*
 *  A "Hello IoT" starter application which initializes an AWS IoT client and
 *  lists the AWS IoT topics in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_iot'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
```

```
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IoT::IoTClient iotClient(clientConfig);
        // List the things in the current account.
        Aws::IoT::Model::ListThingsRequest listThingsRequest;

        Aws::String nextToken; // Used for pagination.
        Aws::Vector<Aws::IoT::Model::ThingAttribute> allThings;

        do {
            if (!nextToken.empty()) {
                listThingsRequest.SetNextToken(nextToken);
            }

            Aws::IoT::Model::ListThingsOutcome listThingsOutcome =
iotClient.ListThings(
                    listThingsRequest);
            if (listThingsOutcome.IsSuccess()) {
                const Aws::Vector<Aws::IoT::Model::ThingAttribute> &things =
listThingsOutcome.GetResult().GetThings();
                allThings.insert(allThings.end(), things.begin(), things.end());
                nextToken = listThingsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "List things failed"
                          << listThingsOutcome.GetError().GetMessage() <<
std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << allThings.size() << " thing(s) found." << std::endl;
        for (auto const &thing: allThings) {
            std::cout << thing.GetThingName() << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
```

```
    return 0;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 listThings。

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotClient;
import software.amazon.awssdk.services.iot.model.ListThingsRequest;
import software.amazon.awssdk.services.iot.model.ListThingsResponse;
import software.amazon.awssdk.services.iot.model.ThingAttribute;
import software.amazon.awssdk.services.iot.paginators.ListThingsIterable;

import java.util.List;

public class HelloIoT {
    public static void main(String[] args) {
        System.out.println("Hello AWS IoT. Here is a listing of your AWS IoT
 Things:");
        IotClient iotClient = IotClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllThings(iotClient);
    }
```

```
    public static void listAllThings(IotClient iotClient) {
        iotClient.listThingsPaginator(ListThingsRequest.builder()
                .maxResults(10)
                .build())
            .stream()
            .flatMap(response -> response.things().stream())
            .forEach(attribute -> {
                System.out.println("Thing name: " + attribute.thingName());
                System.out.println("Thing ARN: " + attribute.thingArn());
            });
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 listThings。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.ListThingsRequest

suspend fun main() {
    println("A listing of your AWS IoT Things:")
    listAllThings()
}

suspend fun listAllThings() {
    val thingsRequest =
        ListThingsRequest {
            maxResults = 10
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
```

```
        val response = iotClient.listThings(thingsRequest)
        val thingList = response.things
        if (thingList != null) {
            for (attribute in thingList) {
                println("Thing name ${attribute.thingName}")
                println("Thing ARN: ${attribute.thingArn}")
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 listThings。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
def hello_iot():
    """
    Use the AWS SDK for Python (Boto3) to create an AWS IoT client and list
    up to 10 things in your AWS IoT account.
    This example uses the default settings specified in your shared credentials
    and config files.
    """
    try:
        iot_client = boto3.client("iot")
        response = iot_client.list_things(maxResults=10)
        things = response.get("things", [])

        print("Hello, AWS IoT! Here are your things:")
        if things:
            for i, thing in enumerate(things, 1):
                print(f"{i}. {thing['thingName']}")
        else:
            print("No things found in your AWS IoT account.")
```

```
    except ClientError as e:
        if e.response["Error"]["Code"] == "UnauthorizedException":
            print("You don't have permission to access AWS IoT.")
        else:
            print(f"Couldn't access AWS IoT. Error: {e}")
    except NoCredentialsError:
        print("No AWS credentials found. Please configure your credentials.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 listThings。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS IoT 使用 AWS SDK 了解 的基本概念

下列程式碼範例示範如何：

- 建立 AWS IoT 物件。
- 產生裝置憑證。
- 使用 屬性更新 AWS IoT 物件。
- 傳回唯一的端點。
- 列出您的 AWS IoT 憑證。
- 更新影 AWS IoT 子。
- 寫出狀態資訊。
- 建立規則。
- 列出您的規則。
- 使用 Thing 名稱搜尋物件。
- 刪除 AWS IoT 物件。

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行示範 AWS IoT 功能的互動式案例。

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/
basics/iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
```

```
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected
 dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";
```

```
        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in
the AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN
value is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
```

```
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing
the communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for
{thingName}? (y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
```

```
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
                    {
                        Console.WriteLine("Certificate attached to Thing
successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Failed to attach certificate to
Thing.");
                    }

                    Console.WriteLine("Thing Details:");
                    Console.WriteLine($"Thing Name: {thingName}");
                    Console.WriteLine($"Thing ARN: {thingArn}");
                }
                else
                {
                    Console.WriteLine("Failed to create certificate.");
                }
            }
            Console.WriteLine(new string('-', 80));

            // Step 4: Update an AWS IoT Thing with Attributes
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
```

```
            Console.WriteLine("IoT Thing attributes, represented as key-value
pairs, offer a pivotal advantage in facilitating efficient data");
            Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var attributes = new Dictionary<string, string>
            {
                { "Location", "Seattle" },
                { "DeviceType", "Sensor" },
                { "Firmware", "1.2.3" }
            };

            await iotWrapper.UpdateThingAsync(thingName, attributes);
            Console.WriteLine("Thing attributes updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 5: Return a unique endpoint specific to the Amazon Web
Services account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
```

```
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5
certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });
```

```
            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a
rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like
to deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for
default '{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();
```

```
                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS
topic: {snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM
role: {roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for
default '{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation
outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
```

```
                    }
                }
                else
                {
                    Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
            }
        }
        else
        {
            Console.WriteLine("Skipping CloudFormation stack deployment and
rule creation.");
        }
        Console.WriteLine(new string('-', 80));

        // Step 10: List your rules
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("10. List your rules.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var rules = await iotWrapper.ListTopicRulesAsync();
        Console.WriteLine("List of IoT Rules:");
        foreach (var rule in rules.Take(5)) // Show first 5 rules
        {
            Console.WriteLine($"Rule Name: {rule.RuleName}");
            Console.WriteLine($"Rule ARN: {rule.RuleArn}");
            Console.WriteLine("--------------");
        }
        Console.WriteLine();
        Console.WriteLine(new string('-', 80));

        // Step 11: Search things using the Thing name
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("11. Search things using the Thing name.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the
 certificate for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
 {thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
 certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
```

```
                    Console.ReadLine();
                }

                await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                await iotWrapper.DeleteCertificateAsync(certificateId);
                Console.WriteLine($"{certificateArn} was successfully
deleted.");
            }
            Console.WriteLine(new string('-', 80));
        }

        // Step 13: Delete the AWS IoT Thing
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("13. Delete the AWS IoT Thing.");
        var deleteThing = "y";
        if (IsInteractive)
        {
            Console.Write($"Do you want to delete the IoT Thing? (y/n)");
            deleteThing = Console.ReadLine();
        }
        else
        {
            Console.WriteLine($"Deleting IoT Thing {thingName}...");
        }

        if (deleteThing?.ToLower() == "y")
        {
            await iotWrapper.DeleteThingAsync(thingName);
            Console.WriteLine($"Deleted Thing {thingName}");
        }
        Console.WriteLine(new string('-', 80));

        // Step 14: Clean up CloudFormation stack
        if (!string.IsNullOrEmpty(snsTopicArn))
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("14. Clean up CloudFormation stack.");
            Console.WriteLine("Deleting the CloudFormation stack and all
resources...");
```

```
                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
 delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
 iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check
 the logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
```

```
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during
CloudFormation stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
```

```
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string>
{ Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await
WaitForStackCompletion(response.StackId, cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created
successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation
failed: {stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
```

```
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}'
already exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
```

```
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack
creation to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };
```

```
            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being
deleted. This may take a few minutes.");
```

```
            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };
```

```
            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to
be deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
```

```
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press
 Enter for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-
a-zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

SDK AWS IoT 方法的包裝函式類別。

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
```

```
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
```

```
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or
 null if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
 CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await
 _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
 {response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
 response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
```

```
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
```

```
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
```

```
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
```

```
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
 shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
 MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
 {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</
returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
```

```
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
 snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
```

```
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
```

```
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT
rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
```

```
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
```

```
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready
after configuration update");
            return new List<ThingDocument>();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
 why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
 why: {ex.Message}");
            return false;
        }
    }
```

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
```

```
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);
```

```
            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
 next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - AttachThingPrincipal
  - CreateKeysAndCertificate

- CreateThing
- CreateTopicRule
- DeleteCertificate
- DeleteThing
- DeleteTopicRule
- DescribeEndpoint
- DescribeThing
- DetachThingPrincipal
- ListCertificates
- ListThings
- SearchIndex
- UpdateIndexingConfiguration
- UpdateThing

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 AWS IoT 物件。

```
    Aws::String thingName = askQuestion("Enter a thing name: ");

    if (!createThing(thingName, clientConfiguration)) {
        std::cerr << "Exiting because createThing failed." << std::endl;
        cleanup("", "", "", "", "", false, clientConfiguration);
        return false;
    }
```

```
//! Create an AWS IoT thing.
```

```
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

產生和附加裝置憑證。

```
    Aws::String certificateARN;
    Aws::String certificateID;
    if (askYesNoQuestion("Would you like to create a certificate for your thing?
 (y/n) ")) {
        Aws::String outputFolder;
        if (askYesNoQuestion(
                "Would you like to save the certificate and keys to file? (y/n)
 ")) {
            outputFolder = std::filesystem::current_path();
            outputFolder += "/device_keys_and_certificates";

            std::filesystem::create_directories(outputFolder);

            std::cout << "The certificate and keys will be saved to the folder: "
                      << outputFolder << std::endl;
```

```
        }

        if (!createKeysAndCertificate(outputFolder, certificateARN,
 certificateID,
                                       clientConfiguration)) {
            std::cerr << "Exiting because createKeysAndCertificate failed."
                      << std::endl;
            cleanup(thingName, "", "", "", "", false, clientConfiguration);
            return false;
        }

        std::cout << "\nNext, the certificate will be attached to the thing.\n"
                  << std::endl;
        if (!attachThingPrincipal(certificateARN, thingName,
 clientConfiguration)) {
            std::cerr << "Exiting because attachThingPrincipal failed." <<
 std::endl;
            cleanup(thingName, certificateARN, certificateID, "", "",
                    false,
                    clientConfiguration);
            return false;
        }
    }
```

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
```

```
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
```

```
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
```

```
    else {
        std::cerr << "Failed to attach principal to thing." <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

對 AWS IoT 物件執行各種操作。

```
   if (!updateThing(thingName, { {"location", "Office"}, {"firmwareVersion",
 "v2.0"} }, clientConfiguration)) {
        std::cerr << "Exiting because updateThing failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
   }

   printAsterisksLine();

   std::cout << "Now an endpoint will be retrieved for your account.\n" <<
 std::endl;
   std::cout << "An IoT Endpoint refers to a specific URL or Uniform Resource
 Locator that serves as the entry point\n"
   << "for communication between IoT devices and the AWS IoT service." <<
 std::endl;

   askQuestion("Press Enter to continue:", alwaysTrueTest);

   Aws::String endpoint;
   if (!describeEndpoint(endpoint, clientConfiguration)) {
        std::cerr << "Exiting because getEndpoint failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
   }
   std::cout <<"Your endpoint is " << endpoint << "." << std::endl;
   printAsterisksLine();

   std::cout << "Now the certificates in your account will be listed." <<
 std::endl;
   askQuestion("Press Enter to continue:", alwaysTrueTest);
```

```
    if (!listCertificates(clientConfiguration)) {
        std::cerr << "Exiting because listCertificates failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now the shadow for the thing will be updated.\n" << std::endl;
    std::cout << "A thing shadow refers to a feature that enables you to create a
 virtual representation, or \"shadow,\"\n"
    << "of a physical device or thing. The thing shadow allows you to synchronize
 and control the state of a device between\n"
    << "the cloud and the device itself. and the AWS IoT service. For example,
 you can write and retrieve JSON data from a thing shadow." << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    if (!updateThingShadow(thingName, R"({"state":{"reported":
{"temperature":25,"humidity":50}}})", clientConfiguration)) {
        std::cerr << "Exiting because updateThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now, the state information for the shadow will be retrieved.\n"
 << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    Aws::String shadowState;
    if (!getThingShadow(thingName, shadowState, clientConfiguration)) {
        std::cerr << "Exiting because getThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
    std::cout << "The retrieved shadow state is: " << shadowState << std::endl;

    printAsterisksLine();
```

```
   std::cout << "A rule with now be added to to the thing.\n" << std::endl;
   std::cout << "Any user who has permission to create rules will be able to
access data processed by the rule." << std::endl;
   std::cout << "In this case, the rule will use an Simple Notification Service
(SNS) topic and an IAM rule." << std::endl;
   std::cout << "These resources will be created using a CloudFormation
template." << std::endl;
   std::cout << "Stack creation may take a few minutes." << std::endl;

   askQuestion("Press Enter to continue: ", alwaysTrueTest);
   Aws::Map<Aws::String, Aws::String> outputs
=createCloudFormationStack(STACK_NAME,clientConfiguration);
   if (outputs.empty()) {
       std::cerr << "Exiting because createCloudFormationStack failed." <<
std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }

   // Retrieve the topic ARN and role ARN from the CloudFormation stack outputs.
   auto topicArnIter = outputs.find(SNS_TOPIC_ARN_OUTPUT);
   auto roleArnIter = outputs.find(ROLE_ARN_OUTPUT);
   if ((topicArnIter == outputs.end()) || (roleArnIter == outputs.end())) {
       std::cerr << "Exiting because output '" << SNS_TOPIC_ARN_OUTPUT <<
       "' or '" << ROLE_ARN_OUTPUT << "'not found in the CloudFormation stack."
<< std::endl;
       cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
               false,
               clientConfiguration);
       return false;
   }

   Aws::String topicArn = topicArnIter->second;
   Aws::String roleArn = roleArnIter->second;
   Aws::String sqlStatement = "SELECT * FROM '";
   sqlStatement += MQTT_MESSAGE_TOPIC_FILTER;
   sqlStatement += "'";

   printAsterisksLine();

   std::cout << "Now a rule will be created.\n" << std::endl;
   std::cout << "Rules are an administrator-level action. Any user who has
permission\n"
```

```
                 << "to create rules will be able to access data processed by the
 rule." << std::endl;
    std::cout << "In this case, the rule will use an SNS topic" << std::endl;
    std::cout << "and the following SQL statement '" << sqlStatement << "'." <<
 std::endl;
    std::cout << "For more information on IoT SQL, see https://
docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html" <<
 std::endl;
    Aws::String ruleName = askQuestion("Enter a rule name: ");
    if (!createTopicRule(ruleName, topicArn, sqlStatement, roleArn,
 clientConfiguration)) {
        std::cerr << "Exiting because createRule failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now your rules will be listed.\n" << std::endl;
    askQuestion("Press Enter to continue: ", alwaysTrueTest);
    if (!listTopicRules(clientConfiguration)) {
        std::cerr << "Exiting because listRules failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();
    Aws::String queryString = "thingName:" + thingName;
    std::cout << "Now the AWS IoT fleet index will be queried with the query\n'"
    << queryString << "'.\n" << std::endl;
    std::cout << "For query information, see https://docs.aws.amazon.com/iot/
latest/developerguide/query-syntax.html" << std::endl;

    std::cout << "For this query to work, thing indexing must be enabled in your
 account.\n"
    << "This can be done with the awscli command line by calling 'aws iot update-
indexing-configuration'\n"
       << "or it can be done programmatically." << std::endl;
    std::cout << "For more information, see https://docs.aws.amazon.com/iot/
latest/developerguide/managing-index.html" << std::endl;
```

```
    if (askYesNoQuestion("Do you want to enable thing indexing in your account?
 (y/n) "))
    {
        Aws::IoT::Model::ThingIndexingConfiguration thingIndexingConfiguration;

 thingIndexingConfiguration.SetThingIndexingMode(Aws::IoT::Model::ThingIndexingMode::REGI

 thingIndexingConfiguration.SetThingConnectivityIndexingMode(Aws::IoT::Model::ThingConnec
        // The ThingGroupIndexingConfiguration object is ignored if not set.
        Aws::IoT::Model::ThingGroupIndexingConfiguration
 thingGroupIndexingConfiguration;
        if (!updateIndexingConfiguration(thingIndexingConfiguration,
 thingGroupIndexingConfiguration, clientConfiguration)) {
            std::cerr << "Exiting because updateIndexingConfiguration failed." <<
 std::endl;
            cleanup(thingName, certificateARN, certificateID, STACK_NAME,
                    ruleName, false,
                    clientConfiguration);
            return false;
        }
    }

    if (!searchIndex(queryString, clientConfiguration)) {

        std::cerr << "Exiting because searchIndex failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }
```

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
```

```
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
```

```
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}

//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
```

```
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}

//! Update the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param document: The state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThingShadow(const Aws::String &thingName,
                                    const Aws::String &document,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient
 iotDataPlaneClient(clientConfiguration);
    Aws::IoTDataPlane::Model::UpdateThingShadowRequest updateThingShadowRequest;
    updateThingShadowRequest.SetThingName(thingName);
    std::shared_ptr<std::stringstream> streamBuf =
 std::make_shared<std::stringstream>(
            document);
    updateThingShadowRequest.SetBody(streamBuf);
    Aws::IoTDataPlane::Model::UpdateThingShadowOutcome outcome =
 iotDataPlaneClient.UpdateThingShadow(
            updateThingShadowRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing shadow." << std::endl;
    }
    else {
        std::cerr << "Error while updating thing shadow."
                  << outcome.GetError().GetMessage() << std::endl;
```

```
    }

    return outcome.IsSuccess();
}

//! Get the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param documentResult: String to receive the state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::getThingShadow(const Aws::String &thingName,
                                 Aws::String &documentResult,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient iotClient(clientConfiguration);
    Aws::IoTDataPlane::Model::GetThingShadowRequest request;
    request.SetThingName(thingName);
    auto outcome = iotClient.GetThingShadow(request);
    if (outcome.IsSuccess()) {
        std::stringstream ss;
        ss << outcome.GetResult().GetPayload().rdbuf();
        documentResult = ss.str();
    }
    else {
        std::cerr << "Error getting thing shadow: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
```

```
                                const Aws::String &snsTopicARN, const Aws::String
 &sql,
                                const Aws::String &roleARN,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});

    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}

//! Lists the AWS IoT topic rules.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listTopicRules(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListTopicRulesRequest request;
```

```
    Aws::Vector<Aws::IoT::Model::TopicRuleListItem> allRules;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::ListTopicRulesOutcome outcome =
 iotClient.ListTopicRules(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListTopicRulesResult &result =
 outcome.GetResult();
            allRules.insert(allRules.end(),
                            result.GetRules().cbegin(),
                            result.GetRules().cend());

            nextToken = result.GetNextToken();
        }
        else {
            std::cerr << "ListTopicRules error: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << "ListTopicRules: " << allRules.size() << " rule(s) found."
              << std::endl;
    for (auto &rule: allRules) {
        std::cout << "  Rule name: " << rule.GetRuleName() << ", rule ARN: "
                  << rule.GetRuleArn() << "." << std::endl;
    }

    return true;
}

//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
```

```
}
```

清除資源。

```
bool
AwsDoc::IoT::cleanup(const Aws::String &thingName, const Aws::String
 &certificateARN,
                     const Aws::String &certificateID, const Aws::String
 &stackName,
                     const Aws::String &ruleName, bool askForConfirmation,
                     const Aws::Client::ClientConfiguration &clientConfiguration)
 {
    bool result = true;

    if (!ruleName.empty() && (!askForConfirmation ||
                              askYesNoQuestion("Delete the rule '" + ruleName +
                                               "'? (y/n) "))) {
        result &= deleteTopicRule(ruleName, clientConfiguration);
    }

    Aws::CloudFormation::CloudFormationClient
 cloudFormationClient(clientConfiguration);

    if (!stackName.empty() && (!askForConfirmation ||
                               askYesNoQuestion(
                                       "Delete the CloudFormation stack '" +
 stackName +
                                       "'? (y/n) "))) {
        result &= deleteStack(stackName, clientConfiguration);
    }

    if (!certificateARN.empty() && (!askForConfirmation ||
                                    askYesNoQuestion("Delete the certificate '" +
                                                     certificateARN + "'? (y/n)
 "))) {
        result &= detachThingPrincipal(certificateARN, thingName,
 clientConfiguration);
        result &= deleteCertificate(certificateID, clientConfiguration);
    }

    if (!thingName.empty() && (!askForConfirmation ||
```

```
                                askYesNoQuestion("Delete the thing '" + thingName
 +
                                                 "'? (y/n) "))) {
        result &= deleteThing(thingName, clientConfiguration);
    }

    return result;
}
```

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
```

```
}

//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);
```

```
    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的下列主題。
  - AttachThingPrincipal
  - CreateKeysAndCertificate

- CreateThing
- CreateTopicRule
- DeleteCertificate
- DeleteThing
- DeleteTopicRule
- DescribeEndpoint
- DescribeThing
- DetachThingPrincipal
- ListCertificates
- ListThings
- SearchIndex
- UpdateIndexingConfiguration
- UpdateThing

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行示範 AWS IoT 功能的互動式案例。

```
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 *
 * This Java example performs these tasks:
 *
 * 1. Creates an AWS IoT Thing.
 * 2. Generate and attach a device certificate.
 * 3. Update an AWS IoT Thing with Attributes.
 * 4. Get an AWS IoT Endpoint.
 * 5. List your certificates.
 * 6. Updates the shadow for the specified thing..
 * 7. Write out the state information, in JSON format
 * 8. Creates a rule
 * 9. List rules
 * 10. Search things
 * 11. Detach amd delete the certificate.
 * 12. Delete Thing.
 */
public class IotScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <roleARN> <snsAction>

                Where:
                    roleARN - The ARN of an IAM role that has permission to work
 with AWS IOT.
                    snsAction  - An ARN of an SNS topic.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        IotActions iotActions = new IotActions();
        String thingName;
        String ruleName;
        String roleARN = args[0];
        String snsAction = args[1];
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println(DASHES);
        System.out.println("Welcome to the AWS IoT basics scenario.");
        System.out.println("""
            This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service. The program guides you through a series
of steps,
            including creating an IoT Thing, generating a device certificate,
updating the Thing with attributes, and so on.
            It utilizes the AWS SDK for Java V2 and incorporates functionality
for creating and managing IoT Things, certificates, rules,
            shadows, and performing searches. The program aims to showcase AWS
IoT capabilities and provides a comprehensive example for
            developers working with AWS IoT in a Java environment.

            Let's get started...

            """);
        System.out.println(DASHES);

        System.out.println("1. Create an AWS IoT Thing.");
        System.out.println("""
            An AWS IoT Thing represents a virtual entity in the AWS IoT service
that can be associated with
            a physical device.
            """);
        // Prompt the user for input.
        System.out.print("Enter Thing name: ");
        thingName = scanner.nextLine();
        iotActions.createIoTThing(thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Generate a device certificate.");
        System.out.println("""
            A device certificate performs a role in securing the communication
between devices (Things)
            and the AWS IoT platform.
            """);

        System.out.print("Do you want to create a certificate for " +thingName
+"? (y/n)");
        String certAns = scanner.nextLine();
        String certificateArn="" ;
        if (certAns != null && certAns.trim().equalsIgnoreCase("y")) {
```

```
            certificateArn = iotActions.createCertificate();
            System.out.println("Attach the certificate to the AWS IoT Thing.");
            iotActions.attachCertificateToThing(thingName, certificateArn);
        } else {
            System.out.println("A device certificate was not created.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Update an AWS IoT Thing with Attributes.");
        System.out.println("""
             IoT Thing attributes, represented as key-value pairs, offer a
pivotal advantage in facilitating efficient data
             management and retrieval within the AWS IoT ecosystem.
            """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Return a unique endpoint specific to the Amazon
Web Services account.");
        System.out.println("""
            An IoT Endpoint refers to a specific URL or Uniform Resource Locator
that serves as the entry point for communication between IoT devices and the AWS
IoT service.
           """);
        waitForInputToContinue(scanner);
        String endpointUrl = iotActions.describeEndpoint();
        System.out.println("The endpoint is "+endpointUrl);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. List your AWS IoT certificates");
        waitForInputToContinue(scanner);
        if (certificateArn.length() > 0) {
            iotActions.listCertificates();
        } else {
            System.out.println("You did not create a certificates. Skipping this
step.");
        }
        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Create an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
        System.out.println("""
            A Thing Shadow refers to a feature that enables you to create a
virtual representation, or "shadow,"
            of a physical device or thing. The Thing Shadow allows you to
synchronize and control the state of a device between
            the cloud and the device itself. and the AWS IoT service. For
example, you can write and retrieve JSON data from a Thing Shadow.
           """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Write out the state information, in JSON
format.");
        waitForInputToContinue(scanner);
        iotActions.getPayload(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Creates a rule");
        System.out.println("""
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """);
        System.out.print("Enter Rule name: ");
        ruleName = scanner.nextLine();
        iotActions.createIoTRule(roleARN, ruleName, snsAction);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List your rules.");
        waitForInputToContinue(scanner);
        iotActions.listIoTRules();
        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Search things using the Thing name.");
        waitForInputToContinue(scanner);
        String queryString = "thingName:"+thingName ;
        iotActions.searchThings(queryString);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        if (certificateArn.length() > 0) {
            System.out.print("Do you want to detach and delete the certificate
for " +thingName +"? (y/n)");
            String delAns = scanner.nextLine();
            if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
                System.out.println("11. You selected to detach amd delete the
certificate.");
                waitForInputToContinue(scanner);
                iotActions.detachThingPrincipal(thingName, certificateArn);
                iotActions.deleteCertificate(certificateArn);
                waitForInputToContinue(scanner);
            } else {
                System.out.println("11. You selected not to delete the
certificate.");
            }
        } else {
            System.out.println("11. You did not create a certificate so there is
nothing to delete.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Delete the AWS IoT Thing.");
        System.out.print("Do you want to delete the IoT Thing? (y/n)");
        String delAns = scanner.nextLine();
        if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
            iotActions.deleteIoTThing(thingName);
        } else {
            System.out.println("The IoT Thing was not deleted.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("The AWS IoT workflow has successfully completed.");
        System.out.println(DASHES);
    }


    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            System.out.println("");
            System.out.println("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input.trim().equalsIgnoreCase("c")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }
    }
}
```

SDK AWS IoT 方法的包裝函式類別。

```
import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.core.retry.RetryPolicy;
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotAsyncClient;
import software.amazon.awssdk.services.iot.model.Action;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.Certificate;
import
 software.amazon.awssdk.services.iot.model.CreateKeysAndCertificateResponse;
import software.amazon.awssdk.services.iot.model.CreateThingRequest;
import software.amazon.awssdk.services.iot.model.CreateThingResponse;
```

```
import software.amazon.awssdk.services.iot.model.CreateTopicRuleRequest;
import software.amazon.awssdk.services.iot.model.CreateTopicRuleResponse;
import software.amazon.awssdk.services.iot.model.DeleteCertificateRequest;
import software.amazon.awssdk.services.iot.model.DeleteCertificateResponse;
import software.amazon.awssdk.services.iot.model.DeleteThingRequest;
import software.amazon.awssdk.services.iot.model.DeleteThingResponse;
import software.amazon.awssdk.services.iot.model.DescribeEndpointRequest;
import software.amazon.awssdk.services.iot.model.DescribeEndpointResponse;
import software.amazon.awssdk.services.iot.model.DescribeThingRequest;
import software.amazon.awssdk.services.iot.model.DescribeThingResponse;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.IotException;
import software.amazon.awssdk.services.iot.model.ListCertificatesResponse;
import software.amazon.awssdk.services.iot.model.ListTopicRulesRequest;
import software.amazon.awssdk.services.iot.model.ListTopicRulesResponse;
import software.amazon.awssdk.services.iot.model.SearchIndexRequest;
import software.amazon.awssdk.services.iot.model.SearchIndexResponse;
import software.amazon.awssdk.services.iot.model.TopicRuleListItem;
import software.amazon.awssdk.services.iot.model.SnsAction;
import software.amazon.awssdk.services.iot.model.TopicRulePayload;
import software.amazon.awssdk.services.iotdataplane.IotDataPlaneAsyncClient;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowRequest;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowResponse;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowRequest;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowResponse;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class IotActions {

    private static IotAsyncClient iotAsyncClient;

    private static IotDataPlaneAsyncClient iotAsyncDataPlaneClient;

    private static final String TOPIC = "your-iot-topic";
```

```
    private static IotDataPlaneAsyncClient getAsyncDataPlaneClient() {
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();

        if (iotAsyncDataPlaneClient == null) {
            iotAsyncDataPlaneClient = IotDataPlaneAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncDataPlaneClient;
    }


    private static IotAsyncClient getAsyncClient() {
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();
```

```
        if (iotAsyncClient == null) {
            iotAsyncClient = IotAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncClient;
    }

    /**
     * Creates an IoT certificate asynchronously.
     *
     * @return The ARN of the created certificate.
     * <p>
     * This method initiates an asynchronous request to create an IoT
certificate.
     * If the request is successful, it prints the certificate details and
returns the certificate ARN.
     * If an exception occurs, it prints the error message.
     */
    public String createCertificate() {
        CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
        final String[] certificateArn = {null};
        future.whenComplete((response, ex) -> {
            if (response != null) {
                String certificatePem = response.certificatePem();
                certificateArn[0] = response.certificateArn();

                // Print the details.
                System.out.println("\nCertificate:");
                System.out.println(certificatePem);
                System.out.println("\nCertificate ARN:");
                System.out.println(certificateArn[0]);

            } else {
                Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
```

```
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }

    /**
     * Creates an IoT Thing with the specified name asynchronously.
     *
     * @param thingName The name of the IoT Thing to create.
     *
     * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
     * If the request is successful, it prints the name of the thing and its ARN
value.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTThing(String thingName) {
        CreateThingRequest createThingRequest = CreateThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
        future.whenComplete((createThingResponse, ex) -> {
            if (createThingResponse != null) {
                System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Attaches a certificate to an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to attach.
     *
     * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
     * If the request is successful, it prints a confirmation message and
additional information about the Thing.
     * If an exception occurs, it prints the error message.
     */
    public void attachCertificateToThing(String thingName, String certificateArn)
{
        AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
            .thingName(thingName)
            .principal(certificateArn)
            .build();

        CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
        future.whenComplete((attachResponse, ex) -> {
            if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
                System.out.println("Certificate attached to Thing
successfully.");

                // Print additional information about the Thing.
                describeThing(thingName);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
```

```
                        attachResponse.sdkHttpResponse().statusCode());
                }
            }
        });

        future.join();
    }

    /**
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Updates the shadow of an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to update the shadow of an
IoT Thing.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void updateShadowThing(String thingName) {
        // Create Thing Shadow State Document.
        String stateDocument = "{\"state\":{\"reported\":{\"temperature\":25,
\"humidity\":50}}}";
        SdkBytes data = SdkBytes.fromString(stateDocument,
StandardCharsets.UTF_8);
        UpdateThingShadowRequest updateThingShadowRequest =
UpdateThingShadowRequest.builder()
            .thingName(thingName)
            .payload(data)
            .build();

        CompletableFuture<UpdateThingShadowResponse> future =
getAsyncDataPlaneClient().updateThingShadow(updateThingShadowRequest);
        future.whenComplete((updateResponse, ex) -> {
            if (updateResponse != null) {
                System.out.println("Thing Shadow updated successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to update Thing Shadow.");
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};

        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }
```

```
    /**
     * Extracts a specific value from the endpoint URL.
     *
     * @param input The endpoint URL to process.
     * @return The extracted value from the endpoint URL.
     */
    private static String getValue(String input) {
        // Define a regular expression pattern for extracting the subdomain.
        Pattern pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\
\.com");

        // Match the pattern against the input string.
        Matcher matcher = pattern.matcher(input);

        // Check if a match is found.
        if (matcher.find()) {
            // Extract the subdomain from the first capturing group.
            String subdomain = matcher.group(1);
            System.out.println("Extracted subdomain: " + subdomain);
            return subdomain ;
        } else {
            System.out.println("No match found");
        }
        return "" ;
    }

    /**
     * Lists all certificates asynchronously.
     *
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
     * If an exception occurs, it prints the error message.
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
 getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
```

```
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
    }

    /**
     * Retrieves the payload of a Thing's shadow asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to get the payload of a
Thing's shadow.
     * If the request is successful, it prints the shadow data.
     * If an exception occurs, it prints the error message.
     */
    public void getPayload(String thingName) {
        GetThingShadowRequest getThingShadowRequest =
GetThingShadowRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<GetThingShadowResponse> future =
getAsyncDataPlaneClient().getThingShadow(getThingShadowRequest);
        future.whenComplete((getThingShadowResponse, ex) -> {
            if (getThingShadowResponse != null) {
                // Extracting payload from response.
                SdkBytes payload = getThingShadowResponse.payload();
                String payloadString = payload.asUtf8String();
                System.out.println("Received Shadow Data: " + payloadString);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
```

```
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to get Thing Shadow payload.");
                }
            }
        });

        future.join();
    }

    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
            .roleArn(roleARN)
            .build();

        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();
```

```
        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });

        future.join();
    }

    /**
     * Lists IoT rules asynchronously.
     *
     * This method initiates an asynchronous request to list IoT rules.
     * If the request is successful, it prints the names and ARNs of the rules.
     * If an exception occurs, it prints the error message.
     */
    public void listIoTRules() {
        ListTopicRulesRequest listTopicRulesRequest =
ListTopicRulesRequest.builder().build();
        CompletableFuture<ListTopicRulesResponse> future =
getAsyncClient().listTopicRules(listTopicRulesRequest);
        future.whenComplete((listTopicRulesResponse, ex) -> {
            if (listTopicRulesResponse != null) {
                System.out.println("List of IoT Rules:");
```

```
                List<TopicRuleListItem> ruleList =
listTopicRulesResponse.rules();
                for (TopicRuleListItem rule : ruleList) {
                    System.out.println("Rule Name: " + rule.ruleName());
                    System.out.println("Rule ARN: " + rule.ruleArn());
                    System.out.println("--------------");
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list IoT Rules.");
                }
            }
        });

        future.join();
    }

    /**
     * Searches for IoT Things asynchronously based on a query string.
     *
     * @param queryString The query string to search for Things.
     *
     * This method initiates an asynchronous request to search for IoT Things.
     * If the request is successful and Things are found, it prints their IDs.
     * If no Things are found, it prints a message indicating so.
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
```

```
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }

    /**
     * Detaches a principal (certificate) from an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to detach.
     *
     * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
     * If the detachment is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void detachThingPrincipal(String thingName, String certificateArn) {
        DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
            .principal(certificateArn)
            .thingName(thingName)
            .build();

        CompletableFuture<DetachThingPrincipalResponse> future =
getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
        future.whenComplete((voidResult, ex) -> {
```

```
            if (ex == null) {
                System.out.println(certificateArn + " was successfully removed
from " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();

        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
```

```
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing to delete.
     *
     * This method initiates an asynchronous request to delete an IoT Thing.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteIoTThing(String thingName) {
        DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    // Get the cert Id  from the Cert ARN value.
    private String extractCertificateId(String certificateArn) {
        // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
```

```
        String[] arnParts = certificateArn.split(":");
        String certificateIdPart = arnParts[arnParts.length - 1];
        return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") +
 1);
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.Action
import aws.sdk.kotlin.services.iot.model.AttachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.AttributePayload
import aws.sdk.kotlin.services.iot.model.CreateThingRequest
import aws.sdk.kotlin.services.iot.model.CreateTopicRuleRequest
import aws.sdk.kotlin.services.iot.model.DeleteCertificateRequest
import aws.sdk.kotlin.services.iot.model.DeleteThingRequest
import aws.sdk.kotlin.services.iot.model.DescribeEndpointRequest
import aws.sdk.kotlin.services.iot.model.DescribeThingRequest
import aws.sdk.kotlin.services.iot.model.DetachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.ListTopicRulesRequest
import aws.sdk.kotlin.services.iot.model.SearchIndexRequest
import aws.sdk.kotlin.services.iot.model.SnsAction
import aws.sdk.kotlin.services.iot.model.TopicRulePayload
import aws.sdk.kotlin.services.iot.model.UpdateThingRequest
import aws.sdk.kotlin.services.iotdataplane.IotDataPlaneClient
import aws.sdk.kotlin.services.iotdataplane.model.GetThingShadowRequest
import aws.sdk.kotlin.services.iotdataplane.model.UpdateThingShadowRequest
import aws.smithy.kotlin.runtime.content.ByteStream
import aws.smithy.kotlin.runtime.content.toByteArray
import java.util.Scanner
import java.util.regex.Pattern
import kotlin.system.exitProcess

/**
 * Before running this Kotlin code example, ensure that your development
 environment
 * is set up, including configuring your credentials.
 *
 * For detailed instructions, refer to the following documentation topic:
 * [Setting Up Your Development Environment](https://docs.aws.amazon.com/sdk-for-
kotlin/latest/developer-guide/setup.html)
 *
 * This code example requires an SNS topic and an IAM Role.
 * Follow the steps in the documentation to set up these resources:
 *
 * - [Creating an SNS Topic](https://docs.aws.amazon.com/sns/latest/dg/sns-
getting-started.html#step-create-topic)
 * - [Creating an IAM Role](https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html)
 */
```

```
val DASHES = String(CharArray(80)).replace("\u0000", "-")
val TOPIC = "your-iot-topic"

suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage:
            <roleARN> <snsAction>

        Where:
            roleARN - The ARN of an IAM role that has permission to work with AWS
 IOT.
            snsAction  - An ARN of an SNS topic.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        exitProcess(1)
    }

    var thingName: String
    val roleARN = args[0]
    val snsAction = args[1]
    val scanner = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the AWS IoT example scenario.")
    println(
        """
        This example program demonstrates various interactions with the AWS
 Internet of Things (IoT) Core service.
        The program guides you through a series of steps, including creating an
 IoT thing, generating a device certificate,
        updating the thing with attributes, and so on.

        It utilizes the AWS SDK for Kotlin and incorporates functionality for
 creating and managing IoT things, certificates, rules,
        shadows, and performing searches. The program aims to showcase AWS IoT
 capabilities and provides a comprehensive example for
        developers working with AWS IoT in a Kotlin environment.
        """.trimIndent(),
    )
```

```
    print("Press Enter to continue...")
    scanner.nextLine()
    println(DASHES)

    println(DASHES)
    println("1. Create an AWS IoT thing.")
    println(
        """
        An AWS IoT thing represents a virtual entity in the AWS IoT service that
can be associated with a physical device.
        """.trimIndent(),
    )
    // Prompt the user for input.
    print("Enter thing name: ")
    thingName = scanner.nextLine()
    createIoTThing(thingName)
    describeThing(thingName)
    println(DASHES)

    println(DASHES)
    println("2. Generate a device certificate.")
    println(
        """
        A device certificate performs a role in securing the communication
between devices (things) and the AWS IoT platform.
        """.trimIndent(),
    )

    print("Do you want to create a certificate for $thingName? (y/n)")
    val certAns = scanner.nextLine()
    var certificateArn: String? = ""
    if (certAns != null && certAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
        certificateArn = createCertificate()
        println("Attach the certificate to the AWS IoT thing.")
        attachCertificateToThing(thingName, certificateArn)
    } else {
        println("A device certificate was not created.")
    }
    println(DASHES)

    println(DASHES)
    println("3. Update an AWS IoT thing with Attributes.")
    println(
```

```
        """
        IoT thing attributes, represented as key-value pairs, offer a pivotal
advantage in facilitating efficient data
        management and retrieval within the AWS IoT ecosystem.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateThing(thingName)
    println(DASHES)

    println(DASHES)
    println("4. Return a unique endpoint specific to the Amazon Web Services
account.")
    println(
        """
        An IoT Endpoint refers to a specific URL or Uniform Resource Locator that
serves as the entry point for communication between IoT devices and the AWS IoT
service.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    val endpointUrl = describeEndpoint()
    println(DASHES)

    println(DASHES)
    println("5. List your AWS IoT certificates")
    print("Press Enter to continue...")
    scanner.nextLine()
    if (certificateArn!!.isNotEmpty()) {
        listCertificates()
    } else {
        println("You did not create a certificates. Skipping this step.")
    }
    println(DASHES)

    println(DASHES)
    println("6. Create an IoT shadow that refers to a digital representation or
virtual twin of a physical IoT device")
    println(
        """
        A thing shadow refers to a feature that enables you to create a virtual
representation, or "shadow,"
```

```
        of a physical device or thing. The thing shadow allows you to synchronize
and control the state of a device between
        the cloud and the device itself. and the AWS IoT service. For example,
you can write and retrieve JSON data from a thing shadow.

        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateShawdowThing(thingName)
    println(DASHES)

    println(DASHES)
    println("7. Write out the state information, in JSON format.")
    print("Press Enter to continue...")
    scanner.nextLine()
    getPayload(thingName)
    println(DASHES)

    println(DASHES)
    println("8. Creates a rule")
    println(
        """
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """.trimIndent(),
    )
    print("Enter Rule name: ")
    val ruleName = scanner.nextLine()
    createIoTRule(roleARN, ruleName, snsAction)
    println(DASHES)

    println(DASHES)
    println("9. List your rules.")
    print("Press Enter to continue...")
    scanner.nextLine()
    listIoTRules()
    println(DASHES)

    println(DASHES)
    println("10. Search things using the name.")
    print("Press Enter to continue...")
    scanner.nextLine()
```

```
    val queryString = "thingName:$thingName"
    searchThings(queryString)
    println(DASHES)

    println(DASHES)
    if (certificateArn.length > 0) {
        print("Do you want to detach and delete the certificate for $thingName?
 (y/n)")
        val delAns = scanner.nextLine()
        if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
 true)) {
            println("11. You selected to detach amd delete the certificate.")
            print("Press Enter to continue...")
            scanner.nextLine()
            detachThingPrincipal(thingName, certificateArn)
            deleteCertificate(certificateArn)
        } else {
            println("11. You selected not to delete the certificate.")
        }
    } else {
        println("11. You did not create a certificate so there is nothing to
 delete.")
    }
    println(DASHES)

    println(DASHES)
    println("12. Delete the AWS IoT thing.")
    print("Do you want to delete the IoT thing? (y/n)")
    val delAns = scanner.nextLine()
    if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
 true)) {
        deleteIoTThing(thingName)
    } else {
        println("The IoT thing was not deleted.")
    }
    println(DASHES)

    println(DASHES)
    println("The AWS IoT workflow has successfully completed.")
    println(DASHES)
}

suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
```

```
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}

suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}

private fun extractCertificateId(certificateArn: String): String? {
    // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
    val arnParts = certificateArn.split(":".toRegex()).dropLastWhile
 { it.isEmpty() }.toTypedArray()
    val certificateIdPart = arnParts[arnParts.size - 1]
    return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") + 1)
}

suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}
```

```
suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}

suspend fun listIoTRules() {
    val listTopicRulesRequest = ListTopicRulesRequest {}

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val listTopicRulesResponse =
 iotClient.listTopicRules(listTopicRulesRequest)
        println("List of IoT rules:")
        val ruleList = listTopicRulesResponse.rules
        ruleList?.forEach { rule ->
            println("Rule name: ${rule.ruleName}")
            println("Rule ARN: ${rule.ruleArn}")
            println("--------------")
        }
    }
}

suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
```

```
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}

suspend fun getPayload(thingNameVal: String?) {
    val getThingShadowRequest =
        GetThingShadowRequest {
            thingName = thingNameVal
        }

    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        val getThingShadowResponse =
 iotPlaneClient.getThingShadow(getThingShadowRequest)
        val payload = getThingShadowResponse.payload
        val payloadString = payload?.let { java.lang.String(it, Charsets.UTF_8) }
        println("Received shadow data: $payloadString")
    }
}

suspend fun listCertificates() {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
```

```
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}

suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}

private fun getValue(input: String?): String {
    // Define a regular expression pattern for extracting the subdomain.
    val pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\\.com")

    // Match the pattern against the input string.
    val matcher = pattern.matcher(input)

    // Check if a match is found.
    if (matcher.find()) {
        val subdomain = matcher.group(1)
        println("Extracted subdomain: $subdomain")
        return subdomain
    } else {
        println("No match found")
    }
    return ""
}

suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
```

```
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}

suspend fun updateShawdowThing(thingNameVal: String?) {
    // Create the thing shadow state document.
    val stateDocument = "{\"state\":{\"reported\":{\"temperature\":25, \"humidity
\":50}}}"
    val byteStream: ByteStream = ByteStream.fromString(stateDocument)
    val byteArray: ByteArray = byteStream.toByteArray()

    val updateThingShadowRequest =
        UpdateThingShadowRequest {
            thingName = thingNameVal
            payload = byteArray
        }

    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        iotPlaneClient.updateThingShadow(updateThingShadowRequest)
        println("The thing shadow was updated successfully.")
    }
}

suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
```

```
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}

suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}

suspend fun createCertificate(): String? {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}

suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
```

```
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Kotlin API 參考》中的下列主題。
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立 IoT 包裝函式類別來管理操作。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)


    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
        :return: The name and ARN of the created thing.
        """
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
```

```
                logger.info("Thing %s already exists. Skipping creation.",
thing_name)
                return None
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise



    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.
```

```
        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response


    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]


    def list_certificates(self):
        """
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
```

```
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
            self.iot_client.update_certificate(
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
```

```
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
                return
            logger.error(
                "Couldn't delete certificate. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
        :param role_arn: The ARN of the IAM role.
        """
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def list_topic_rules(self):
        """
        Lists AWS IoT topic rules.

        :return: The list of topic rules.
        """
        try:
            rules = []
            paginator = self.iot_client.get_paginator("list_topic_rules")
            for page in paginator.paginate():
                rules.extend(page["rules"])
            logger.info("Retrieved %s topic rules.", len(rules))
            return rules
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list topic rules. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise



    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response.get("things", [])


    def update_indexing_configuration(self):
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise


    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def update_thing_shadow(self, thing_name, shadow_state):
        """
        Updates the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param shadow_state: The shadow state as a dictionary.
        """
        import json
        try:
            self.iot_data_client.update_thing_shadow(
                thingName=thing_name, payload=json.dumps(shadow_state)
            )
            logger.info("Updated shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot update thing shadow. Resource not found.")
                return
            logger.error(
                "Couldn't update thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def get_thing_shadow(self, thing_name):
        """
        Gets the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :return: The shadow state as a dictionary.
        """
        import json
        try:
            response =
 self.iot_data_client.get_thing_shadow(thingName=thing_name)
            shadow = json.loads(response["payload"].read())
            logger.info("Retrieved shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot get thing shadow. Resource not found.")
                return None
            logger.error(
                "Couldn't get thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return shadow
```

執行示範 IoT 基本概念的互動式案例。

```
class IoTScenario:
    """Runs an interactive scenario that shows how to use AWS IoT."""

    is_interactive = True

    def __init__(self, iot_wrapper, iot_data_client, cfn_client,
 stack_name="IoTBasicsStack", template_path=None):
        """
        :param iot_wrapper: An instance of the IoTWrapper class.
        :param iot_data_client: A Boto3 IoT Data Plane client.
        :param cfn_client: A Boto3 CloudFormation client.
```

```
        :param stack_name: Name for the CloudFormation stack.
        :param template_path: Path to the CloudFormation template file.
        """
        self.iot_wrapper = iot_wrapper
        self.iot_data_client = iot_data_client
        self.cfn_client = cfn_client
        self.thing_name = None
        self.certificate_arn = None
        self.certificate_id = None
        self.rule_name = None
        self.stack_name = stack_name
        self.template_path = template_path or "../../../scenarios/basics/iot/
iot_usecase/resources/cfn_template.yaml"

    def _deploy_stack(self):
        """Deploy CloudFormation stack and return outputs."""
        with open(self.template_path, "r") as f:
            template_body = f.read()

        try:
            self.cfn_client.create_stack(
                StackName=self.stack_name,
                TemplateBody=template_body,
                Capabilities=["CAPABILITY_NAMED_IAM"]
            )

            waiter = self.cfn_client.get_waiter("stack_create_complete")
            waiter.wait(StackName=self.stack_name)

            response = self.cfn_client.describe_stacks(StackName=self.stack_name)
            outputs = {output["OutputKey"]: output["OutputValue"]
                       for output in response["Stacks"][0]["Outputs"]}
            return outputs["SNSTopicArn"], outputs["RoleArn"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "AlreadyExistsException":
                response =
 self.cfn_client.describe_stacks(StackName=self.stack_name)
                outputs = {output["OutputKey"]: output["OutputValue"]
                           for output in response["Stacks"][0]["Outputs"]}
                return outputs["SNSTopicArn"], outputs["RoleArn"]
            raise

    def _cleanup_stack(self):
        """Delete CloudFormation stack."""
```

```
        try:
            self.cfn_client.delete_stack(StackName=self.stack_name)
            waiter = self.cfn_client.get_waiter("stack_delete_complete")
            waiter.wait(StackName=self.stack_name)
            print("CloudFormation stack deleted successfully.")
        except ClientError as err:
            logger.error(f"Failed to delete stack: {err}")

    def run_scenario(self, thing_name, rule_name):
        """
        Runs the IoT basics scenario.

        :param thing_name: The name of the thing to create.
        :param rule_name: The name of the topic rule to create.
        """
        print("-" * 88)
        print("Welcome to the AWS IoT basics scenario!")
        print("-" * 88)
        print(
            "This scenario demonstrates how to interact with AWS IoT using the
AWS SDK for Python (Boto3).\n"
            "AWS IoT provides secure, bi-directional communication between
Internet-connected devices\n"
            "and the AWS cloud. You can manage device connections, process device
data, and build IoT applications.\n"
        )

        self.thing_name = thing_name
        self.rule_name = rule_name

        try:
            print("\nDeploying CloudFormation stack...")
            sns_topic_arn, role_arn = self._deploy_stack()
            print(f"Stack deployed. SNS Topic: {sns_topic_arn}")

            input("\nNext, we'll create an AWS IoT thing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("1. Create an AWS IoT thing")
            print("-" * 88)
            response = self.iot_wrapper.create_thing(thing_name)
            print(f"Created thing: {response['thingName']}")
            print(f"Thing ARN: {response['thingArn']}")
```

```
            input("\nNext, we'll list things. Press Enter to continue...") if
self.is_interactive else None
            print("\n" + "-" * 88)
            print("2. List things")
            print("-" * 88)
            things = self.iot_wrapper.list_things()
            print(f"Found {len(things)} thing(s) in your account")
            for thing in things[:5]:  # Show first 5
                print(f"  Thing name: {thing['thingName']}")

            input("\nNext, we'll generate a device certificate. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("3. Generate a device certificate")
            print("-" * 88)
            cert_response = self.iot_wrapper.create_keys_and_certificate()
            self.certificate_arn = cert_response["certificateArn"]
            self.certificate_id = cert_response["certificateId"]
            print(f"Created certificate: {self.certificate_id}")

            input("\nNext, we'll attach the certificate to the thing. Press Enter
to continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("4. Attach the certificate to the thing")
            print("-" * 88)
            self.iot_wrapper.attach_thing_principal(thing_name,
self.certificate_arn)
            print(f"Attached certificate to thing: {thing_name}")

            input("\nNext, we'll update the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("5. Update the thing shadow")
            print("-" * 88)
            shadow_state = {"state": {"reported": {"temperature": 25, "humidity":
50}}}
            self.iot_wrapper.update_thing_shadow(thing_name, shadow_state)
            print(f"Updated shadow for thing: {thing_name}")

            input("\nNext, we'll get the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("6. Get the thing shadow")
            print("-" * 88)
```

```
            shadow = self.iot_wrapper.get_thing_shadow(thing_name)
            print(f"Shadow state: {json.dumps(shadow['state'], indent=2)}")

            input("\nNext, we'll get the AWS IoT endpoint. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("7. Get the AWS IoT endpoint")
            print("-" * 88)
            endpoint = self.iot_wrapper.describe_endpoint()
            print(f"IoT endpoint: {endpoint}")

            input("\nNext, we'll list certificates. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("8. List certificates")
            print("-" * 88)
            certificates = self.iot_wrapper.list_certificates()
            print(f"Found {len(certificates)} certificate(s)")
            for cert in certificates:
                print(f"  Certificate ID: {cert['certificateId']}")
                print(f"  Certificate ARN: {cert['certificateArn']}")
                print(f"  Status: {cert['status']}")

            input("\nNext, we'll create a topic rule. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("9. Create a topic rule")
            print("-" * 88)
            self.iot_wrapper.create_topic_rule(
                rule_name, f"device/{thing_name}/data", sns_topic_arn, role_arn
            )
            print(f"Created topic rule: {rule_name}")

            input("\nNext, we'll list topic rules. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("10. List topic rules")
            print("-" * 88)
            rules = self.iot_wrapper.list_topic_rules()
            print(f"Found {len(rules)} topic rule(s)")
            for rule in rules:
                print(f"  Rule name: {rule['ruleName']}")
                print(f"  Rule ARN: {rule['ruleArn']}")
```

```
            input("\nNext, we'll configure thing indexing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("11. Configure thing indexing")
            print("-" * 88)
            self.iot_wrapper.update_indexing_configuration()
            print("Enabled thing indexing")
            print("Waiting for indexing to be ready...")
            time.sleep(10)

            input("\nNext, we'll search for things. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("12. Search for things")
            print("-" * 88)
            try:
                things = self.iot_wrapper.search_index(f"thingName:{thing_name}")
                if things:
                    print(f"Found {len(things)} thing(s) matching the query")
                    for thing in things:
                        print(f"  Thing name: {thing.get('thingName', 'N/A')}")
                        print(f"  Thing ID: {thing.get('thingId', 'N/A')}")
                else:
                    print("No things found. Indexing may take a few minutes.")
            except ClientError as err:
                if err.response["Error"]["Code"] in [
                    "IndexNotReadyException",
                    "InvalidRequestException",
                ]:
                    print("Search index not ready yet. This is expected.")
                else:
                    raise

        except ClientError as err:
            logger.error(
                "Scenario failed: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        finally:
            self._cleanup()

    def _cleanup(self):
```

```
        """Cleans up resources created during the scenario."""
        if not self.thing_name:
            return

        print("\n" + "-" * 88)
        print("Cleanup")
        print("-" * 88)

        if q.ask("Do you want to delete the resources? (y/n) ", q.is_yesno):
            try:
                if self.certificate_arn:
                    print(f"Detaching certificate from thing: {self.thing_name}")
                    self.iot_wrapper.detach_thing_principal(
                        self.thing_name, self.certificate_arn
                    )

                if self.certificate_id:
                    print(f"Deleting certificate: {self.certificate_id}")
                    self.iot_wrapper.delete_certificate(self.certificate_id)

                if self.thing_name:
                    print(f"Deleting thing: {self.thing_name}")
                    self.iot_wrapper.delete_thing(self.thing_name)

                if self.rule_name:
                    print(f"Deleting topic rule: {self.rule_name}")
                    self.iot_wrapper.delete_topic_rule(self.rule_name)

                self._cleanup_stack()
                print("Resources deleted successfully.")
            except ClientError as err:
                logger.error(
                    "Cleanup failed: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
        else:
            print("Resources will remain in your account.")

        print("\n" + "-" * 88)
        print("Thanks for using AWS IoT!")
        print("-" * 88)
```

- 如需 API 詳細資訊，請參閱《適用於 Python (Boto3) 的AWS SDK API 參考》中的下列主題。
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS IoT 使用 AWS SDKs的動作

下列程式碼範例示範如何使用 AWS SDKs執行個別 AWS IoT 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

這些摘錄會呼叫 AWS IoT API，是必須在內容中執行之大型程式的程式碼摘錄。您可以在 AWS IoT 使用 AWS SDKs案例 中查看內容中的動作。

下列範例僅包含最常使用的動作。如需完整清單，請參閱《AWS IoT API 參考》。

範例

- AttachThingPrincipal 搭配 AWS SDK 或 CLI 使用

- CreateKeysAndCertificate 搭配 AWS SDK 或 CLI 使用
- CreateThing 搭配 AWS SDK 或 CLI 使用
- CreateTopicRule 搭配 AWS SDK 或 CLI 使用
- DeleteCertificate 搭配 AWS SDK 或 CLI 使用
- DeleteThing 搭配 AWS SDK 或 CLI 使用
- DeleteTopicRule 搭配 AWS SDK 或 CLI 使用
- DescribeEndpoint 搭配 AWS SDK 或 CLI 使用
- DescribeThing 搭配 AWS SDK 或 CLI 使用
- DetachThingPrincipal 搭配 AWS SDK 或 CLI 使用
- ListCertificates 搭配 AWS SDK 或 CLI 使用
- ListThings 搭配 AWS SDK 或 CLI 使用
- SearchIndex 搭配 AWS SDK 或 CLI 使用
- UpdateIndexingConfiguration 搭配 AWS SDK 或 CLI 使用
- UpdateThing 搭配 AWS SDK 或 CLI 使用

`AttachThingPrincipal` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `AttachThingPrincipal`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 AttachThingPrincipal。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
    else {
        std::cerr << "Failed to attach principal to thing." <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 AttachThingPrincipal。

CLI

AWS CLI

將憑證連接到物件

下列 `attach-thing-principal` 範例會將憑證連接至 MyTemperatureSensor 物件。憑證由 ARN 識別。您可以在 AWS IoT 主控台中找到憑證的 ARN。

```
aws iot attach-thing-principal \
    --thing-name MyTemperatureSensor \
    --principal arn:aws:iot:us-
west-2:123456789012:cert/2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8
```

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 AttachThingPrincipal。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Attaches a certificate to an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to attach.
     *
     * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
     * If the request is successful, it prints a confirmation message and
additional information about the Thing.
     * If an exception occurs, it prints the error message.
```

```
     */
    public void attachCertificateToThing(String thingName, String certificateArn)
{
        AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
            .thingName(thingName)
            .principal(certificateArn)
            .build();

        CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
        future.whenComplete((attachResponse, ex) -> {
            if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
                System.out.println("Certificate attached to Thing
successfully.");

                // Print additional information about the Thing.
                describeThing(thingName);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
                        attachResponse.sdkHttpResponse().statusCode());
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 AttachThingPrincipal。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 AttachThingPrincipal。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
 thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 AttachThingPrincipal。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
        lo_iot->attachthingprincipal(
          iv_thingname = iv_thing_name
          iv_principal = iv_principal ).
        MESSAGE |Principal attached to IoT thing '{ iv_thing_name }'.| TYPE 'I'.
      CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
        MESSAGE |Resource not found when attaching principal to
'{ iv_thing_name }'.| TYPE 'I'.
        RAISE EXCEPTION lo_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
        MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_svc_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 AttachThingPrincipal。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateKeysAndCertificate` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `CreateKeysAndCertificate`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   /// <summary>
   /// Creates a device certificate for AWS IoT.
   /// </summary>
   /// <returns>The certificate details including ARN and certificate PEM, or
null if creation failed.</returns>
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
           var request = new CreateKeysAndCertificateRequest
           {
               SetAsActive = true
           };

           var response = await
_amazonIoT.CreateKeysAndCertificateAsync(request);
           _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
           return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
       {
```

```
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
 {ex.Message}");
            return null;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateKeysAndCertificate。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
```

```
                                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
```

```
                            << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                            << "'."
                            << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateKeysAndCertificate。

CLI

AWS CLI

建立 RSA 金鑰對並發行 X.509 憑證

下列 `create-keys-and-certificate` 會使用已發行的公開金鑰，建立 2048 位元 RSA 金鑰對，並發行 X.509 憑證。由於這是 AWS IoT 為此憑證提供私有金鑰的唯一時間，請務必將其保存在安全的位置。

```
aws iot create-keys-and-certificate \
    --certificate-pem-outfile "myTest.cert.pem" \
```

```
    --public-key-outfile "myTest.public.key" \
    --private-key-outfile "myTest.private.key"
```

輸出：

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\nMMEXAMPLEuuN/
dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y
+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQ
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB
\n-----END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omittted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

如需詳細資訊，請參閱 AWS IoT 開發人員指南中的建立和註冊 物聯網裝置憑證。 AWS IoT

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateKeysAndCertificate。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   /**
    * Creates an IoT certificate asynchronously.
    *
    * @return The ARN of the created certificate.
    * <p>
    * This method initiates an asynchronous request to create an IoT
certificate.
    * If the request is successful, it prints the certificate details and
returns the certificate ARN.
    * If an exception occurs, it prints the error message.
    */
   public String createCertificate() {
       CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
       final String[] certificateArn = {null};
       future.whenComplete((response, ex) -> {
           if (response != null) {
               String certificatePem = response.certificatePem();
               certificateArn[0] = response.certificateArn();

               // Print the details.
               System.out.println("\nCertificate:");
               System.out.println(certificatePem);
               System.out.println("\nCertificate ARN:");
               System.out.println(certificateArn[0]);

           } else {
               Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
```

```
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateKeysAndCertificate。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createCertificate(): String? {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateKeysAndCertificate。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.

        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
 self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 CreateKeysAndCertificate。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   TRY.
       oo_result = lo_iot->createkeysandcertificate( iv_setasactive =
abap_true ).
       MESSAGE |Certificate created: { oo_result->get_certificateid( ) }| TYPE
'I'.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
       MESSAGE lo_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_ex.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 CreateKeysAndCertificate。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateThing` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `CreateThing`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };
```

```
            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
 {ex.Message}");
            return null;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateThing。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateThing。

CLI

AWS CLI

範例 1：在登錄檔中建立物件記錄

下列create-thing範例會在 AWS IoT 物件登錄檔中為裝置建立項目。

```
aws iot create-thing \
    --thing-name SampleIoTThing
```

輸出：

```
{
    "thingName": "SampleIoTThing",
    "thingArn": "arn:aws:iot:us-west-2: 123456789012:thing/SampleIoTThing",
    "thingId": " EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE "
}
```

範例 2：定義與物件類型相關聯的物件

下列 create-thing 範例會建立具有指定物件類型及其屬性的物件。

```
aws iot create-thing \
    --thing-name "MyLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

輸出：

```
{
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797"
}
```

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件，和物件類型。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateThing。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Creates an IoT Thing with the specified name asynchronously.
     *
     * @param thingName The name of the IoT Thing to create.
     *
     * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
     * If the request is successful, it prints the name of the thing and its ARN
value.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTThing(String thingName) {
        CreateThingRequest createThingRequest = CreateThingRequest.builder()
```

```
            .thingName(thingName)
            .build();

        CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
        future.whenComplete((createThingResponse, ex) -> {
            if (createThingResponse != null) {
                System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateThing。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }
```

```
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateThing。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
```

```
        :return: The name and ARN of the created thing.
        """
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Thing %s already exists. Skipping creation.",
thing_name)
                return None
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 CreateThing。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
  CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
  DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
  DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
  TRY.
      oo_result = lo_iot->creatething(
        iv_thingname = iv_thing_name ).
```

```
        MESSAGE |IoT thing created: { oo_result->get_thingname( ) } ARN:
{ oo_result->get_thingarn( ) }| TYPE 'I'.
      CATCH /aws1/cx_iotresrcalrdyexistsex.
        MESSAGE |IoT thing '{ iv_thing_name }' already exists.| TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 CreateThing。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`CreateTopicRule` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `CreateTopicRule`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
```

```
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
 snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
 {ex.Message}");
```

```
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateTopicRule。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);
```

```
    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});

    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 CreateTopicRule。

CLI

AWS CLI

建立傳送 Amazon SNS 提醒的規則

下列 `create-topic-rule` 範例會建立一個規則，就是當裝置影子中找到的土壤濕度讀數較低，傳送 Amazon SNS 訊息。

```
aws iot create-topic-rule \
    --rule-name "LowMoistureRule" \
    --topic-rule-payload file://plant-rule.json
```

此範例需要將下列 JSON 程式碼儲存至名為 `plant-rule.json` 的檔案：

```
{
    "sql": "SELECT * FROM '$aws/things/MyRPi/shadow/update/accepted' WHERE
 state.reported.moisture = 'low'\n",
```

```
    "description": "Sends an alert whenever soil moisture level readings are too
 low.",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
            "sns": {
                "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyRPiLowMoistureTopic",
                "roleArn": "arn:aws:iam::123456789012:role/service-role/
MyRPiLowMoistureTopicRole",
                "messageFormat": "RAW"
            }
    }]
}
```

此命令不會產生輸出。

如需詳細資訊，請參閱 AWS IoT 開發人員指南中的建立 IoT 規則。 AWS IoT

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateTopicRule。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
 actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
```

```
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
            .roleArn(roleARN)
            .build();

        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();

        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });
```

```
        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateTopicRule。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
```

```
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 CreateTopicRule。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
```

```
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
        :param role_arn: The ARN of the IAM role.
        """
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 CreateTopicRule。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    " Build the SNS action that will receive messages matching the rule.
    DATA lo_sns_action TYPE REF TO /aws1/cl_iotsnsaction.
    CREATE OBJECT lo_sns_action
      EXPORTING
        iv_targetarn = iv_sns_action_arn
        iv_rolearn   = iv_role_arn.

    DATA lo_action TYPE REF TO /aws1/cl_iotaction.
    CREATE OBJECT lo_action
      EXPORTING
        io_sns = lo_sns_action.

    DATA lt_actions TYPE /aws1/cl_iotaction=>tt_actionlist.
    APPEND lo_action TO lt_actions.

    " iv_topic = 'my/iot/topic' - The MQTT topic pattern to match
    DATA lo_payload TYPE REF TO /aws1/cl_iottopicrulepayload.
    CREATE OBJECT lo_payload
      EXPORTING
        iv_sql     = |SELECT * FROM '{ iv_topic }'|
        it_actions = lt_actions.

    lo_iot->createtopicrule(
      iv_rulename         = iv_rule_name
      io_topicrulepayload = lo_payload ).
    MESSAGE |IoT topic rule created: { iv_rule_name }| TYPE 'I'.
  CATCH /aws1/cx_iotresrcalrdyexistsex.
    MESSAGE |Topic rule '{ iv_rule_name }' already exists.| TYPE 'I'.
```

```
    CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
      MESSAGE lo_ex->get_text( ) TYPE 'I'.
      RAISE EXCEPTION lo_ex.
  ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 CreateTopicRule。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteCertificate` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteCertificate`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
```

```
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteCertificate。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteCertificate。

CLI

AWS CLI

刪除裝置憑證

下列 delete-certificate 範例會刪除具有指定 ID 的裝置憑證。

```
aws iot delete-certificate \
    --certificate-
id c0c57bbc8baaf4631a9a0345c957657f5e710473e3ddbee1428d216d54d53ac9
```

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT API 參考》中的 DeleteCertificate。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteCertificate。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
```

```
            .build();

        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteCertificate。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
```

```
        println("$certificateArn was successfully deleted.")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteCertificate。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
            self.iot_client.update_certificate(
```

```
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
                return
            logger.error(
                "Couldn't delete certificate. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 DeleteCertificate。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    " Certificates must be deactivated before they can be deleted.
    lo_iot->updatecertificate(
      iv_certificateid = iv_certificate_id
      iv_newstatus     = 'INACTIVE' ).
    lo_iot->deletecertificate( iv_certificateid = iv_certificate_id ).
    MESSAGE |Certificate deleted: { iv_certificate_id }| TYPE 'I'.
  CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
```

```
        MESSAGE |Certificate '{ iv_certificate_id }' not found.| TYPE 'I'.
        RAISE EXCEPTION lo_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
        MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_svc_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DeleteCertificate。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteThing` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteThing`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
```

```
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteThing。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
```

```
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteThing。

CLI

AWS CLI

顯示詳細的物件資訊

下列delete-thing範例會從您 AWS 帳戶的 AWS IoT 登錄檔中刪除物件。

aws iot delete-thing --thing-name "FourthBulb"

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteThing。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Deletes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing to delete.
     *
     * This method initiates an asynchronous request to delete an IoT Thing.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteIoTThing(String thingName) {
        DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteThing。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DeleteThing。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 DeleteThing。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    lo_iot->deletething( iv_thingname = iv_thing_name ).
    MESSAGE |IoT thing deleted: { iv_thing_name }| TYPE 'I'.
  CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
    MESSAGE |IoT thing '{ iv_thing_name }' not found.| TYPE 'I'.
    RAISE EXCEPTION lo_ex.
  CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
    MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
    RAISE EXCEPTION lo_svc_ex.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DeleteThing。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteTopicRule` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteTopicRule`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

C++

SDK for C++

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DeleteTopicRule。

CLI

AWS CLI

刪除規則

以下 delete-topic-rule 範例會刪除指定規則。

```
aws iot delete-topic-rule \
    --rule-name "LowMoistureRule"
```

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的刪除規則。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteTopicRule。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
```

```
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 DeleteTopicRule。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    lo_iot->deletetopicrule( iv_rulename = iv_rule_name ).
    MESSAGE |IoT topic rule deleted: { iv_rule_name }| TYPE 'I'.
```

```
    CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
      MESSAGE lo_ex->get_text( ) TYPE 'I'.
      RAISE EXCEPTION lo_ex.
  ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DeleteTopicRule。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeEndpoint` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeEndpoint`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <summary>
/// Gets the AWS IoT endpoint URL.
/// </summary>
/// <returns>The endpoint URL, or null if retrieval failed.</returns>
public async Task<string?> DescribeEndpointAsync()
{
    try
```

```
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeEndpoint。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Describe the endpoint specific to the AWS account making the call.
/*!
```

```
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeEndpoint。

CLI

AWS CLI

範例 1：取得您目前的 AWS 端點

下列`describe-endpoint`範例會擷取套用所有命令的預設 AWS 端點。

```
aws iot describe-endpoint
```

輸出：

```
{
    "endpointAddress": "abc123defghijk.iot.us-west-2.amazonaws.com"
}
```

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 DescribeEndpoint。

範例 2：取得您的 ATS 端點

下列 `describe-endpoint` 範例會擷取 Amazon Trust Services (ATS) 端點。

```
aws iot describe-endpoint \
    --endpoint-type iot:Data-ATS
```

輸出：

```
{
    "endpointAddress": "abc123defghijk-ats.iot.us-west-2.amazonaws.com"
}
```

如需詳細資訊，請參閱 AWS IoT 開發人員指南中的 X.509 憑證和 AWS IoT。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeEndpoint。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
```

```
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};

        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeEndpoint。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的 AWS SDK API 參考》中的 DescribeEndpoint。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""
```

```
    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 DescribeEndpoint。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn show_address(client: &Client, endpoint_type: &str) -> Result<(), Error>
 {
    let resp = client
        .describe_endpoint()
        .endpoint_type(endpoint_type)
        .send()
        .await?;

    println!("Endpoint address: {}", resp.endpoint_address.unwrap());

    println!();

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 Rust 的AWS SDK API 參考》中的 DescribeEndpoint。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
```

```
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    " iv_endpoint_type = 'iot:Data-ATS' - Endpoint type for data operations
    TRY.
        DATA(lo_result) = lo_iot->describeendpoint( iv_endpointtype =
iv_endpoint_type ).
        ov_endpoint_address = lo_result->get_endpointaddress( ).
        MESSAGE |Endpoint address: { ov_endpoint_address }| TYPE 'I'.
      CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
        MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
        RAISE EXCEPTION lo_throttle_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DescribeEndpoint。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeThing` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeThing`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

C++

SDK for C++

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Describe an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeThing(const Aws::String &thingName,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DescribeThingRequest request;
    request.SetThingName(thingName);

    Aws::IoT::Model::DescribeThingOutcome outcome =
 iotClient.DescribeThing(request);

    if (outcome.IsSuccess()) {
        const Aws::IoT::Model::DescribeThingResult &result = outcome.GetResult();
        std::cout << "Retrieved thing '" << result.GetThingName() << "'" <<
 std::endl;
        std::cout << "thingArn: " << result.GetThingArn() << std::endl;
        std::cout << result.GetAttributes().size() << " attribute(s) retrieved"
                  << std::endl;
        for (const auto &attribute: result.GetAttributes()) {
            std::cout << "  attribute: " << attribute.first << "=" <<
 attribute.second
                      << std::endl;
        }
    }
    else {
        std::cerr << "Error describing thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DescribeThing。

CLI

AWS CLI

顯示詳細的物件資訊

下列`describe-thing`範例顯示 AWS 您的帳戶的 AWS IoT 登錄檔中定義的物件 （裝置） 的相關資訊。

aws iot describe-thing --thing-name "MyLightBulb"

輸出：

```
{
    "defaultClientId": "MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    },
    "version": 1
}
```

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeThing。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
```

```
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DescribeThing。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 DescribeThing。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DetachThingPrincipal` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DetachThingPrincipal`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念
- 開始使用物聯網簡訊

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
 why: {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DetachThingPrincipal。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);
```

```
    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 DetachThingPrincipal。

CLI

AWS CLI

將憑證/主體從物件分開

下列 detach-thing-principal 範例會從指定的物件移除代表主體的憑證。

```
aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
```

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DetachThingPrincipal。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   /**
    * Detaches a principal (certificate) from an IoT Thing asynchronously.
    *
    * @param thingName The name of the IoT Thing.
    * @param certificateArn The ARN of the certificate to detach.
    *
    * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
    * If the detachment is successful, it prints a confirmation message.
    * If an exception occurs, it prints the error message.
    */
   public void detachThingPrincipal(String thingName, String certificateArn) {
       DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
           .principal(certificateArn)
           .thingName(thingName)
           .build();

       CompletableFuture<DetachThingPrincipalResponse> future =
getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
       future.whenComplete((voidResult, ex) -> {
           if (ex == null) {
               System.out.println(certificateArn + " was successfully removed
from " + thingName);
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else {
                   System.err.println("Unexpected error: " + ex.getMessage());
               }
```

```
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DetachThingPrincipal。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS 適用於 Kotlin 的 SDK API 參考》中的 DetachThingPrincipal。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
 thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
```

```
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 DetachThingPrincipal。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   TRY.
       lo_iot->detachthingprincipal(
         iv_thingname = iv_thing_name
         iv_principal = iv_principal ).
       MESSAGE |Principal detached from IoT thing '{ iv_thing_name }'.| TYPE
'I'.
     CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
       MESSAGE |Resource not found when detaching principal from
'{ iv_thing_name }'.| TYPE 'I'.
       RAISE EXCEPTION lo_ex.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
       MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_svc_ex.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 DetachThingPrincipal。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListCertificates` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `ListCertificates`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);
```

```
            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ListCertificates。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
```

```
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 ListCertificates。

CLI

AWS CLI

範例 1：列出在您 AWS 帳戶中註冊的憑證

下列 `list-certificates` 範例列出您帳戶中註冊的所有憑證。如果您有超過預設分頁限制 25，您可以使用此命令的 `nextMarker` 回應值，並將其提供給下一個命令以取得下一批結果。重複此動作，直到 `nextMarker` 傳回的不包含任何值為止。

```
aws iot list-certificates
```

輸出：

```
{
    "certificates": [
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
            "certificateId":
 "604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36",
            "status": "ACTIVE",
            "creationDate": 1556810537.617
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e
            "certificateId":
 "262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e",
            "status": "ACTIVE",
            "creationDate": 1546447050.885
        },
        {
            "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "certificateId":
 "b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "status": "ACTIVE",
            "creationDate": 1546292258.322
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42
```

```
            "certificateId":
 "7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42",
            "status": "ACTIVE",
            "creationDate": 1541457693.453
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3
            "certificateId":
 "54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3",
            "status": "ACTIVE",
            "creationDate": 1541113568.611
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e
            "certificateId":
 "4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e",
            "status": "ACTIVE",
            "creationDate": 1541022751.983
        }
    ]
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListCertificates。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Lists all certificates asynchronously.
     *
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
     * If an exception occurs, it prints the error message.
```

```
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 ListCertificates。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun listCertificates() {
```

```
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 ListCertificates。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def list_certificates(self):
        """
```

```
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 ListCertificates。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
```

```
        " Collect all certificates by following the pagination marker.
        DATA lv_marker TYPE /aws1/iotmarker.
        DATA lv_count  TYPE i.

        DO.
          oo_result = lo_iot->listcertificates( iv_marker = lv_marker ).
          lv_count  = lv_count + lines( oo_result->get_certificates( ) ).
          lv_marker = oo_result->get_nextmarker( ).
          IF lv_marker IS INITIAL.
            EXIT.
          ENDIF.
        ENDDO.

        MESSAGE |Retrieved { lv_count } IoT certificates.| TYPE 'I'.
      CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
        MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
        RAISE EXCEPTION lo_throttle_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 ListCertificates。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListThings` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `ListThings`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
```

```
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListThings。

CLI

AWS CLI

範例 1：列出登錄檔中的所有物件

下列`list-things`範例列出在您 AWS 帳戶的 AWS IoT 登錄檔中定義的物件（裝置）。

```
aws iot list-things
```

輸出：

```
{
    "things": [
        {
            "thingName": "ThirdBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/ThirdBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
```

```
            },
            "version": 2
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        },
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
        "thingName": "SampleIoTThing",
        "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/SampleIoTThing",
        "attributes": {},
        "version": 1
        }
    ]
}
```

範例 2：列出具有特定屬性的定義物件

下列 `list-things` 範例顯示具有名為 `wattage` 之屬性的物件清單。

```
aws iot list-things \
    --attribute-name wattage
```

輸出：

```
{
```

```
    "things": [
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        }
    ]
}
```

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的如何使用登錄檔管理物件。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListThings。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""
```

```
    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 ListThings。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn show_things(client: &Client) -> Result<(), Error> {
    let resp = client.list_things().send().await?;

    println!("Things:");

    for thing in resp.things.unwrap() {
        println!(
            "  Name:  {}",
            thing.thing_name.as_deref().unwrap_or_default()
        );
        println!(
            "  Type:  {}",
            thing.thing_type_name.as_deref().unwrap_or_default()
        );
        println!(
            "  ARN:   {}",
            thing.thing_arn.as_deref().unwrap_or_default()
        );
        println!();
    }

    println!();

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 ListThings。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    " Collect all things by following the pagination token.
    DATA lv_nexttoken TYPE /aws1/iotnexttoken.
    DATA lv_count     TYPE i.

    DO.
      oo_result    = lo_iot->listthings( iv_nexttoken = lv_nexttoken ).
      lv_count     = lv_count + lines( oo_result->get_things( ) ).
      lv_nexttoken = oo_result->get_nexttoken( ).
      IF lv_nexttoken IS INITIAL.
        EXIT.
      ENDIF.
    ENDDO.

    MESSAGE |Retrieved { lv_count } IoT things.| TYPE 'I'.
  CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
    MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
    RAISE EXCEPTION lo_throttle_ex.
  CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
    MESSAGE lo_ex->get_text( ) TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 ListThings。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`SearchIndex` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `SearchIndex`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
```

```
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
```

```
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
```

```
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready
after configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 SearchIndex。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
```

```
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
```

```
                  << std::endl;
    }
    return true;
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 SearchIndex。

CLI

AWS CLI

查詢物件索引

下列 `search-index` 範例會查詢 `AWS_Things` 索引中是否有類型為 `LightBulb` 的物件。

```
aws iot search-index \
    --index-name "AWS_Things" \
    --query-string "thingTypeName:LightBulb"
```

輸出：

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
            "thingId": "40da2e73-c6af-406e-b415-15acae538797",
            "thingTypeName": "LightBulb",
            "thingGroupNames": [
                "LightBulbs",
                "DeadBulbs"
            ],
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "ThirdBulb",
            "thingId": "615c8455-33d5-40e8-95fd-3ee8b24490af",
```

```
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingId": "6dae0d3f-40c1-476a-80c4-1ed24ba6aa11",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        }
    ]
}
```

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的管理物件索引。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 SearchIndex。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /**
     * Searches for IoT Things asynchronously based on a query string.
     *
     * @param queryString The query string to search for Things.
```

```
     *
     * This method initiates an asynchronous request to search for IoT Things.
     * If the request is successful and Things are found, it prints their IDs.
     * If no Things are found, it prints a message indicating so.
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 SearchIndex。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 SearchIndex。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response.get("things", [])
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 SearchIndex。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   " iv_query_string = 'thingName:MyThing*' - Fleet indexing query string
   TRY.
       oo_result = lo_iot->searchindex( iv_querystring = iv_query_string ).
       MESSAGE |Found { lines( oo_result->get_things( ) ) } IoT things matching
query.| TYPE 'I'.
     CATCH /aws1/cx_iotindexnotreadyex INTO DATA(lo_idx_ex).
       MESSAGE 'Fleet indexing is not ready. Enable indexing with
UpdateIndexingConfiguration first.' TYPE 'I'.
       RAISE EXCEPTION lo_idx_ex.
     CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
       MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
       RAISE EXCEPTION lo_throttle_ex.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
       MESSAGE lo_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_ex.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 SearchIndex。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`UpdateIndexingConfiguration` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 UpdateIndexingConfiguration。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

C++

SDK for C++

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Update the indexing configuration.
/*!
  \param thingIndexingConfiguration: A ThingIndexingConfiguration object which is
 ignored if not set.
  \param thingGroupIndexingConfiguration: A ThingGroupIndexingConfiguration
 object which is ignored if not set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateIndexingConfiguration(
        const Aws::IoT::Model::ThingIndexingConfiguration
 &thingIndexingConfiguration,
        const Aws::IoT::Model::ThingGroupIndexingConfiguration
 &thingGroupIndexingConfiguration,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::UpdateIndexingConfigurationRequest request;

    if (thingIndexingConfiguration.ThingIndexingModeHasBeenSet()) {
        request.SetThingIndexingConfiguration(thingIndexingConfiguration);
    }
```

```
    if (thingGroupIndexingConfiguration.ThingGroupIndexingModeHasBeenSet()) {

 request.SetThingGroupIndexingConfiguration(thingGroupIndexingConfiguration);
    }

    Aws::IoT::Model::UpdateIndexingConfigurationOutcome outcome =
 iotClient.UpdateIndexingConfiguration(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "UpdateIndexingConfiguration succeeded." << std::endl;
    }
    else {
        std::cerr << "UpdateIndexingConfiguration failed."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 UpdateIndexingConfiguration。

CLI

AWS CLI

啟用物件索引

下列update-indexing-configuration範例可讓物件索引支援使用 AWS_Things 索引搜尋登錄檔資料、影子資料和物件連線狀態。

```
aws iot update-indexing-configuration
    --thing-indexing-
configuration thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS
```

此命令不會產生輸出。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的管理物件索引。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateIndexingConfiguration。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def update_indexing_configuration(self):
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 UpdateIndexingConfiguration。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
        DATA lo_idx_conf TYPE REF TO /aws1/cl_iotthingindexingconf.
        CREATE OBJECT lo_idx_conf
          EXPORTING
            iv_thingindexingmode = 'REGISTRY'.
        lo_iot->updateindexingconfiguration(
          io_thingindexingconf = lo_idx_conf ).
        MESSAGE 'IoT thing indexing configuration updated to REGISTRY mode.' TYPE
'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 UpdateIndexingConfiguration。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`UpdateThing` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 UpdateThing。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
```

```
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 UpdateThing。

C++

適用於 C++ 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 UpdateThing。

CLI

AWS CLI

將物件類型與物件類型相關聯

下列update-thing範例會將 AWS IoT 登錄檔中的物件與物件類型建立關聯。當您建立關聯時，您可以為物件類型定義的屬性提供值。

```
aws iot update-thing \
    --thing-name "MyOtherLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

此命令未產生輸出。使用 describe-thing 命令查看結果：

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的物件類型。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 UpdateThing。

Kotlin

適用於 Kotlin 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Kotlin 的AWS SDK API 參考》中的 UpdateThing。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS IoT 使用 AWS SDKs案例

下列程式碼範例示範如何在 AWS SDKs中 AWS IoT 實作常見案例。這些案例說明如何透過在 內呼叫多個函數 AWS IoT 或與其他函數結合，來完成特定任務 AWS 服務。每個案例均包含完整原始碼的連結，您可在連結中找到如何設定和執行程式碼的相關指示。

案例的目標是獲得中等水平的經驗，協助您了解內容中的服務動作。

範例

- 開始使用物聯網裝置保護
- 開始使用物聯網簡訊

開始使用物聯網裝置保護

以下程式碼範例顯示做法：

- 建立必要的 IAM 角色
- 啟用 IoT Device Defender 稽核檢查
- 執行隨需稽核
- 建立緩解動作
- 將緩解動作套用至問題清單
- 設定 SNS 通知（選用）
- 啟用 IoT 記錄

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在範例開發人員教學課程儲存庫中設定和執行。

```
#!/bin/bash
```

```
# AWS IoT Device Defender Getting Started Script
# This script demonstrates how to use AWS IoT Device Defender to enable audit
 checks,
# view audit results, create mitigation actions, and apply them to findings.

set -euo pipefail

# Set up logging
LOG_FILE="iot-device-defender-script-$(date +%Y%m%d%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
echo "AWS IoT Device Defender Getting Started Script"
echo "================================================="
echo "Starting script execution at $(date)"
echo ""

# Function to check for errors in command output
check_error() {
    if echo "$1" | grep -iE "An error occurred|Exception|Failed|usage: aws" > /
dev/null; then
        echo "ERROR: Command failed with the following output:"
        echo "$1"
        return 1
    fi
    return 0
}

# Function to safely extract JSON values using jq
extract_json_value() {
    local json="$1"
    local key="$2"
    echo "$json" | jq -r ".${key} // empty" 2>/dev/null || echo ""
}

# Function to validate JSON
validate_json() {
    local json="$1"
    echo "$json" | jq empty 2>/dev/null
}

# Function to check AWS CLI availability
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
```

```
        echo "ERROR: AWS CLI is not installed or not in PATH"
        return 1
    fi
    if ! command -v jq &> /dev/null; then
        echo "ERROR: jq is not installed or not in PATH"
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
    local account_id
    account_id=$(aws sts get-caller-identity --query 'Account' --output text 2>/
dev/null) || true
    if [ -z "$account_id" ]; then
        echo "ERROR: Could not retrieve AWS account ID"
        return 1
    fi
    echo "$account_id"
    return 0
}

# Function to create IAM roles with retry logic
create_iam_role() {
    local ROLE_NAME=$1
    local TRUST_POLICY=$2
    local MANAGED_POLICY=$3
    local RETRY_COUNT=0
    local MAX_RETRIES=3

    echo "Creating IAM role: $ROLE_NAME"

    # Validate trust policy JSON
    if ! validate_json "$TRUST_POLICY"; then
        echo "ERROR: Invalid trust policy JSON for role $ROLE_NAME"
        return 1
    fi

    # Check if role already exists
    if aws iam get-role --role-name "$ROLE_NAME" >/dev/null 2>&1; then
        echo "Role $ROLE_NAME already exists, skipping creation"
        ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn'
 --output text 2>/dev/null) || true
```

```
        if [ -z "$ROLE_ARN" ]; then
            echo "ERROR: Could not retrieve ARN for existing role $ROLE_NAME"
            return 1
        fi
        echo "Role ARN: $ROLE_ARN"
        return 0
    fi

    # Create the role with trust policy and retry logic
    while [ $RETRY_COUNT -lt $MAX_RETRIES ]; do
        ROLE_RESULT=$(aws iam create-role \
            --role-name "$ROLE_NAME" \
            --assume-role-policy-document "$TRUST_POLICY" 2>&1) || true

        if check_error "$ROLE_RESULT"; then
            break
        fi

        RETRY_COUNT=$((RETRY_COUNT + 1))
        if [ $RETRY_COUNT -lt $MAX_RETRIES ]; then
            echo "Retrying role creation (attempt $((RETRY_COUNT + 1))/
$MAX_RETRIES)..."
            sleep $((RETRY_COUNT * 2))
        fi
    done

    if ! check_error "$ROLE_RESULT"; then
        echo "Failed to create role $ROLE_NAME after $MAX_RETRIES attempts"
        return 1
    fi

    aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true

    # For IoT logging role, create an inline policy instead of using a managed
 policy
    if [[ "$ROLE_NAME" == "AWSIoTLoggingRole" ]]; then
        local LOGGING_POLICY
        LOGGING_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
}
EOF
)

        if ! validate_json "$LOGGING_POLICY"; then
            echo "ERROR: Invalid logging policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$LOGGING_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    elif [[ "$ROLE_NAME" == "IoTMitigationActionErrorLoggingRole" ]]; then
        local MITIGATION_POLICY
        MITIGATION_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:SetV2LoggingOptions",
                "iot:SetLoggingOptions",
                "iot:AddThingToThingGroup"
```

```
            ],
            "Resource": "arn:aws:iot:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        }
    ]
}
EOF
)

        if ! validate_json "$MITIGATION_POLICY"; then
            echo "ERROR: Invalid mitigation policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$MITIGATION_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    else
        # Attach managed policy to role if provided
        if [ -n "$MANAGED_POLICY" ]; then
            ATTACH_RESULT=$(aws iam attach-role-policy \
                --role-name "$ROLE_NAME" \
                --policy-arn "$MANAGED_POLICY" 2>&1) || true

            if ! check_error "$ATTACH_RESULT"; then
                echo "Failed to attach policy to role $ROLE_NAME"
                return 1
            fi
        fi
```

```
    fi

    echo "Role $ROLE_NAME created successfully"

    # Get the role ARN with error handling
    ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn' --
output text 2>/dev/null) || true
    if [ -z "$ROLE_ARN" ]; then
        echo "ERROR: Could not retrieve ARN for newly created role $ROLE_NAME"
        return 1
    fi
    echo "Role ARN: $ROLE_ARN"
    return 0
}

# Array to store created resources for cleanup
declare -a CREATED_RESOURCES

# Validate prerequisites
echo "Validating prerequisites..."
if ! check_aws_cli; then
    echo "ERROR: Prerequisites not met"
    exit 1
fi

ACCOUNT_ID=$(get_account_id) || exit 1
echo "AWS Account ID: $ACCOUNT_ID"
echo ""

# Step 1: Create IAM roles needed for the tutorial
echo "=================================================="
echo "Step 1: Creating required IAM roles"
echo "=================================================="

# Create IoT Device Defender Audit role
IOT_DEFENDER_AUDIT_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "AWSIoTDeviceDefenderAuditRole"
 "$IOT_DEFENDER_AUDIT_TRUST_POLICY" "arn:aws:iam::aws:policy/service-role/
AWSIoTDeviceDefenderAudit"; then
    echo "ERROR: Failed to create audit role"
    exit 1
fi
AUDIT_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTDeviceDefenderAuditRole")

# Create IoT Logging role
IOT_LOGGING_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "AWSIoTLoggingRole" "$IOT_LOGGING_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create logging role"
    exit 1
fi
LOGGING_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTLoggingRole")

# Create IoT Mitigation Action role
IOT_MITIGATION_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "IoTMitigationActionErrorLoggingRole"
 "$IOT_MITIGATION_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create mitigation role"
    exit 1
fi
MITIGATION_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: IoTMitigationActionErrorLoggingRole")

# Wait for IAM role propagation
echo "Waiting for IAM role propagation..."
sleep 5

# Step 2: Enable audit checks
echo ""
echo "================================================="
echo "Step 2: Enabling AWS IoT Device Defender audit checks"
echo "================================================="

# Get current audit configuration
echo "Getting current audit configuration..."
CURRENT_CONFIG=$(aws iot describe-account-audit-configuration --output json 2>&1)
 || true
if validate_json "$CURRENT_CONFIG"; then
    echo "$CURRENT_CONFIG" | jq '.' 2>/dev/null || echo "Could not parse current
 configuration"
fi

# Enable specific audit checks with proper JSON escaping
echo "Enabling audit checks..."
AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":true}}'
```

```
if ! validate_json "$AUDIT_CONFIG"; then
    echo "ERROR: Invalid audit configuration JSON"
    exit 1
fi

UPDATE_RESULT=$(aws iot update-account-audit-configuration \
  --role-arn "$AUDIT_ROLE_ARN" \
  --audit-check-configurations "$AUDIT_CONFIG" 2>&1) || true

if ! check_error "$UPDATE_RESULT"; then
    echo "Failed to update audit configuration"
    exit 1
fi

echo "Audit checks enabled successfully"

# Step 3: Run an on-demand audit
echo ""
echo "=================================================="
echo "Step 3: Running an on-demand audit"
echo "=================================================="

echo "Starting on-demand audit task..."
AUDIT_TASK_RESULT=$(aws iot start-on-demand-audit-task \
  --target-check-names LOGGING_DISABLED_CHECK --output json 2>&1) || true

if ! check_error "$AUDIT_TASK_RESULT"; then
    echo "Failed to start on-demand audit task"
    exit 1
fi

TASK_ID=$(extract_json_value "$AUDIT_TASK_RESULT" "taskId")
if [ -z "$TASK_ID" ]; then
    echo "ERROR: Could not extract task ID from response"
    exit 1
fi

echo "Audit task started with ID: $TASK_ID"
CREATED_RESOURCES+=("Audit Task: $TASK_ID")

# Tag the audit task via IoT service
aws iot tag-resource --resource-arn "arn:aws:iot:$(aws configure get region):
${ACCOUNT_ID}:audittask/${TASK_ID}" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true
```

```
# Wait for the audit task to complete
echo "Waiting for audit task to complete (this may take a few minutes)..."
TASK_STATUS="IN_PROGRESS"
TIMEOUT=0
MAX_TIMEOUT=600
POLL_INTERVAL=15

while [ "$TASK_STATUS" != "COMPLETED" ]; do
    if [ $TIMEOUT -ge $MAX_TIMEOUT ]; then
        echo "WARNING: Audit task did not complete within ${MAX_TIMEOUT} seconds,
 continuing..."
        break
    fi

    sleep "$POLL_INTERVAL"
    TIMEOUT=$((TIMEOUT + POLL_INTERVAL))

    TASK_DETAILS=$(aws iot describe-audit-task --task-id "$TASK_ID" --output json
 2>&1) || true
    if validate_json "$TASK_DETAILS"; then
        TASK_STATUS=$(extract_json_value "$TASK_DETAILS" "taskStatus")
        echo "Current task status: $TASK_STATUS (elapsed: ${TIMEOUT}s)"

        if [ "$TASK_STATUS" = "FAILED" ]; then
            echo "WARNING: Audit task failed, continuing with script..."
            FAILURE_REASON=$(extract_json_value "$TASK_DETAILS"
 "taskStatistics.failedChecksNotApplicable")
            if [ -n "$FAILURE_REASON" ]; then
                echo "Reason: $FAILURE_REASON"
            fi
            break
        fi
    else
        echo "WARNING: Could not parse task details, retrying..."
    fi
done

echo "Audit task processing completed"

# Get audit findings (non-blocking)
echo "Getting audit findings..."
FINDINGS=$(aws iot list-audit-findings \
  --task-id "$TASK_ID" --output json 2>&1) || true
```

```
if validate_json "$FINDINGS"; then
    FINDING_COUNT=$(echo "$FINDINGS" | jq '.findings | length' 2>/dev/null ||
 echo "0")
    echo "Audit findings count: $FINDING_COUNT"
    if [ "$FINDING_COUNT" -gt 0 ]; then
        echo "Sample finding:"
        echo "$FINDINGS" | jq '.findings[0]' 2>/dev/null || echo "Could not parse
 finding"
    fi
else
    echo "WARNING: Could not parse audit findings response"
    FINDINGS='{"findings":[]}'
fi

# Check if we have any non-compliant findings
FINDING_ID=$(extract_json_value "$FINDINGS" "findings[0].findingId")
if [ -n "$FINDING_ID" ]; then
    echo "Found non-compliant finding with ID: $FINDING_ID"
    HAS_FINDINGS=true
else
    echo "No non-compliant findings detected"
    HAS_FINDINGS=false
fi

# Step 4: Create a mitigation action
echo ""
echo "================================================="
echo "Step 4: Creating a mitigation action"
echo "================================================="

# Check if mitigation action already exists and delete it
if aws iot describe-mitigation-action --action-name "EnableErrorLoggingAction" >/
dev/null 2>&1; then
    echo "Mitigation action 'EnableErrorLoggingAction' already exists, deleting
 it first..."
    aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction"
 2>&1 || true
    sleep 2
fi

echo "Creating mitigation action to enable AWS IoT logging..."

# Build mitigation action parameters JSON
```

```
MITIGATION_PARAMS=$(cat <<EOF
{
  "enableIoTLoggingParams": {
    "roleArnForLogging": "$LOGGING_ROLE_ARN",
    "logLevel": "ERROR"
  }
}
EOF
)

if ! validate_json "$MITIGATION_PARAMS"; then
    echo "ERROR: Invalid mitigation parameters JSON"
    exit 1
fi

MITIGATION_RESULT=$(aws iot create-mitigation-action \
  --action-name "EnableErrorLoggingAction" \
  --role-arn "$MITIGATION_ROLE_ARN" \
  --action-params "$MITIGATION_PARAMS" --output json 2>&1) || true

if ! check_error "$MITIGATION_RESULT"; then
    echo "Failed to create mitigation action"
    exit 1
fi

if validate_json "$MITIGATION_RESULT"; then
    echo "Mitigation action created successfully"
    MITIGATION_ACTION_ARN=$(extract_json_value "$MITIGATION_RESULT" "actionArn")
    if [ -n "$MITIGATION_ACTION_ARN" ]; then
        echo "Mitigation Action ARN: $MITIGATION_ACTION_ARN"
        aws iot tag-resource --resource-arn "$MITIGATION_ACTION_ARN" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs 2>&1
 || true
    fi
else
    echo "WARNING: Could not validate mitigation action response, but action may
 have been created"
fi

CREATED_RESOURCES+=("Mitigation Action: EnableErrorLoggingAction")

# Step 5: Apply mitigation action to findings (if any)
if [ "$HAS_FINDINGS" = true ]; then
    echo ""
```

```
    echo "=================================================="
    echo "Step 5: Applying mitigation action to findings"
    echo "=================================================="

    MITIGATION_TASK_ID="MitigationTask-$(date +%s)"
    echo "Starting mitigation actions task with ID: $MITIGATION_TASK_ID"

    # Build target JSON
    TARGET_JSON=$(cat <<EOF
{
  "findingIds": ["$FINDING_ID"]
}
EOF
)

    if ! validate_json "$TARGET_JSON"; then
        echo "ERROR: Invalid target JSON"
        exit 1
    fi

    # Build audit check to actions mapping JSON
    AUDIT_CHECK_MAPPING=$(cat <<EOF
{
  "LOGGING_DISABLED_CHECK": ["EnableErrorLoggingAction"]
}
EOF
)

    if ! validate_json "$AUDIT_CHECK_MAPPING"; then
        echo "ERROR: Invalid audit check mapping JSON"
        exit 1
    fi

    MITIGATION_TASK_RESULT=$(aws iot start-audit-mitigation-actions-task \
      --task-id "$MITIGATION_TASK_ID" \
      --target "$TARGET_JSON" \
      --audit-check-to-actions-mapping "$AUDIT_CHECK_MAPPING" --output json 2>&1)
 || true

    if ! check_error "$MITIGATION_TASK_RESULT"; then
        echo "WARNING: Failed to start mitigation actions task, continuing..."
    else
        echo "Mitigation actions task started successfully"
        CREATED_RESOURCES+=("Mitigation Task: $MITIGATION_TASK_ID")
```

```
    fi
else
    echo ""
    echo "=================================================="
    echo "Step 5: Skipping mitigation action application (no findings)"
    echo "=================================================="
fi

# Step 6: Set up SNS notifications (optional)
echo ""
echo "=================================================="
echo "Step 6: Setting up SNS notifications"
echo "=================================================="

# Check if SNS topic already exists
SNS_TOPICS=$(aws sns list-topics --output json 2>&1) || true
TOPIC_ARN=""
if validate_json "$SNS_TOPICS"; then
    TOPIC_ARN=$(echo "$SNS_TOPICS" | jq -r '.Topics[] | select(.TopicArn |
 contains("IoTDDNotifications")) | .TopicArn' 2>/dev/null | head -1 || echo "")
fi

if [ -n "$TOPIC_ARN" ]; then
    echo "SNS topic 'IoTDDNotifications' already exists, using existing topic..."
    echo "Topic ARN: $TOPIC_ARN"
else
    echo "Creating SNS topic for notifications..."
    SNS_RESULT=$(aws sns create-topic --name "IoTDDNotifications" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs --
output json 2>&1) || true

    if ! check_error "$SNS_RESULT"; then
        echo "WARNING: Failed to create SNS topic, continuing..."
        SNS_RESULT=""
    else
        TOPIC_ARN=$(extract_json_value "$SNS_RESULT" "TopicArn")
        if [ -n "$TOPIC_ARN" ]; then
            echo "SNS topic created with ARN: $TOPIC_ARN"
            CREATED_RESOURCES+=("SNS Topic: IoTDDNotifications")
        fi
    fi
fi

if [ -n "$TOPIC_ARN" ]; then
```

```
    echo "Updating audit configuration to enable SNS notifications..."

    # Build SNS notification configuration JSON
    SNS_CONFIG=$(cat <<EOF
{
  "SNS": {
    "targetArn": "$TOPIC_ARN",
    "roleArn": "$AUDIT_ROLE_ARN",
    "enabled": true
  }
}
EOF
)

    if ! validate_json "$SNS_CONFIG"; then
        echo "ERROR: Invalid SNS configuration JSON"
        exit 1
    fi

    SNS_UPDATE_RESULT=$(aws iot update-account-audit-configuration \
      --audit-notification-target-configurations "$SNS_CONFIG" 2>&1) || true

    if ! check_error "$SNS_UPDATE_RESULT"; then
        echo "WARNING: Failed to update audit configuration for SNS
 notifications"
    else
        echo "SNS notifications enabled successfully"
    fi
else
    echo "Skipping SNS configuration due to topic creation failure"
fi

# Step 7: Enable AWS IoT logging
echo ""
echo "================================================="
echo "Step 7: Enabling AWS IoT logging"
echo "================================================="

echo "Setting up AWS IoT logging options..."

LOGGING_RESULT=$(aws iot set-v2-logging-options \
  --role-arn "$LOGGING_ROLE_ARN" \
  --default-log-level "ERROR" 2>&1) || true
```

```
if ! check_error "$LOGGING_RESULT"; then
    echo "V2 logging setup failed, trying v1 logging..."

    V1_LOGGING_CONFIG=$(cat <<EOF
{
  "roleArn": "$LOGGING_ROLE_ARN",
  "logLevel": "ERROR"
}
EOF
)

    if ! validate_json "$V1_LOGGING_CONFIG"; then
        echo "ERROR: Invalid v1 logging configuration JSON"
        exit 1
    fi

    LOGGING_RESULT_V1=$(aws iot set-logging-options \
      --logging-options-payload "$V1_LOGGING_CONFIG" 2>&1) || true

    if ! check_error "$LOGGING_RESULT_V1"; then
        echo "WARNING: Failed to set up AWS IoT logging with both v1 and v2
 methods, continuing..."
    else
        echo "AWS IoT v1 logging enabled successfully"
    fi
else
    echo "AWS IoT v2 logging enabled successfully"
fi

# Verify logging is enabled
echo "Verifying logging configuration..."
LOGGING_CONFIG=$(aws iot get-v2-logging-options --output json 2>&1) || true
if [ -n "$LOGGING_CONFIG" ] && ! check_error "$LOGGING_CONFIG" && validate_json
 "$LOGGING_CONFIG"; then
    echo "Logging configuration verified"
    echo "$LOGGING_CONFIG" | jq '.' 2>/dev/null || echo "Configuration retrieved
 but could not display details"
else
    echo "Could not verify logging configuration, but setup completed"
fi

# Script completed successfully
echo ""
echo "================================================="
```

```
echo "AWS IoT Device Defender setup completed successfully!"
echo "================================================="
echo "The following resources were created:"
for resource in "${CREATED_RESOURCES[@]+"${CREATED_RESOURCES[@]}"}"; do
    echo "- $resource"
done
echo ""

# Cleanup phase
echo "=========================================="
echo "CLEANUP"
echo "=========================================="
echo "Starting automatic cleanup of resources..."
echo "Waiting 10 seconds before cleanup to allow resource stabilization..."
sleep 10

# Disable AWS IoT logging
echo "Disabling AWS IoT logging..."

DISABLE_V2_RESULT=$(aws iot set-v2-logging-options \
  --default-log-level "DISABLED" 2>&1) || true

if check_error "$DISABLE_V2_RESULT"; then
    echo "V2 logging disabled successfully"
else
    echo "Attempting v1 logging disable..."

    V1_DISABLE_CONFIG=$(cat <<'EOF'
{
  "logLevel": "DISABLED"
}
EOF
)

    DISABLE_V1_RESULT=$(aws iot set-logging-options \
      --logging-options-payload "$V1_DISABLE_CONFIG" 2>&1) || true

    if check_error "$DISABLE_V1_RESULT"; then
        echo "V1 logging disabled successfully"
    else
        echo "WARNING: Could not disable logging"
    fi
fi
```

```
# Delete mitigation action
echo "Deleting mitigation action..."
aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction" 2>&1 ||
 true

# Reset audit configuration
echo "Resetting IoT Device Defender audit configuration..."
RESET_AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":false}}'
aws iot update-account-audit-configuration \
  --audit-check-configurations "$RESET_AUDIT_CONFIG" 2>&1 || true

# Delete SNS topic
echo "Deleting SNS topic..."
if [ -n "${TOPIC_ARN:-}" ] && [ "$TOPIC_ARN" != "null" ]; then
    aws sns delete-topic --topic-arn "$TOPIC_ARN" 2>&1 || true
fi

# Clean up IAM roles with improved error handling
echo "Cleaning up IAM roles..."

cleanup_role() {
    local role_name=$1
    echo "Cleaning up role: $role_name"

    if aws iam get-role --role-name "$role_name" >/dev/null 2>&1; then
        ROLE_POLICIES=$(aws iam list-role-policies --role-name "$role_name" --
output json 2>&1 || echo '{"PolicyNames":[]}')
        if validate_json "$ROLE_POLICIES"; then
            while IFS= read -r policy_name; do
                if [ -n "$policy_name" ] && [ "$policy_name" != "null" ]; then
                    echo "  Deleting inline policy: $policy_name"
                    aws iam delete-role-policy \
                        --role-name "$role_name" \
                        --policy-name "$policy_name" 2>&1 || true
                fi
            done < <(echo "$ROLE_POLICIES" | jq -r '.PolicyNames[]' 2>/dev/null
 || echo "")
        fi

        ATTACHED_POLICIES=$(aws iam list-attached-role-policies --role-name
 "$role_name" --output json 2>&1 || echo '{"AttachedPolicies":[]}')
        if validate_json "$ATTACHED_POLICIES"; then
            while IFS= read -r policy_arn; do
                if [ -n "$policy_arn" ] && [ "$policy_arn" != "null" ]; then
```

```
                    echo "  Detaching managed policy: $policy_arn"
                    aws iam detach-role-policy \
                        --role-name "$role_name" \
                        --policy-arn "$policy_arn" 2>&1 || true
                fi
            done < <(echo "$ATTACHED_POLICIES" | jq -r
 '.AttachedPolicies[].PolicyArn' 2>/dev/null || echo "")
        fi

        echo "  Deleting role: $role_name"
        aws iam delete-role --role-name "$role_name" 2>&1 || true
    else
        echo "  Role $role_name does not exist or already deleted"
    fi
}

cleanup_role "AWSIoTDeviceDefenderAuditRole"
cleanup_role "AWSIoTLoggingRole"
cleanup_role "IoTMitigationActionErrorLoggingRole"

echo "Cleanup completed successfully"

echo ""
echo "Script execution completed at $(date)"
echo "Log file: $LOG_FILE"
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - AttachRolePolicy
  - CreateMitigationAction
  - CreateRole
  - CreateTopic
  - DeleteAccountAuditConfiguration
  - DeleteMitigationAction
  - DeleteRole
  - DeleteRolePolicy
  - DeleteTopic
  - DescribeAccountAuditConfiguration
  - DescribeAuditTask

- DetachRolePolicy
- GetLoggingOptions
- GetRole
- GetV2LoggingOptions
- ListAuditFindings
- ListAuditMitigationActionsTasks
- ListMitigationActions
- ListRolePolicies
- ListTopics
- PutRolePolicy
- SetLoggingOptions
- SetV2LoggingOptions
- StartAuditMitigationActionsTask
- StartOnDemandAuditTask
- UpdateAccountAuditConfiguration

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

開始使用物聯網簡訊

以下程式碼範例顯示做法：

- 建立 IoT 資源
- 設定您的裝置
- 執行範例應用程式
- 清除資源

Bash

AWS CLI 使用 Bash 指令碼

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在範例開發人員教學課程儲存庫中設定和執行。

```
#!/bin/bash

# AWS IoT Core Getting Started Script
# This script creates AWS IoT resources, configures a device, and runs a sample
 application

# Set up logging
LOG_FILE="iot-core-setup.log"
echo "Starting AWS IoT Core setup at $(date)" > $LOG_FILE

# Function to log commands and their outputs
log_cmd() {
    echo "$(date): Running command: $1" >> $LOG_FILE
    eval "$1" 2>&1 | tee -a $LOG_FILE
    return ${PIPESTATUS[0]}
}

# Function to check for errors
check_error() {
    if [ $1 -ne 0 ]; then
        echo "ERROR: Command failed with exit code $1" | tee -a $LOG_FILE
        echo "Please check the log file $LOG_FILE for details" | tee -a $LOG_FILE
        cleanup_on_error
        exit $1
    fi
}

# Function to cleanup resources on error
cleanup_on_error() {
    echo "Error encountered. Attempting to clean up resources..." | tee -a
 $LOG_FILE
    echo "Resources created:" | tee -a $LOG_FILE
```

```
    if [ ! -z "$CERTIFICATE_ARN" ]; then
        echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
        if [ ! -z "$POLICY_NAME" ]; then
            log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
        fi
        if [ ! -z "$THING_NAME" ]; then
            log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --
principal $CERTIFICATE_ARN"
        fi
        if [ ! -z "$CERTIFICATE_ID" ]; then
            log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID
 --new-status INACTIVE"
            log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"
        fi
    fi
    if [ ! -z "$THING_NAME" ]; then
        echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-thing --thing-name $THING_NAME"
    fi
    if [ ! -z "$POLICY_NAME" ]; then
        echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    fi
    if [ ! -z "$SHARED_POLICY_NAME" ]; then
        echo "Shared Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"
    fi
}

# Generate unique identifiers
RANDOM_SUFFIX=$(openssl rand -hex 4)
THING_NAME="MyIoTThing-${RANDOM_SUFFIX}"
POLICY_NAME="MyIoTPolicy-${RANDOM_SUFFIX}"
SHARED_POLICY_NAME="SharedSubPolicy-${RANDOM_SUFFIX}"
CERTS_DIR="$HOME/certs"

echo "================================================" | tee -a $LOG_FILE
echo "AWS IoT Core Getting Started" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "This script will:" | tee -a $LOG_FILE
echo "1. Create AWS IoT resources (policy, thing, certificate)" | tee -a
 $LOG_FILE
echo "2. Configure your device" | tee -a $LOG_FILE
```

```
echo "3. Set up for running the sample application" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Certificates Directory: $CERTS_DIR" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE

# Get AWS account ID
echo "Getting AWS account ID..." | tee -a $LOG_FILE
ACCOUNT_ID=$(log_cmd "aws sts get-caller-identity --query Account --output text")
check_error $?

# Get AWS region
echo "Getting AWS region..." | tee -a $LOG_FILE
REGION=$(log_cmd "aws configure get region")
check_error $?
if [ -z "$REGION" ]; then
    echo "AWS region not configured. Please run 'aws configure' to set your
 region." | tee -a $LOG_FILE
    exit 1
fi

echo "Using AWS Account ID: $ACCOUNT_ID and Region: $REGION" | tee -a $LOG_FILE

# Step 1: Create AWS IoT Resources
echo "" | tee -a $LOG_FILE
echo "Step 1: Creating AWS IoT Resources..." | tee -a $LOG_FILE

# Create IoT policy
echo "Creating IoT policy document..." | tee -a $LOG_FILE
cat > iot-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/test-*"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic"
      ]
    }
  ]
}
EOF

echo "Creating IoT policy: $POLICY_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-policy --policy-name $POLICY_NAME --policy-
document file://iot-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?

# Create IoT thing
echo "Creating IoT thing: $THING_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-thing --thing-name $THING_NAME"
check_error $?

THING_ARN=$(aws iot describe-thing --thing-name "$THING_NAME" --query 'thingArn'
 --output text)
aws iot tag-resource --resource-arn "$THING_ARN" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-iot-core-gs

# Create directory for certificates
echo "Creating certificates directory..." | tee -a $LOG_FILE
log_cmd "mkdir -p $CERTS_DIR"
check_error $?
```

```
# Create keys and certificate
echo "Creating keys and certificate..." | tee -a $LOG_FILE
CERT_OUTPUT=$(log_cmd "aws iot create-keys-and-certificate --set-as-active
 --certificate-pem-outfile $CERTS_DIR/device.pem.crt --public-key-outfile
 $CERTS_DIR/public.pem.key --private-key-outfile $CERTS_DIR/private.pem.key")
check_error $?

# Extract certificate ARN and ID
CERTIFICATE_ARN=$(echo "$CERT_OUTPUT" | grep "certificateArn" | cut -d'"' -f4)
CERTIFICATE_ID=$(echo "$CERTIFICATE_ARN" | cut -d/ -f2)

if [ -z "$CERTIFICATE_ARN" ] || [ -z "$CERTIFICATE_ID" ]; then
    echo "Failed to extract certificate ARN or ID" | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE

# Attach policy to certificate
echo "Attaching policy to certificate..." | tee -a $LOG_FILE
log_cmd "aws iot attach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Attach certificate to thing
echo "Attaching certificate to thing..." | tee -a $LOG_FILE
log_cmd "aws iot attach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"
check_error $?

# Download Amazon Root CA certificate
echo "Downloading Amazon Root CA certificate..." | tee -a $LOG_FILE
log_cmd "curl -s -o $CERTS_DIR/Amazon-root-CA-1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem"
check_error $?

# Step 2: Configure Your Device
echo "" | tee -a $LOG_FILE
echo "Step 2: Configuring Your Device..." | tee -a $LOG_FILE

# Check if Git is installed
echo "Checking if Git is installed..." | tee -a $LOG_FILE
```

```
if ! command -v git &> /dev/null; then
    echo "Git is not installed. Please install Git and run this script again." |
 tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Check if Python is installed
echo "Checking if Python is installed..." | tee -a $LOG_FILE
if ! command -v python3 &> /dev/null; then
    echo "Python 3 is not installed. Please install Python 3 and run this script
 again." | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Install AWS IoT Device SDK for Python
echo "Installing AWS IoT Device SDK for Python..." | tee -a $LOG_FILE
log_cmd "python3 -m pip install awsiotsdk"
check_error $?

# Clone the AWS IoT Device SDK for Python repository
echo "Cloning AWS IoT Device SDK for Python repository..." | tee -a $LOG_FILE
if [ ! -d "$HOME/aws-iot-device-sdk-python-v2" ]; then
    log_cmd "cd $HOME && git clone https://github.com/aws/aws-iot-device-sdk-
python-v2.git"
    check_error $?
else
    echo "AWS IoT Device SDK for Python repository already exists." | tee -a
 $LOG_FILE
fi

# Step 3: Get AWS IoT Endpoint
echo "" | tee -a $LOG_FILE
echo "Step 3: Getting AWS IoT Endpoint..." | tee -a $LOG_FILE

IOT_ENDPOINT=$(log_cmd "aws iot describe-endpoint --endpoint-type iot:Data-ATS --
query endpointAddress --output text")
check_error $?

echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE

# Create a shared subscription policy (optional)
echo "" | tee -a $LOG_FILE
```

```
echo "Creating shared subscription policy (optional)..." | tee -a $LOG_FILE

cat > shared-sub-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic",
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/\$share/*/test/topic"
      ]
    }
  ]
}
EOF

log_cmd "aws iot create-policy --policy-name $SHARED_POLICY_NAME --policy-
document file://shared-sub-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?
```

```
log_cmd "aws iot attach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Summary of created resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Setup Complete! Resources Created:" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Shared Subscription Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE
echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate Files Location: $CERTS_DIR" | tee -a $LOG_FILE
echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE

# Instructions for running the sample application
echo "" | tee -a $LOG_FILE
echo "To run the sample application, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 pubsub.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "To run the shared subscription example, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 mqtt5_shared_subscription.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key \\" | tee -a $LOG_FILE
echo "  --group_identifier consumer" | tee -a $LOG_FILE

# Ask if user wants to clean up resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "CLEANUP CONFIRMATION" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Do you want to clean up all created resources? (y/n): " | tee -a $LOG_FILE
read -r CLEANUP_CHOICE
```

```
if [[ $CLEANUP_CHOICE =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..." | tee -a $LOG_FILE

    # Detach policies from certificate
    echo "Detaching policies from certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
    log_cmd "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"

    # Detach certificate from thing
    echo "Detaching certificate from thing..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"

    # Update certificate status to INACTIVE
    echo "Setting certificate to inactive..." | tee -a $LOG_FILE
    log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE"

    # Delete certificate
    echo "Deleting certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"

    # Delete thing
    echo "Deleting thing..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-thing --thing-name $THING_NAME"

    # Delete policies
    echo "Deleting policies..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"

    echo "Cleanup complete!" | tee -a $LOG_FILE
else
    echo "Resources were not cleaned up. You can manually clean them up later." |
 tee -a $LOG_FILE
    echo "To clean up resources, run the following commands:" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
```

```
    echo "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE" | tee -a $LOG_FILE
    echo "aws iot delete-certificate --certificate-id $CERTIFICATE_ID" | tee -a
 $LOG_FILE
    echo "aws iot delete-thing --thing-name $THING_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $POLICY_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $SHARED_POLICY_NAME" | tee -a
 $LOG_FILE
fi

echo "" | tee -a $LOG_FILE
echo "Script execution completed. See $LOG_FILE for details." | tee -a $LOG_FILE
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的下列主題。
  - AttachPolicy
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreatePolicy
  - CreateThing
  - DeleteCertificate
  - DeletePolicy
  - DeleteThing
  - DescribeEndpoint
  - DetachPolicy
  - DetachThingPrincipal
  - GetCallerIdentity
  - UpdateCertificate

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 AWS IoT 搭配 AWS SDK 使用。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

AWS IoT 配額

您可以在 AWS 一般參考中找到 AWS IoT 配額的相關資訊。

- 如需 AWS IoT Core 配額資訊，請參閱AWS IoT Core 端點和配額。
- 如需 AWS IoT Device Management 配額資訊，請參閱AWS IoT Device Management 端點和配額。
- 如需 AWS IoT Device Defender 配額資訊，請參閱AWS IoT Device Defender 端點和配額。

AWS IoT Core 定價

您可以在AWS 行銷頁面和 AWS IoT Core 定價AWS 計算器中找到定價的相關資訊。

- 若要檢查 AWS IoT Core 定價資訊，請參閱 AWS IoT Core 定價。
- 若要預估架構師解決方案的成本，請參閱 AWS 定價計算器。

支援通訊協定的FAQs

除了現有的 AWS JSON 通訊協定之外， AWS IoT 安全通道服務正在新增對 Smithy RPC v2 CBOR 通訊協定的 SDK 支援。Smithy RPC v2 CBOR 通訊協定的效能高於大多數工作負載的 AWS JSON。AWS SDKs會優先考慮每種語言效能最佳的通訊協定。

什麼是 AWS JSON 和 Smithy RPC v2 CBOR 通訊協定，它們有何不同？

線路通訊協定可做為用戶端和 服務之間的中介裝置。 AWS 開發套件會透明地處理兩個通訊協定的用戶端和伺服器之間的序列化（將物件轉換為線路格式）和還原序列化（將線路格式轉換為物件）。

AWS JSON - JSON 是系統之間最廣泛使用和接受的通訊資料格式之一。此服務使用 AWS JSON 做為媒體，在 AWS SDK 用戶端和伺服器之間進行通訊。服務 API 操作的 HTTP 請求接受 JSON 格式的輸入、執行操作，並以 JSON 格式將回應傳回至 SDK 用戶端。

Smithy RPC v2 CBOR - Smithy RPC v2 CBOR（簡潔二進位物件表示法）是由 Amazon 開發的通訊協定，可提供更有效率的資料傳輸。它是一種資料格式，旨在保持訊息大小精簡，並允許格式演進，而無需在系統之間進行版本交涉。與 AWS JSON 的主要區別是 AWS JSON 使用人類可讀文字（如 `{"name": "John"}`），而 Smithy RPC v2 CBOR 使用二進位格式 (1 和 0 的序列）。這種二進位方法會建立較小的資料套件，以更快的速度通過網路，並縮短因電腦直接處理二進位資料而產生的序列化時間。

如何開始使用 Smithy RPC v2 CBOR 通訊協定？

若要開始使用，請使用最新的 AWS SDK/CLI 版本來使用此服務的 Smithy RPC v2 CBOR 通訊協定。開發套件將在可用時自動使用 Smithy RPC v2 CBOR 通訊協定。將您的 AWS SDK 升級至指定的版本或任何後續版本。

從 2026-07-23 開始，下列 AWS SDKs將使用 Smithy RPC v2 CBOR 通訊協定，而其他 SDKs 將繼續使用 AWS JSON 通訊協定：

- AWS 適用於 C++ 的 SDK
- AWS 適用於 Go V2 的 SDK
- AWS 適用於 Java 的 SDK 2.x
- AWS 適用於 .NET v4 的 SDK

- AWS PowerShell v5 的工具
- AWS 適用於 Swift 的 SDK
- AWS 適用於 Kotlin 的 SDK

啟用 Smithy RPC v2 CBOR 通訊協定有哪些風險？

在 SDKs 中新增對 Smithy RPC v2 CBOR 的支援，預期對客戶而言是完全透明且輕鬆的。不過，如果您使用 AWS SDK 的自訂實作，或自訂用戶端和 SDK AWS 的組合，取決於 AWS JSON 特定的序列化行為，則可能與 Smithy RPC v2 CBOR 通訊協定不相容。

如果我的解決方案不支援 Smithy RPC v2 CBOR，該怎麼辦？

若要使用 AWS JSON 通訊協定，請將 SDK 降級至 2026-07-23 之前發行的版本。

哪些 AWS 區域支援 Smithy RPC v2 CBOR 通訊協定？

提供此服務的所有AWS 區域都支援 Smithy RPC v2 CBOR 通訊協定。

升級至支援 Smithy RPC v2 CBOR 的 SDK 版本時，可以預期哪些延遲改善？

與 AWS JSON 相比，Smithy RPC v2 CBOR 的延遲降低高達 13%，承載大小降低高達 15%。二進位編碼可消除 JSON 文字剖析和字串逸出的額外負荷，進而降低網路傳輸時間和序列化成本。

AWS JSON 通訊協定是否會棄用？

AWS 舊版 SDKs 將繼續支援 JSON 通訊協定。此外，某些語言特定的 SDKs 將繼續使用 AWS JSON，因為它比 Smithy RPC v2 CBOR 更高效能，因為語言的可用 JSON 實作比其 CBOR 實作更快。

哪裡可以找到有關 Smithy RPC v2 CBOR 的詳細資訊？

您可以在 Smithy 文件中找到有關 AWS JSON 通訊協定在 AWS JSON 1.0 通訊協定和 AWS JSON 1.1 通訊協定的詳細資訊，以及在 Smithy 文件中找到有關 Smithy RPC v2 CBOR 在 Smithy RPC v2 CBOR 通訊協定的詳細資訊。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。